Automatic conversion of sound to the MIDI-format*

Johan Forsberg

Abstract

An experimental system for converting an audio signal into MIDI representation is presented. The system works in the frequency domain. A constant Q transform is used to compute spectrograms from which the partials are tracked. The partial trajectories are then parameterized in time and frequency resulting in a "pianola roll" representation, suitable for conversion to MIDI-code. The system performance is acceptable for music with reasonably well defined boundaries between notes. Typical errors which occur in the present state of development are placement of notes in the wrong octave, and generation of "ghost notes" which not belong to the score.

Introduction

Methods for converting performed music into some form of music notation is a recurring area of research in musical acoustics. In the late 70´s, a number of researchers worked on this problem. Piszczalski & Galler (1977) developed a system for transcription of monophonic music all the way to common music notation. Their system used a DFT to convert the acoustic signal to the frequency domain, after which the fundamental frequencies were detected. A pattern matching approach was then used to find the start and end of notes, and a score was generated. The system was limited to instruments with a strong fundamental, and had some problems with determining the correct length of notes and pauses, but otherwise it performed reasonably well.

Another successful early system was developed by James A Moorer (1977) at Stanford University. He managed to transcribe guitar and violin duets into common music notation. His system worked directly on the time domain data, using a series of complex filtering functions to determine the partials. The partials were then grouped together and a score was printed. This system generated very accurate scores, but could not resolve notes sharing common partials, nor notes with vibrato.

After a brief revival in the 80´s (Chafe et al., 1982) not much work on automatic transcription of music has been reported. The exceptions are some studies of transcription of keyboard performances, in which the problem of pitch tracking is avoided by identifying which keys are engaged (Martin, 1996).

The main drawback of most systems described in the literature seems to be that they are applicable only to a narrow range of cases. Some systems require the instruments to be known beforehand, some can not handle polyphonic music, and some are very sensitive to noise.

The present system is an attempt of using a more general approach that can handle a wide range of musical instruments and performing styles, at the expense of a limited accuracy in some cases. The program, having been written in limited amount of time, can not handle strictly polyphonic music. It allows for several simultaneously sounding notes, but the note onsets must be slightly separated in time. For a detailed description of the system, including the source code, see Forsberg (1997).

General outline

The main processing steps of the system are illustrated in Figure 1, showing the way from the audio signal to a musical score for MIDI representation.

---

* Summary of thesis work for MSc degree.

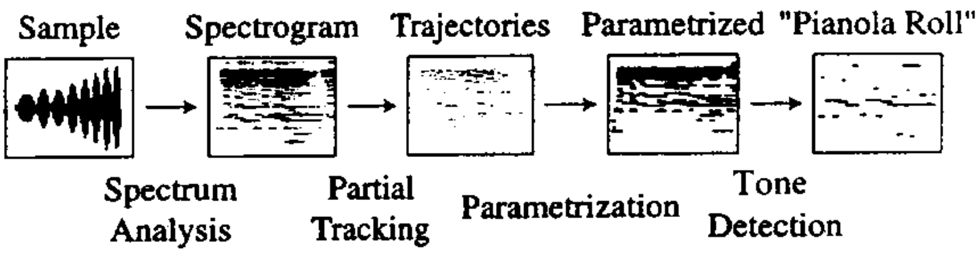


Figure 1. Overview of the system for automatic transcription of music showing the processing steps and data representation.

The guiding design principles of the system have been

- to reduce complexity wherever possible by choosing simple, easily understood algorithms
- to adhere to some of the basic characteristics of hearing
- to allow processing in a streamed fashion, so that output data starts being written before the input data has ceased.

The purpose of the third point is to allow the program to produce continuous output data from a live microphone input with a minimum of lag, provided that the program runs on a sufficiently fast computer.

1. Spectrum analysis

The sampled audio data (often from a CD) are analysed at constant time intervals, generating a stream of spectra, typically at a rate of 100 spectra/s. The frequency range is divided into a number of frequency bins. In each bin, a value is stored which represents the total energy of the signal within the frequency boundaries of the bin. The spacing of the bins should be no more than a semitone throughout the fundamental frequency range of the analysed music, preferably less. The choice of spectral transform is important as will be discussed below.

2. Partial tracking

The spectra are analysed for local maxima, corresponding to partials. The evolution of these maxima are traced over time, reducing the influence of noise and spurious signals caused by the frequency transform. The output is a set of frequency and magnitude trajectories, describing the development of the partials over time. In some difficult cases the magnitude may be disregarded, and only the frequency trajectory is used in the following processing.

3. Parameterization

The partial trajectories produced in stage two are approximated by some simpler functions, such as splines or straight lines. Sinusoids may be used when tracking sounds with pronounced vibrato.

4. Tone detection

The parameterised partial trajectories are combined into tones, either assuming perfectly harmonic sounds, or allowing a certain degree of inharmonicity.

5. Scoring

The result from stage four is basically a ’pianola roll’ representation of the music, which easily can be described in MIDI-code. In order to generate a score in common music notation a lot of additional processing is needed. The most troublesome part is the quantization of the duration of the notes into note values. Extraction of meter and key signature are also possible (Chafe et al., 1982; Longuet-Higgins, 1976). These parts of the transcription problem were, however, not within the scope of the present study.

Description of the processing

The main processing steps of the system are discussed in the following sections. The processing of a melody through the system is illustrated by a one-octave diatonic major scale, played with a synthesised saw-tooth wave. In the spectrograms and partial tracking images (Figures 2, 4, and 7), the amplitude range from full white to black is 60 dB.

1. Spectrum analysis

The use of a standard FFT for the spectral analysis is hardly possible, as this will not give high enough resolution both in time and frequency. With an output in real time, processing is limited to a window length which equals one full period of the lowest frequency the algorithm should detect. A low spectrum rate allowing for a longer window increases the frequency resolution while decreasing time resolution. Another problem is that the frequency bins of the standard FFT are linearly spaced, while musical pitch is better mapped on a logarithmic scale. Using a FFT-analysis, the number of frequency bins will double for each octave. This causes problems when analysing music with notes spanning many octaves, such as piano music. For example, by using 48 bins for the lowest octave of the piano (A_0 - A_1), the highest octave (A_6 - A_7) will contain $48 \cdot 2^6 = 3072$ bins. This is a clear waste of memory and processing power.

The solution is to use an alternative frequency transform such as the constant Q transform (Brown, 1991), or a related wavelet transform with a constant number of frequency bins in each octave. The present system uses the original constant Q transform, which as an added bonus has the frequency bins logarithmically spaced, further simplifying the processing.

The constant Q transform

The constant Q transform has a constant, user-configurable ratio of frequency versus resolution, $Q = f/df$ (Brown, 1991). For quarter-note resolution (50 cent), this ratio is

$$\left(\frac{1}{2^{1/24}-1}\right) \approx 34$$

The constant Q transform is derived from the discrete short time Fourier transform

$$X[k] = \sum_{n=0}^{N-1} W[n]\, x[n]\, e^{-j2\pi kn/N} \qquad (1)$$

where *x[n]* denotes the *n*:th sample of the signal and $2\pi k/N$ is the digital frequency.

For a constant Q variation, each spectral component is calculated over a window of different length, which makes *N* dependent on frequency, and thus *k*. The digital frequency of the *k*:th component becomes $2\pi Q/N[k]$. The window function *W[n]* has the same shape for each bin, but different length. For two bins spaced an octave apart, the window length for the lower bin is twice that of the upper. Since the number of terms of the sum varies with *k*, we must also normalise by dividing the sum by *N[k]*.

Equation (1) then becomes

$$X[k] = \frac{1}{N[k]} \sum_{n=0}^{N[k]-1} W[k,n]\, x[n] e^{-j2\pi Qkn/N[k]} \qquad (2)$$

Further refinements of this formula are possible. In particular, it can be expressed in terms of regular fourier transforms and implemented using efficient DFTs (Brown, 1992). The present system relies on a brute force implementation, directly applying Eq. (2) to the sampled data. In order to reduce computational complexity, the algorithm goes through the following steps.

1. Calculation of the constant Q transform for the top octave of the spectrum.
2. Filtering out the top octave by multiplying the window with an approximated *sin(x)/x* kernel.
3. Downsampling the data by a factor two and repeat.

The amount of calculation necessary to extract the top octave is only half of what it would cost to extract the next lower octave. This means that a substantial amount of computer processing is eliminated.

The window function used is a Hamming function

$$W[k,n] = \alpha + (1-\alpha)\cos(2\pi n/N[k]) \qquad (3)$$

where $\alpha = 25/46$ and

$$0 \le n \le N[k]-1$$

This window shape is optimized to give a minimal amount of spill-over between adjacent frequency bins.

The input to the spectrum analysis program is assumed to be a mono stream of signed 16-bit numbers at 22 050 Hz. Default parameters are a frequency resolution of 48 bins per octave (25 cents spacing between adjacent bins), and a time resolution of 100 spectra/s. The output is a series of magnitude spectra with magnitude values converted to dB. The spectra can be presented as spectrograms (Figure 2).

2. Partial tracking

This part of the system handles partial detection and tracking. It also generates a pair of special channels that are useful for the following stages. These are the total intensity of each spectrum, and a measure of the ”clarity” of the spectrum. The clarity index *cl* is calculated as

$$cl = \frac{\left|\frac{dX[k]}{dk}\right|}{|X[k]|} \qquad (4)$$

where *X[k]* is the content of the *k*:th bin of the spectrum *X*. This measure gives an estimation of the definition of the spectral peaks. In contrast to sounds with a clear pitch for which the energy is concentrated to clearly defined partial peaks, noisy and percussive sounds have the spectral energy distributed over a wide frequency range. This makes the average derivative of the spectral curve lower in relation to overall intensity, resulting in a low clarity measure.

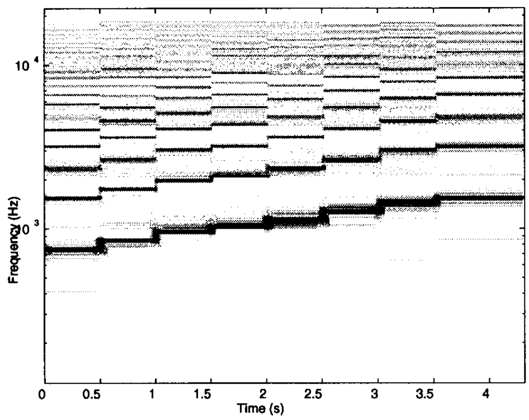


Figure 2. Spectrum analysis of a saw-tooth scale.

Partial detection

Partials will appear as local maxima (peaks) in the spectrum. However, all peaks do not correspond to real partials. Some of them are due to “noise” in the music (attack components, spurious sounds), and some are “ghost” partials, generated in the spectral analysis. On the other hand, all real partials will not be continuously present as pronounced peaks in a series of spectra over some time. In order to find which spectral peaks are important, the system compares the height of a peak with the neighbouring local minima (valleys). If this difference is less than a threshold value (typically 40 dB), the peak is discarded. One problem in this step is that noise tend to make the spectral peaks jagged, leading to difficulties in detecting the surrounding valleys. The solution was to apply a smoothing function

$$\bar{X}[k] = \frac{X[k-1] + 2X[k] + X[k+1]}{4} \qquad (5)$$

to the spectrum values *X [k].* The effect of the smoothing is illustrated in Figure 3.

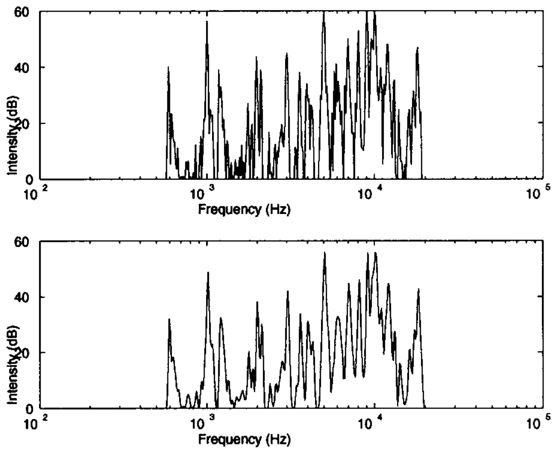


Figure 3. Spectrum before and after smoothing. The spectrum data represents a time window of 10 ms.

Partial tracking

The tracking program attempts to tie the detected spectral peaks together into a set of partial trajectories, both in frequency and magnitude (Figure 4). This is essentially a variation of the line detection problem in computer vision. The program implements a simple heuristic approach to this problem. For each peak in the spectrum, the frequency is compared with the frequencies of the spectral peaks in the previous time step. If the two frequencies match within a certain small frequency interval, the two peaks are connected in time. If not, the algorithm looks two time steps back, then three, and so on up to a maximum value. This allows partials to have small gaps, and a certain amount of vibrato, glissando and frequency jitter. If there is no peak to connect to, a new partial is considered to have begun. In this way, a list of ”suspected” partials, their frequencies and amplitudes are generated for each time step.

The results are not written to the output file at once, but entered into a buffer of approximately 0.5 s length. When a new partial reaches the end of the buffer (half a second after it was first detected), the prominence of the partial is checked. If the duration of the partial was too short, or if the magnitude was too low,

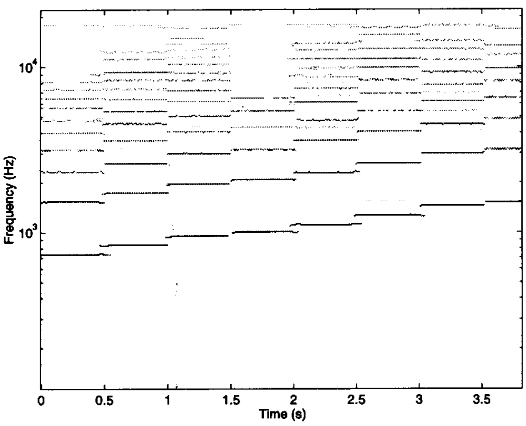


Figure 4. Partial tone tracking of the saw-tooth scale.

it is discarded, otherwise it is added to the output file. This reduces the amount of false partials caused by noise and the spectrum analysis drastically. However, sometimes weak partials are removed along with the false ones, so some fine tuning of the parameters of this program is always necessary.

The output of this program is a triplet for each partial and time step (partial identity, frequency, and magnitude), and a pair of special channels as discussed above (total intensity, clarity).

3. Parameterization

The parameterization program analyses the generated trajectories representing partial frequencies and magnitude levels, and attempts to fit them to simple, parameterized curves. Because an exact handling of pitch bend is not covered by the MIDI standard, the curve type selected was a straight horizontal line.

Overlapping notes cause particular problems. The frequency trajectories are traced through a series of time steps, and an average value is calculated. If the instantaneous frequency differs from this average by more than a certain minimum value, the partial is considered to have ended, and a new one with slightly different frequency to have begun. If the amplitude of the partial suddenly rises high over the average, the partial is considered to have ended, and a new one with the same frequency to have begun. The reason for this approach, is that if the music is played legato, or with a significant amount of reverberation, partials of two successive notes with similar frequencies tend to merge.

Once a partial has been traced from start to end, a parameterized version is generated with a frequency equal to the average frequency, and a magnitude equal to the average magnitude (Figure 5). The start and end of the partial are stored as separate timed events in a time sorted output buffer. In this step, time units are converted from basic time slice numbers to milliseconds, and frequency bin numbers to Hz.

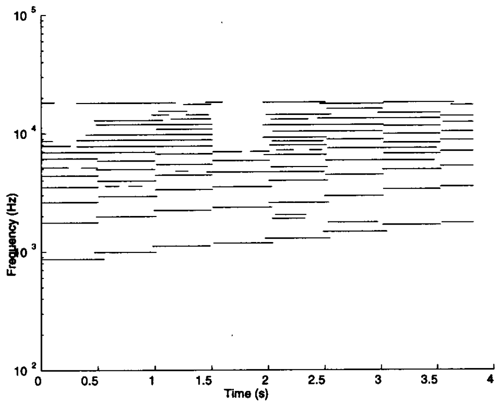


Figure 5. Partial tone parameterization of the saw-tooth scale.

4. Tone detection

This part of the system attempts to find sets of partials that start at approximately the same time, and group them together to a tone (score note) by calculating a common fundamental.

Fundamental detector

All partials of a note are supposed to start within a relatively small time interval, henceforth called an ´onset window.' 'Note start' events are read from the list of parameterized partial events until the time difference between the first and last events is larger than the width of the onset window. This list of events is supposed to comprise the partials of the tone, and possibly some false partials. Each partial is then analysed in turn. The partial under analysis (hereafter written in italics, *partial*) is first assumed to be the fundamental. The other partials are then checked in turn to see if their frequencies have an integer relationship to this assumed fundamental frequency, within some small tolerance (about 2%). If this is the case, an accumulated score *sc* is assigned to *partial* according to the formula

$$sc = \sum k^{\alpha} \qquad (6)$$

where k is the frequency relation to the assumed fundamental *partial* (k = 1,2,3,4…) and α is a negative number.

For instance, with $\alpha = -1$ the contributions from the fundamental, a second harmonic, and a third harmonic will be 1, 1/2, and 1/3, respectively. Then, the next higher partial from the list is assumed to be the fundamental (*partial*) and the process is repeated.

In a following step, *partial* is considered to be the second harmonic of a tone with a missing fundamental, and the entire process is repeated for all partials on the list. Finally, *partial* is assumed to be the third harmonic of a tone with both a missing fundamental and a missing second harmonic and the entire process is repeated.

If the maximum score achieved in these computations is larger than a threshold value, the frequency of the corresponding *partial* (or half or a third of the frequency value depending on the case) is assumed to be the fundamental frequency. If not, the first partial on the list is thrown away, and the list is updated with a new partial at the end, shifting the onset window to a later time point.

If a fundamental is found, the set of partials on the list is accepted as a tone, and the start and end times are calculated as the average of the starting and ending times of the partials. A total magnitude is calculated as the sum of the magnitudes of the partials, and the tone is put into a queue for being written as MIDI code. The score system described above with a power function and a negative exponent, allows tones with missing partials to be accepted, as long as the lowest harmonics are present. It also allows tones with weak or missing low harmonics to be accepted if they have a large amount of higher harmonics present.

5. Scoring

After the tone detection stage is finished, the resulting 'note start'/'note end' events are written to an output MIDI file, consisting of a single track and containing only "Note On" and "Note Off" MIDI events. The final score can be viewed as a pianola roll (Figure 6).

Results

The system has been tested on collection of widely differing signals, ranging from simple examples as the scale played with a synthetic

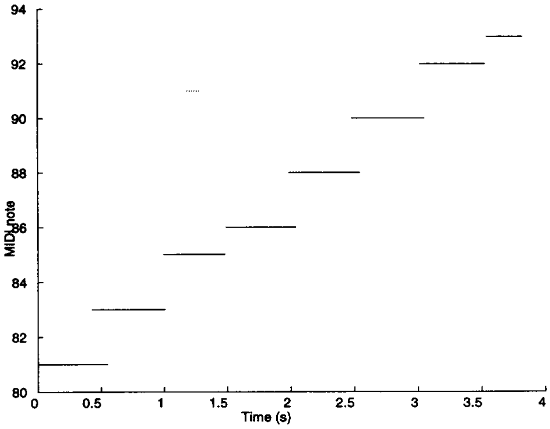


Figure 6. MIDI scoring ("pianola roll") of the saw-tooth scale. The short high-pitched note above the third scale note is a false note.

saw-tooth wave (Figs. 2 - 6) to excerpts from symphonic orchestra music with a strong melody line. This selection showed rather well the limits of the system. For instance, even in the simple example with the saw-tooth wave the frequency spectra are not as clear as desirable. This is probably because of the down-sampling approach in the implementation of the constant Q transform, together with some noise-inducing behaviour of the Hamming window. Nevertheless, the system managed to identify all eight notes of the scale correctly, but added one false note (Figure 6).

Some types of synthesiser music were also handled convincingly by the system. The suite of images in Figure 7 shows the analysis of an excerpt of synthesiser music with relatively simple structure. The final MIDI representation places all notes correctly except for one note which is a half-note too high. However, there are several high, false notes as well. This is an artifact created by the scoring program. When partials are grouped together to form a tone, any "left over" partials (i.e. partials which not have a integer frequency relation to the fundamental) are left unprocessed. If by chance some of these left-over partials have a integer relationship between their frequencies, the probability is high that they will be considered to be a simultaneously sounding note. This problem should be relatively easy to solve by improving the partial-combination algorithm.

The tests with the saw-tooth signal and the synthesiser music showed that the system performs fairly well on melodies that either have marked boundaries between tones, or do not repeat notes with the same fundamental frequency too closely in time. Short notes tend to have a short overall duration in relation to their decay time, which makes it hard to determine where they end, and they may easily blur into the following notes.

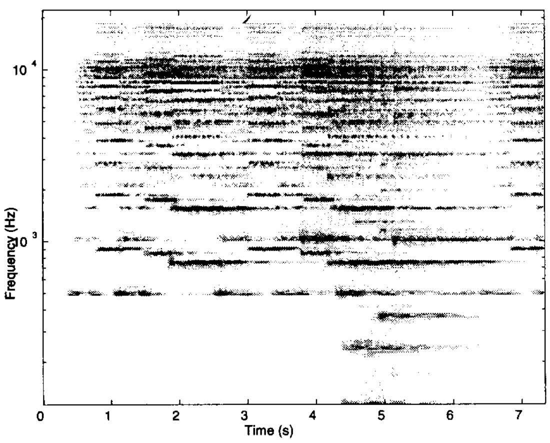
When tested on orchestra music, the system performed much more poorly. The ”convert-one-octave-then-downsample” approach produced heavy banding effects in the spectra, obscuring partials. Also, rich instrument spectra in combination with fast notes and long reverberation caused partials of unrelated notes to merge and to be interpreted as single partials.

Further, instruments with weak second and third partials were often interpreted as sounding two octaves higher than in reality, even if they had a strong fundamental. This is due to the power law scoring system which gives large weight to the lowest partials, and much less to the higher. For example, partials four, five and six give larger contributions to the accumulated score when the fourth partial is assumed to be the fundamental, than with the lowest partial as fundamental. A strong fundamental is thus not a guarantee for a correct calculation.

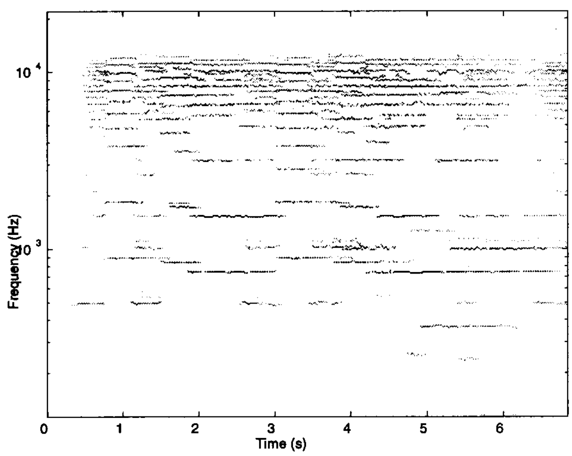
Conclusions

The system performs well on simple types of music with well defined boundaries between notes, such as some types of synthesiser music.

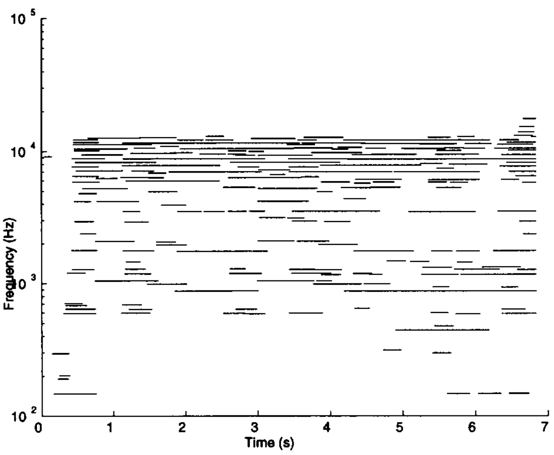
For traditional ensemble music with acoustical instruments there is still a lot of room for improvement. The particular implementation of the constant Q transform used (“convert one



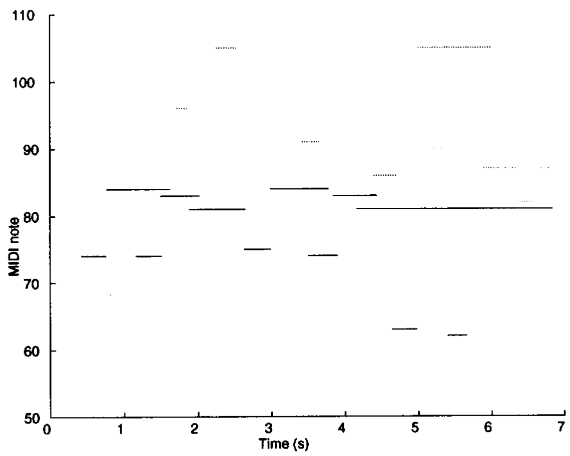
(a) Spectrum analysis



(b) Partial tone tracking



(c) Parameterisation



(d) MIDI score.

Figure 7. Analysis of an excerpt of synthesizer music (from "New Life" performed by Depeche Mode) showing (a) spectrum analysis, (b) partial tone tracking, (c) partial tone parameterization, and (d) MIDI score. The faint lines above MIDI note number 85 are false notes.

octave and downsample") in combination with reverberation tend to hide partials and merge notes. Still, the approach is interesting and seems worthwhile to try to develop. It is worth remembering that apart from the many practical applications of automatic transcription systems, the mistakes generated by the systems gives valuable insights into our perception of musical sounds.

Acknowledgements

The author is indebted to Johan Liljencrants for providing the source code for the constant Q transform, and to Anders Askenfelt who supervised the thesis work.

References

Brown J (1991). Calculation of a constant Q spectral transform. *J Acoust Soc Am* 89: 425-434.

Brown J (1992). An efficient algorithm for the calculation of a constant Q transform. *J Acoust Soc Am* 92: 2698-2701.

Brown J (1992). Musical fundamental frequency tracking using a pattern recognition method. *J Acoust Soc Am* 92: 1394-1402.

Chafe C, Mont-Reynaud B & Rush L (1982). Toward an intelligent editor of digital audio: Recognition of musical constructs, *Computer Music Journal* 6/1: 30-41.

Forsberg J (1997). Automatic conversion of sound to the MIDI-format. *MSc thesis report*, Department of Speech, Music and Hearing, Royal Institute of Technology, Stockholm.

Longuet-Higgins HC (1976). Perception of melodies. *Nature* 263/5579: 646-653.

Martin K (1996). Automatic transcription of simple polyphonic music. *J Acoust Soc Am* 100/4: 2813(A).

Moorer J (1977). On the transcription of musical sound by computer. *Computer Music Journal*, Nov 1977; 32-38.

Piszczalski M & Galler BA (1977). Automatic music transcription, *Computer Music Journal*, Nov 1977; 24-31.