Real-time audio analysis tools for Pd and MSP

Miller S. Puckette, UCSD (msp@ucsd.edu)
Theodore Apel, CRCA, UCSD (tapel@ucsd.edu)
David D. Zicarelli, Cycling74 (www.cycling74.com)

reprinted from *Proceedings*, ICMC 98

Abstract

Two "objects," which run under Max/MSP or Pd, do different kinds of real-time analysis of musical sounds. *Fiddle* is a monophonic or polyphonic maximum-likelihood pitch detector similar to Rabiner's, which can also be used to obtain a raw list of a signal's sinusoidal components. *Bonk* does a bounded-Q analysis of an incoming sound to detect onsets of percussion instruments in a way which outperforms the standard envelope following technique. The outputs of both objects appear as Max-style control messages.

1 Tools for real-time audio analysis

The new real-time patchable software synthesizers have finally brought audio signal processing out of the ivory tower and into the homes of working computer musicians. Now audio can be placed at the center of real-time computer music production, and MIDI, which for a decade was the backbone of the electronic music studio, can be relegated to its appropriate role as a low-bandwidth I/O solution for keyboards and other input devices. Many other sources of control "input" can be imagined than are provided by MIDI devices. This paper, for example, explores two possibilities for deriving a control stream from an incoming audio stream.

First, the sound might contain quasi-sinusoidal "partials" and we might wish to know their frequencies and amplitudes. In the case that the audio stream comes from a monophonic or polyphonic pitched instrument, we would like to be able to determine the pitch(es) and loudness(es) of the components. It's clear that we'll never have a perfect pitch detector, but the `fiddle` object described here does fairly well in some cases.

For the many sounds which don't lend themselves to sinusoidal decomposition, we can still get useful information from the overall spectral envelope. For instance, rapid changes in the spectral envelope turn out to be a much more reliable indicator of percussive attacks than are changes in the overall power reported by a classical envelope follower. The `bonk` object does a bounded-Q filterbank of an incoming sound and can either output the raw analysis or detect onsets which can then be compared to a collection of known spectral templates in order to guess which of several possible kinds of attack has occurred.

The `fiddle` and `bonk` objects are low tech; the algorithms would be easy to re-code in another language or for other environments from the ones considered here. Our main concern is to get predictable and acceptable behavior using easy-to-understand techniques which won't place an unacceptable computational load on a late-model computer.

Some effort was taken to make `fiddle` and `bonk` available on a variety of platforms. They run under Max/MSP (Macintosh), Pd (Wintel, SGI, Linux) and `fiddle` also runs under FTS (available on several platforms.) Both are distributed with source code; see `http://man104nfs.ucsd.edu/~mpuckett/` for details.

2 Analysis of discrete spectra

Two problems are of interest here: getting the frequencies and amplitudes of the constituent partials of a sound, and then guessing the pitch. Our program follows the ideas of [Noll 69] and [Rabiner 78]. Whereas the earlier `pitch~` object reported in [Puckette 95] departs substantially from the earlier approaches, the algorithm used here adhere more closely to them.

First we wish to get a list of peaks with their frequencies and amplitudes. The incoming signal is broken into segments of N samples with N a power of two typically between 256 and 2048. A new analysis is made every $N/2$ samples. For each analysis the N samples are zero-padded to $2N$ samples and a rectangular-window DFT is taken. An interesting trick reduces the computation time roughly in half for this setup; see the source code to see how this is done.

If we let $X[k]$ denote the zero-padded DFT, we can do a three-point convolution in the frequency domain to get the Hanning-windowed DFT:

$$X_H[k] = X[k]/2 - (X[k+2] + X[k-2])/4$$

Any of the usual criteria can be applied to identify peaks in this spectrum. We then go back to the non-windowed spectrum to find the peak frequency using the phase vocoder with hop 1:

$$\omega = \frac{\pi}{N}\left(k + re\left[\frac{X[k-2] - X[k+2]}{2X[k] - X[k-2] - X[k+2]}\right]\right).$$

This is a special case of a more general formula derived in [Puckette 98]. The amplitude estimate is simply the windowed peak strength at the strongest bin, which because of the zero-padding won't differ by more than about 1 dB from the true peak strength. The phase could be obtained in the same way but we won't bother with that here.

2.1 Guessing fundamental frequencies

Fundamental frequencies are guessed using a scheme somewhat suggestive of the maximum-likelihood estimator. Our "likelihood function" is a non-negative function $\mathcal{L}(f)$ where f is frequency. The presence of peaks at or near multiples of f increases $\mathcal{L}(f)$ in a way which depends on the peak's amplitude and frequency as shown:

$$\mathcal{L}(f) = \sum_{i=0}^{k} a_i t_i n_i$$

where k is the number of peaks in the spectrum, a_i is a factor depending on the amplitude of the ith peak, t_i depends on how closely the ith peak is tuned to a multiple of f, and n_i depends on whether the peak is closest to a low or a high multiple of f. The exact choice of how these factors should depend on f and the peak's frequency and amplitude is a subject of constant tinkering.

For monophonic pitch estimation, we simply output the value of f whose "likelihood" is highest. For polyphonic pitch estimation, we successively take the values of f of greatest likelihood which are neither multiples nor submultiples of a previous one.

In all cases, an additional criterion is used to make a pitched/nonpitched decision since $\mathcal{L}(f)$ will always have a maximum, even when no pitch is present. Our criterion is that there either be at least four peaks present or else that the fundamental be present and the total power of the contributing peaks be at least a hundredth of the signal power.

2.2 Object design

The `fiddle` object has a signal input and a varying number of control outputs depending on its creation arguments:

```
fiddle [npoints] [npitches]
 [npeaks-analyzed] [npeaks-output]
```

where `npoints` gives the (power of two) number of points in each analysis window, `npitches` gives the number of separate pitches to report (one by default), `npeaks-analyzed` gives the maximum number of peaks to consider in determining pitch (default 20) and `npeaks-output` gives the number of peaks which are to be output raw. Setting `npitches` to zero suppresses pitch estimation (and saves computation time).

The outlets, from left to right, are:

- a floating-point pitch which is output when a new, stable note is found
- a bang which is output conditionally on "attacks", whether or not a pitch is found
- from 0 to 3 lists, each giving the pitch and loudness of a pitch track
- the continuous signal power in dB
- a list, which iteratively sends triples giving each peak's index, frequency, and amplitude.

The following messages print or set the fudge mix:

print print out the parameters controlled by the following messages:

amp-range (low) (high) set the (low) and (high) amplitude thresholds in dB. Note-on detection requires that the signal exceed (high); if a pitch track's strength goes below (low) it is dropped.

reattack (time) (dB) Pitch tracks whose strength increases by more than (dB) within (time) msec output a new "note" message.

vibrato (time) (half-tones) warn `fiddle` that the instrument is capable of vibrato. New notes will not be reported until (time) msec have passed with the pitch remaining within (half-tones) of a center pitch; the center pitch is then reported. If the instantaneous pitch differs by more than (half-tones) from the reported pitch, the search begins for a new note.

npartial (n) The jth partial is weighted as $n/(n+j)$ in the likelihood formula.

The following messages are also defined:

uzi (onoff) Turn "uzi" mode on or off; by default it's on. Turn it off if you want to poll for pitch tracks or sinusoidal components yourself; otherwise they come out on every analysis period.

bang ... poll them.

debug turn on debugging.

The computation load of `fiddle` varies depending on the input signal. In an informal test, running `fiddle` on a sawtooth plus white noise used 21 percent of the available CPU time on a 300 MHz. Pentium 2 machine running NT.

3 Bounded-Q Analysis

The `bonk` object was written for dealing with sound sources for which sinusoidal decomposition breaks down; the first application has been to drums and percussion. The design emphasizes speed; the hardwired analysis window size is 256 samples or 5.8 msec at 44K1; the hop size can be as low as 64 samples.

The first stage of analysis in `bonk` is a downsampling FIR filterbank of the sort described in [Brown 92]. Letting N be the window size, $M = N/2$, and $x[n], n = 0, ..., N-1$ the input signal, ω a center frequency and δ a filter bandwidth, we can compute the estimated signal power at ω with bandwidth δ as:

$$P(\omega, \delta) \equiv \left| \sum_{m=-T}^{T-1} \exp i\omega m \frac{1+\cos(\delta m)}{2} x[M+m] \right|^2$$

where

$$T = \lceil \pi/\delta \rceil.$$

The minimum bandwidth we can achieve is thus $2\pi/N$. The particular choice of frequency/bandwidth combinations in `bonk` was two filters per octave except where prohibited by the bandwidth limit:

$$(\omega, \delta) = (2\pi/N, 2\pi/N), (4\pi/N, 2\pi/N), (6\pi/N, 2\pi/N),$$
$$(6\sqrt{2}\pi/N, 2\sqrt{2}\pi/N), (12\pi/N, 4\pi/N),$$
$$(12\sqrt{2}\pi/N, 4\sqrt{2}\pi/N), (24\pi/N, 8\pi/N)$$

and so on up to the Nyquist, giving a total of 11 filters for $N = 256$.

3.1 Detecting attacks

The most satisfying application of this analysis is in detecting percussive attacks. The most popular way of doing this is to use an envelope follower and look for rapid rises in follower output; but any kind of ringing can set off trains of unwanted attacks, or oppositely, can mask true attacks. The analysis used by `bonk` can often detect new attacks which appear as sharp relative changes in the spectrum without any accompanying large change in the overall power; conversely, ringing instruments don't often give rapidly changing spectra and hence don't attract `bonk`'s attention.

We define a growth function as follows. For each channel we maintain a mask m which represents the current power in the channel. To accomplish this, after each analysis we look at the current power p. If $p > m$ we replace m by p; otherwise if m hasn't been updated for more than *masktime* analyses (5 by default), the mask decays by multiplication by *maskdecay* (0.8). Since the default analysis interval is 3 msec, the default mask time is about 15 msec; much shorter than this and your kettle drum will set off an attack every half period.

The growth in each channel is the dimensionless quantity,

$$g = \max(0, p/m - 1)$$

which is 1 if the current power is twice the mask, for instance. Next we add up the growth estimates for all eleven channels. If this total exceeds *hithresh* (default 12), we'll report an attack. However, we don't actually report the attack until the spectrum *stops* growing, *i.e.,* the growth must decrease to a value below *lothresh* (default 6). This is done so that the true loudness and spectrum of the new event can be reported.

3.2 Matching spectral templates

It is also possible to ask `bonk` to test any new attack against a menu of pre-recorded attacks in order to guess which of several possible instruments was responsible for the new attack. To do this, first we store spectral templates for each of the instruments. Thereafter, any new attack is compared with the stored ones and the closest match is reported. The underlying assumption, that there is actually some repeatability in the spectral envelopes of attacks of percussive instruments certainly doesn't hold true in the real world, but it is interesting to learn which sorts of instruments `bonk` can identify in this way and which it can't.

To describe how template matching works we first add indices to the variables for power and mask; p_i, m_i are the power and mask for the ith channel for $i = 1, ..., 11$. Now suppose s_i, t_i, are the spectra of two pre-recorded attacks normalized so that $|S| = |T| = 1$ as real 11-dimensional vectors. The simplest test would be to ask which of $P \cdot S$, $P \cdot T$ is greater. However, some information might be missing in P because of masking so a more appropriate measure of agreement between P and S, for example, is to weight each component by its "clarity" which we measure by the growth g_i; so the value of the fit between P and S is thus

$$\frac{(\sum g_i s_i p_i)^2}{\sum g_i {s_i}^2 \sum g_i {p_i}^2}.$$

the template which agrees best with P in this sense is the output reported.

3.3 The bonk object design

Like `fiddle`, `bonk` has one inlet which takes an audio signal and messages to alter its settings and to learn, store and recall templates:

thresh (lothresh) (hithresh) Set the attack thresholds

mask (masktime) (maskdecay) Set the mask parameters

debounce (debounce-time) Set the minimum time in msec between two attacks

print (more) Print the values of the parameters and (if more) the current analysis vector

learn (flag) turn "learn" mode on or off

write (filename) write learned templates to a text file

read (filename) read templates from a text file

bang output current spectrum as a list

Two outputs are provided; the rightmost reports the entire eleven-element spectrum on attacks and on `bang`; the other gives the number of the best matching template for each attack and the overall loudness of the attack. If you just want to be able to poll the spectrum, set the threshold to an impossible value so `bonk` won't volunteer output on its own.

4 Acknowledgement

This work was generously supported by the Intel Corporation. The techniques described here have evolved over many years of collaboration with researchers and artists, especially Mark Danks, A. Couturier Lippe, Philippe Maunoury, Joel Settel, Vibeke Sorensen, and Rand Steiger. The `bonk` and `fiddle` objects described here were developed as part of Puckette, Sorensen and Steiger's *Lemma 1*, shown at ICMC97, Thessaloniki, Greece, played by George Lewis and Steven Schick, and at UCSD by Vanessa Tomlinson and Michael Dessen.

References

[Brown 92] Brown, J.C., and Puckette, M.S., (1992). "An Efficient Algorithm for the Calculation of a Constant Q Transform", J. Acoust. Soc. Am. 92, 2698-2701.

[Noll 69] Noll, A. M., 1969. "Pitch determination of human speech by the harmonic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate." Proc. Symp. Computer Proc. in Comm., pp. 779-798.

[Puckette 96] Puckette, M., 1996. "Pure Data: another integrated computer music environment." Proc. the Second Intercollege Computer Music Concerts, Tachikawa, pp. 37-41. Reprinted as `ftp://crca-ftp.ucsd.edu:~/pub/msp/pd-kcm.ps`

[Puckette 98] Puckette, M., and Brown, J., 1998. "Accuracy of frequency estimates from the phase vocoder." IEEE Transactions on Speech and Audio Processing 6/2, pp. 166-176.

[Rabiner 78] Rabiner, L.R., and Schafer, R.W., 1978. *Digital Processing of Speech Signals.* Englewood Cliffs, N.J.: Prentice-Hall.