Real-Time Time-Domain Pitch Tracking Using Wavelets

Eric Larson
Departments of Mathematics, Physics, and Philosophy, Kalamazoo College
larson.eric.d@gmail.com

Ross Maddox
Center for Performing Arts Technology
University of Michigan School of Music
rkmaddox@umich.edu

ABSTRACT

A pitch tracker based on the fast lifting wavelet transform (FLWT) is developed in MATLAB and C++. Emphasis is placed on low latency (~25 ms), high time resolution, and accuracy. The pitch tracking algorithm implements the FLWT using the Haar wavelet, a transform which is shown to be mathematically equivalent to splitting a signal into low-pass and high-pass components and downsampling (generating *approximations* and *details*, respectively). Approximations are used in combination with intelligent peak detection to determine the pitch of vocal samples. An averaging scheme is employed to provide enhanced frequency resolution. This algorithm was tested on natural and synthetic signals—including a female vocal sample and two sets of generated sinusoidal signals—to determine its performance characteristics. Overall performance exceeded expectations, with most errors made in voiced/unvoiced detection.

I. Background

Natural sounds are a composition of a fundamental frequency with a set of harmonics which occur at near integer multiples of that fundamental. The frequency that the human ear interprets as the *pitch* of a sound is this fundamental frequency, even if it is absent in the sound. The pitch (or fundamental frequency)[1] of natural sounds is important in many contexts. Pitch is largely responsible for inflections in human speech that carry conversational cues (e.g. discriminating between the spoken phrases "you are going to the store." and "you are going to the store?"). These inflections also play a role in allowing us to consistently identify a given speaker. In addition to speech, musical compositions are typically sets of organized pitches. These pitches, in combination with their timing, enable us to distinguish Beethoven's third symphony from Led Zeppelin's "Stairway to Heaven".

Pitch tracking, then, is useful for musical analysis as well as for speech analysis. General pitch tracking is used in speech recognition and identification schemes as well as in the automated transcription of music. Real-time pitch tracking can be used as an aide for (or judge of) musical performers, revealing intricacies of performance not immediately revealed by the ear.

Tracking the pitch of natural sounds—here the focus is primarily on speech—is not a trivial task. The topic of pitch tracking has been well explored, and there are several established pitch tracking methods available today, alongside several unconventional ones. All pitch trackers have certain advantages and disadvantages.

[1] Although there are classical distinctions between *pitch* and *fundamental frequency*, the two will be considered equivalent for the purposes of this paper.

Any real-time pitch tracker relies on processing consecutive small portions of a signal to produce pitch values. This process is called *windowing*. With a sample rate of 44100 Hz, it is not uncommon to use a window length of 1024 or 2048 samples—which correspond to minimum pitch-determination latencies of about 25 and 50 ms, respectively. Processing time following the windowing adds to the latency. Although shorter windows can lead to lower latency and higher time resolution, they can compromise resolution in the frequency domain. In the time domain, shorter windows limit the lowest detectable frequency.

Pitch trackers fall into two general categories: time-domain and frequency-domain. The former analysis examines the original signal, often applying filters and/or convolution to analyze the signal in its original state, amplitude vs. time. The latter uses a transform (usually the Fast Fourier Transform, FFT) to break the signal down into its frequency components, yielding information about its amplitude vs. frequency. It then analyzes this to determine the fundamental frequency. Both of these have advantages and disadvantages when it comes to frequency resolution and processing time.

A good summary of established real-time pitch detection methods is provided in [2]. However, two of the most popular methods will be described briefly here.

Autocorrelation is a primary time-domain method. Autocorrelation selects a portion of the windowed signal and computes the correlation (sum of the products divided by the length of the portion, a measure of similarity between two signals) between that portion and an equal-length sliding segment of the windowed signal, at every point in that window. It then uses the locations of spikes in the correlation to determine the period and, hence, the pitch of the signal.

The primary frequency-domain analysis method is the *cepstrum* (whose name is a rearrangement of the word *spectrum*), which involves taking the magnitude spectrum of the log of the magnitude spectrum of the windowed signal. The peaks of this double-FFT can be examined to determine the frequency of the original signal by relying on the "periodicity" of its harmonics, which in natural sounds are integer multiples of the fundamental.

All real-time pitch trackers can be evaluated in terms of four main performance characteristics. First is computation time, which should be minimized in order to minimize latency. The cepstrum performs two FFTs, and autocorrelation requires many convolutions, which each consist of many multiplication operations—which means both methods require large computation time, and hence larger latencies. Second is determination of voiced segments from unvoiced (or sounded from unsounded for instrumental samples). A pitch tracker should be able to distinguish unpitched consonants from pitched vowels, as well as ignore areas of silence. Third is pitch accuracy, how closely the pitch tracker's estimate matches the true pitch. A pitch tracker should have good resolution as well as avoid making gross errors, such as doublings, halvings, and large spikes, which can occur due to noise, transients, strong upper harmonics, and changing overall amplitude within a window.

The currently established methods of pitch tracking all have drawbacks in some of these categories. In this paper, we present a pitch tracker with enhanced performance in all of these respects in an attempt to offer a viable, fast real-time alternative to other pitch trackers.

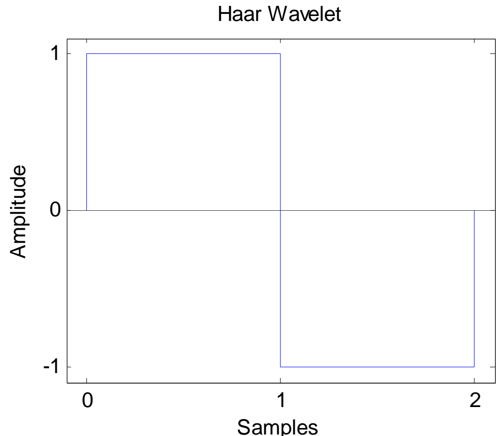

FIG. 1. The Haar wavelet, which serves as the basis of the FLWT (fast lifting wavelet transform) used in this algorithm.

II. Method

When visually examining a periodic signal, it is almost always easy to see the periodicity. In designing this pitch tracker, we set out to model the process of visually determining the period of a windowed signal using simple logic and computationally inexpensive transforms to find the extrema which correctly reveal the periodicity of waveforms.

A. Fast Lifting Wavelet Transform

The algorithm uses an implementation of the Fast Lifting Wavelet Transform (FLWT). A wavelet transform splits a signal into an approximation and a detail. Successive transforms can be applied to the approximations to reduce noise and reveal underlying periodicity which is normally more difficult to extract from the original signal. Any of several different mother wavelets can be used to perform the transform. This pitch-tracking algorithm implements a FLWT using the Haar wavelet. For details on wavelets and the FLWT in a general setting, refer to [1]. Figure 1 shows the Haar wavelet, the mother wavelet used to generate the FLWT of this algorithm.

The FLWT using a Haar wavelet is mathematically equivalent to running a low-pass filter and downsampling to produce the approximation component and running a high-pass filter and downsampling to produce the detail component. The equations derived previously by Debauchies and Sweldens for the FLWT with the Haar wavelet are

$$\begin{aligned}
\mathbf{d_0}(n) &= \mathbf{x}(2n+1) \\
\mathbf{a_0}(n) &= \mathbf{x}(2n) \\
\mathbf{d_1}(n) &= \mathbf{d_0}(n) - \mathbf{a_0}(n) \\
\mathbf{a_1}(n) &= \mathbf{a_0}(n) + \mathbf{d_1}(n),
\end{aligned} \tag{1}$$

where $\mathbf{x}(n)$ is the original signal, $\mathbf{a_1}(n)$ the first approximation, and $\mathbf{d_1}(n)$ the first detail. With some algebra, it follows that these are equivalent to

$$\begin{aligned}
\mathbf{d_1}(n) &= \mathbf{x}(2n+1) - \mathbf{x}(2n) \\
\mathbf{a_1}(n) &= \frac{\mathbf{x}(2n+1) + \mathbf{x}(2n)}{2}.
\end{aligned} \tag{2}$$

From these equations it is clear that the approximation component is simply an application of an averaging filter (a first-order low-pass) with downsampling (taking every other result); and the detail component is an application of a first-difference filter (a high-pass) with downsampling (taking every other result). In this way, the Haar wavelet FLWT provides a quick method of splitting a signal into high-pass and low-pass components.

This splitting scheme can help reveal underlying periodicity. By discarding the detail component and performing another FLWT on the approximation, additional levels of the wavelet transform are generated. The repeated (but limited) application of the FLWT in this manner ideally yields a pitch-tracker that is robust to noise and able to distinguish pitched sounds from unvoiced consonants. Using

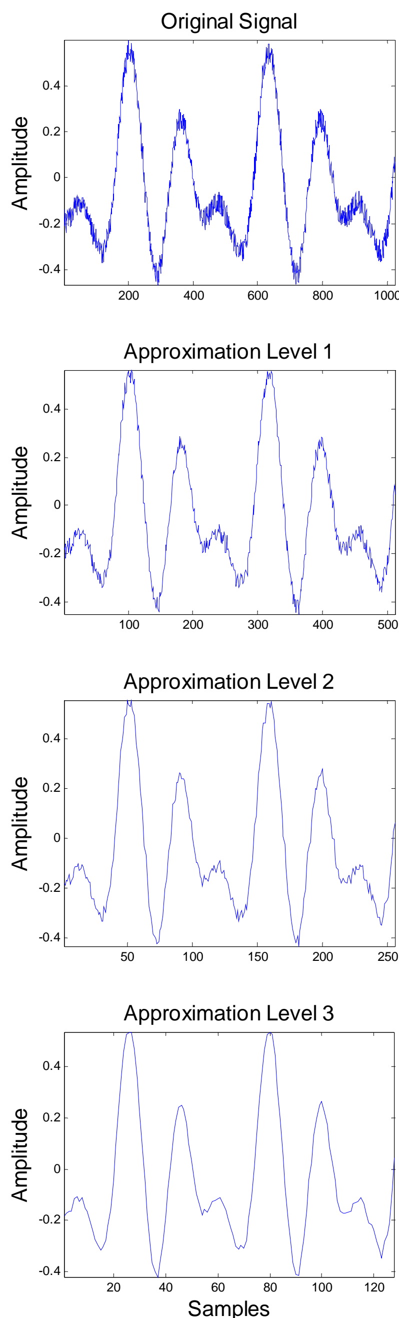

FIG. 2. Amplitude vs. Time for an original signal (top) and the first/second/third wavelet approximations (second, third, and fourth from the top). Note the reduction of noise and simplification of the signal with each successive approximation.

this low-pass technique allows the computer to "see" the underlying periodic waveform that is already apparent to the human observer. Since the desired frequency range of operation is from about 80 Hz to 3000 Hz, the use of several repeated low-pass operations and downsamplings does not compromise the frequency components that allow pitch determination. Since at least two wavelet approximations must be performed, there is an inherent ideal frequency ceiling of Nyquist/4, which for CD sample rate is about 5500 Hz. See Figure 2 for an image of the wavelet transformation, containing an original signal and the first, second, and third wavelet approximations. Note that with each successive wavelet transform, the number of samples in the approximation is halved as the signal is split into approximation and detail; this provides a practical operational limit on the number of wavelet transform levels that can be run on any given window of a signal.

B. Algorithm

The steps of the pitch tracking algorithm for each current window of the signal are as follows:

1) Perform the FLWT, discard the detail component.
2) Find the first local maximum/minimum after each zero-crossing of the current approximation.
3) Determine the mode distance between the maxima/minima.
4) Check to see if the mode distance is equivalent to that of the previous level. If so take that to be the period and move to the next window; if not, go to the next level (not to exceed a specified wavelet transform level limit) and return to step 1; if the level limit is exceeded, assume that the signal is unpitched and move to the next window.

One of the first uses of the wavelet transform in pitch detection came from Kadambe in [4], and proved successful. In

the same vein, Erçelebi in [3] used the FLWT to speed up the implementation. Our work is similar to his in many respects. We build on his work, employing several improvements: 1) a more robust peak-detection algorithm (accompanied by a more explicit description of method); 2) a different wavelet level decision scheme; 3) a transient detector that marks the windowed signal as unpitched if the RMS of the first third is more than four times that of the last third (or vice versa); as well as 4) a mode-averaging method that increases frequency resolution, especially in the high frequencies.

C. Peak Detection/Differences

The maximum-finding process[2] involves several steps. The average amplitude is calculated and subtracted from every element of the windowed signal, removing the DC component.[3] A lower threshold for maxima is set which is a percentage of the window global maximum. Next, the location of the first local maximum (following the first derivative test—a change from positive to negative slope) between every set of zero crossings with an amplitude greater than the threshold is recorded as a location of a maximum. This procedure sets an upper limit on the number of maxima equal to one fewer than the number of zero crossings, providing an upper limit on the frequency. Also, no maximum index m_i can be recorded if it is within some minimum distance δ of the previous maximum index m_{i-1}. The minimum distance δ is a controllable parameter related to the specified maximum returnable frequency F and the current wavelet level i in the following way (where F_s is the sample rate):

$$\delta = \max\left(\left\lfloor \frac{F_s}{2^i F} \right\rfloor, 1 \right). \tag{3}$$

Next, the distances between these peaks are calculated. For each maximum index m_i, the distances between m_i and subsequent maximum indices m_{i+1}, m_{i+2}, m_{i+3}, …, m_{i+N} are calculated and stored (the number N of subsequent peaks taken into consideration is a controllable parameter). Taking several distance levels helps to ensure that a waveform which yields more than one maximum value per period is still analyzed correctly. While this may appear prima facie to also induce halvings or thirdings of the frequency, the finitude of the window size ensures that there are more of the correct distance than of its integer multiples resulting from taking several levels of differences.

These differences calculated using the maxima are combined with the differences calculated using the minima to determine a mode distance. For each distance in set, the number of distances close to (i.e. within a specified tolerance δ of) it are counted; the distance with the most other distances close to it in value is taken to be the center mode. In the case of a tie between two such modes, the larger mode is taken if it is twice as long as the smaller mode (biasing toward frequency halvings). The mode from the previous window pitch detection is also passed to the function, assisting in the case of a near-tie. If the number of occurrences of a mode candidate is within one of that of the actual mode, it biases mode selection toward the previous window's period.

D. Mode Averaging

---

[2] The minimum finding process is equivalent to the maximum finding process, with appropriate sign and inequality adjustments

[3] In practice, the value is not truly subtracted from every element, but the method implemented generates equivalent results with less computation.

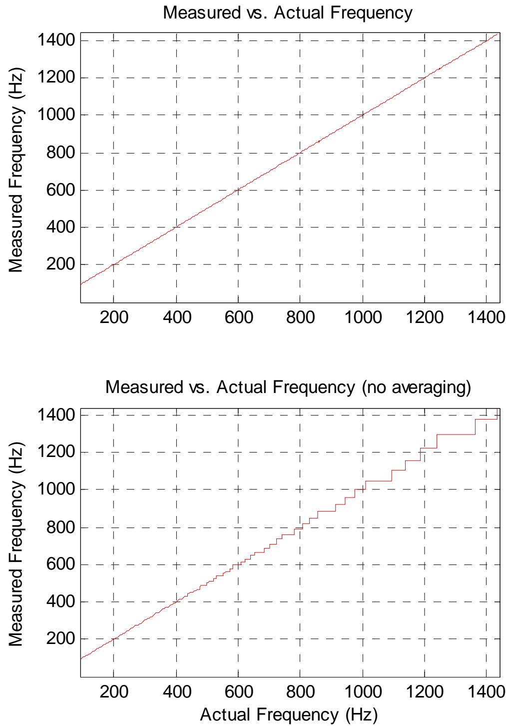

FIG. 3. The response of the pitch tracker to a 1-Hz-stepped sinusoidal signal from 100 to 1500 Hz. Averaging method (upper plot) improves accuracy over the non-averaging method (bottom).

Once this center mode is selected, an averaging scheme is employed to increase frequency resolution. The mean of the distances within δ of the center mode is taken to be the period of the signal (with appropriate scaling by a power of two to compensate for the downsampling of the FLWT). In the low frequencies, this averaging occurs over only a limited number of periods and has a small effect on the increased resolution; in the high frequencies, where a simple integer mode would yield a high pitch detection error (due to discretization and sample rate limitations), the averaging function enhances the performance by providing a more accurate divisor when calculating frequency from period. See Figure 3 for an illustration of this increase in frequency resolution in a four-octave test from 90 to 1440 Hz.

This algorithm has been implemented in MATLAB and C++; see appendix A for the MATLAB implementation.

III. Results and Discussion

The controllable parameters of the algorithm are the maximum frequency F from which δ is derived; the number of subsequent peaks N to consider when calculating distances; the global threshold percentage M of the window maximum that a local max must exceed to be counted; and L, the maximum number of wavelet transforms to perform before a window is deemed pitchless. There was much experimentation with all of these numbers, and pitch detection performance was highest using the values listed in Table I.

The criteria used to evaluate the pitch tracker are computation time (latency), voiced/unvoiced determination, and overall pitch detection accuracy. Each of these performance characteristics is discussed below. Performance was tested using recorded or synthesized samples, rather than live input to allow for greater control and repeatability of tests. In general, the pitch tracker was calibrated to function over a frequency range of about 100-1500 Hz, so most tests were run on samples from 90-1440 Hz to allow test over a large range and easy octave-based bracketing.

A. Computation Time

The latency is comprised of two parts: the buffering period, in which the win-

TABLE I. Pitch Tracker Parameters

Maximum Frequency F	Difference Levels N	Maxima Threshold M	FLWT Levels L
3000 Hz	3	0.75	6

Table I. List of algorithm parameters experimentally determined to yield best performance.

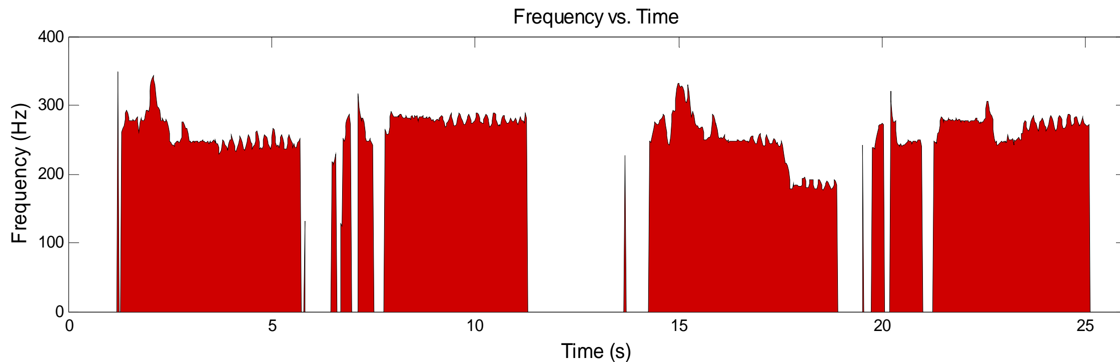

FIG. 4. Frequency vs. Time for a 26-second sample of female vocals. One voiced-to-unvoiced and three unvoiced-to-voiced errors occurred (~0.4 % error rate). Good time and frequency resolution of the pitch tracker reveal vibrato, toward the end of sustained notes, and vocal trills.

dowed signal is acquired, and the pitch computation on that window. For the tests done on this pitch tracker, the sample rate was 44100 Hz and the window size was 1024 samples, yielding a buffering time of 23 ms. Using a 3 GHz Pentium 4 Processor, the computation time for each window in MATLAB was 4 ms, yielding a total latency of 27 ms. The C++ implementation has an even shorter computation time, with overall latency even closer to the 23 ms buffering time. This implies that if the minimum frequency present in the signal were closer to 200 Hz, the window size could be decreased to 512 samples, reducing both window computation time and buffering time, nearly cutting the latency in half.

B. Voiced/Unvoiced Detection

The voiced/unvoiced detection in the pitch tracker was its weakest performance category. There were specific windows which the pitch tracker failed to recognize as unvoiced, yielding a spike in the frequency where there should have been a zero to denote unvoiced. See Figure 4 for an illustration of this on a 26-second female vocal recording. In over 1000 sample windows, three unvoiced-to-voiced errors and one voiced-to-unvoiced error occurred, yielding an error rate of less than 0.4%. Although this is not an exhaustive analysis, it is representative of the algorithm's general behavior; voiced/unvoiced error rates among other test samples were less than 1%.

C. Pitch Detection Accuracy

The first pitch detection accuracy measure was an error-per-octave test on sinusoidal waveforms of constant frequency within a 1024-sample window. The accuracy of the pitch tracker was very good throughout the tested frequency range of 90 – 1440 Hz, in increments of 1 Hz each interval. See Figure 5 values at each octave of the RMS error in Hz, RMS error in cents (defined as hundredths of a musical semitone), and mean error in Hz. The maximum RMS error was about 0.6 Hz in the 720-1440 Hz range. Mean errors were negligibly centered about zero. This test indicates extremely good performance on sinusoidal waveforms, with the RMS error increasing approximately exponentially with frequency, doubling with every octave band. This leads to a nearly uniformly small (1 ± 0.5) RMS error in cents across the octave bands tested. This indicates that the pitch tracker has near-uniform accuracy in terms of musical in-

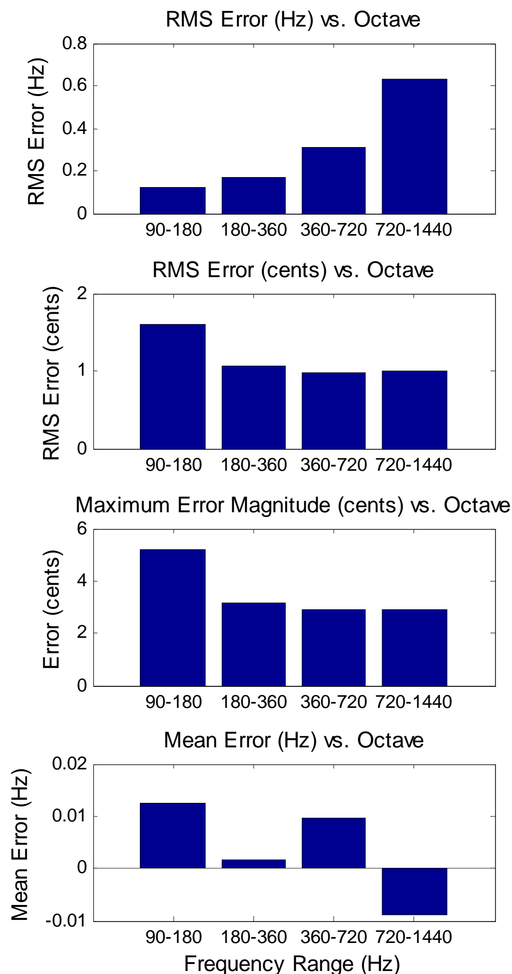

FIG. 5. RMS Error (Hz), RMS Error (cents), Maximum Error Magnitude (cents), and Mean Error (Hz) vs. Frequency Range (Hz) over four octaves. Mean error across all octaves is about 0 ± 0.01 Hz. RMS error increases exponentially, doubling with every octave, achieving a maximum value (worst performance) of about 0.6 Hz in the 720-1440 octave.

tervals across its frequency range. Additionally, this accuracy is below the just noticeable difference (JND) of pitches. According to [5], although the JND varies from person to person and across frequencies, a good estimate is that the JND for pitch is about 5 cents. Since the maximum error across all frequency bands was about 5 cents, the algorithm provides accuracy

TABLE II. Missing Fundamental First Failures

F_0	90	127	180	255	360	509	720
F_n	720	1018	1620	1527	1440	1527	1440
n	8	8	9	6	4	3	2

Table II: Missing Fundamental test first failures. For each fundamental frequency F_0 (in Hz), the lowest frequency of the first adjacent upper harmonic pair to fail, F_n (in Hz), is listed, as well as its harmonic number n. Note that the missing fundamental detection fails (except for the 90 Hz case) when at least one of F_n and F_{n+1} is above the frequency range of the pitch tracker.

within the JND an overwhelming proportion of the time.

A second, qualitative analysis of accuracy was performed on a 26-second female vocal sample. See Figure 4 for the pitch tracker's analysis of this sample. This analysis reveals the details of the singer's vibrato at the end of words, and trills. No doublings or halvings occurred during the analysis. Although rare, other tested samples did have halvings, at a rate less than one half of one percent. No doublings were found in any of the analyses of test samples.

The third test was a missing-fundamental test. Signals were synthesized using upper harmonics (with at least one odd harmonic), omitting the fundamental, for fundamental frequencies spanning 90 – 720 Hz by half-octaves. For each frequency, nine 100-window samples were constructed by adding the adjacent harmonic pairs (2nd and 3rd, then 3rd and 4th, and so on up to the 10th and 11th). Each of these nine samples was analyzed, and Table II shows the threshold of errors for each missing fundamental F_0.

With the exception of the 90 Hz case, the pitch tracker accurately estimated the pitch of the missing fundamental up to the point at which the higher of the two summed harmonics was above the maxi-

mum detectable frequency. So long as both harmonics were below around 2500 Hz, the pitch tracker detected the missing fundamental. In the 90 Hz case, the pitch tracker began failing when the upper harmonics were 720 and 810 Hz; the reason for this failure was not determined. However, a signal containing only the 8th and 9th harmonics is unlikely to be encountered in practical use. Overall, the missing fundamental performance was very good, limited mainly by the highest detectable frequency of the pitch tracker.

D. Discussion

This pitch detection method has very good resolution in the low frequencies (due to its use of time-domain methods) and in the high-frequencies (due to the averaging method). This resolution enables accurate pitch detection as outlined above. The use of a 1024-sample analysis window with sounds sampled at 44100 Hz yields very good time resolution as well.

When dealing with vocal samples, this algorithm is relatively immune to pitch doublings and halvings, which are a common problem for pitch trackers. For other types of sound, such as distorted electric guitar, limited testing revealed doublings due to strong upper harmonics. Halvings were rare in all samples tested, and did not appear systematically.

An absolute threshold could be introduced to reduce the voiced/unvoiced errors seen with this algorithm. A threshold reduces spikes in unvoiced sections, but also trims very low-level voiced sections, which was considered a poor tradeoff. If a high enough input signal level could be guaranteed, an absolute threshold could be a valuable and computationally cheap improvement for real-time applications. For non real-time applications, a threshold based on the global maximum/minimum value proves useful, and could be easily added to the existing code.

In non real-time applications, a smoothing function could also be implemented. Most pitch trackers that deal with signals in non real-time implement some form of a smoothing algorithm to deal with voiced/unvoiced errors and pitch doublings and halvings. This algorithm would benefit similarly from such a method, but would lose its real-time functionality.

The pitch tracker compared favorably to the established methods of cepstrum and autocorrelation in all three categories. Computationally this algorithm was cheaper, voiced/unvoiced errors were similarly frequent across methods, and pitch accuracy was as good or better.

IV. Conclusions

In this paper a real-time pitch tracking algorithm was presented. Low measurement error (high pitch detection accuracy) expectations were exceeded by the algorithm due to its good frequency resolution in both low and high frequencies. Short analysis window length led to good time resolution. Error rates (voiced/unvoiced and frequency halvings) were low, but further improvements could be made. Computational costs were even lower than expected, yielding very low latency periods. Compared to established methods, this algorithm compares favorably, and could be useful in real-time applications where latency and pitch detection accuracy are emphasized.

V. Acknowledgments

We would like to thank Prof. S. Errede for the opportunity work for him this summer; J. Boparai for his help in the lab; M. Winkler for working with us and helping us in the lab; J. Beauchamp and M. Bay for advice on pitch tracking; The Physics department at University of Illi-

nois at Urbana Champaign for hosting the REU program; and the National Science Foundation for support via grant PHY-0243675.

VI. References

[1] Daubechies, Ingrid and Wim Sweldens. “Factoring Wavelet Transforms into Lifting Steps.” J. Fourier Anal. Appl, 1998.

[2] de la Cuadra, Patricio et al. “Efficient Pitch Detection Techniques for Interactive Music.” Proceedings of the 2001 International Computer Music …, 2001.

[3] Erçelebi, Ergun. “Second generation wavelet transform-based pitch period estimation and voiced/unvoiced decision for speech signals.” Applied Acoustics 64 (2003) 25–41.

[4] Kadambe S, Faye Boudreaux-Bartels G. Application of the wavelets transform for pitch detection of speech signals. IEEE Trans on Information Theory 1992;38(2):917–24.

[5] Rossing, Thomas D., The Science of Sound 2nd Ed, Addison-Wesley 1990.

Appendix A: wavePitch.m

```
function freq = wavePitch(data,fs,oldFreq)
%
%   WAVEPITCH Determine the pitch of a given short portion of data.
%
%   WAVEPITCH(data, fs, oldFreq) data is a [ 1 x samples ] array. Optional
%   inputs are fs (the sample rate of the signal, defaulting to 44100 if
%   unspecified), and oldFreq (the frequency from the previous window).
%
%   N.B. Data input should be at least 256 samples long. 1024 is recommended.
%        It also must be a multiple of 64.
%
%   Copyright 2005 Ross Maddox (University of Michigan)
%              and Eric Larson (Kalamazoo College)

if (nargin < 1) return; if (nargin < 2) fs = 44100; if (nargin < 3) oldFreq = 0; end

oldMode = 0;
if(oldFreq)
  oldMode = fs/oldFreq;
end

dataLen = length(data);
freq = 0;              % The freq to return
lev = 6;               % Six levels of analysis
globalMaxThresh = .75; % Thresholding of maximum values to consider
maxFreq = 3000;        % Yields minimum distance to consider valid
diffLevs = 3;          % Number of differences to go through (3 is diff @ third neighbor)

maxCount(1) = 0;
minCount(1) = 0;

a(1,:) = data;
aver = mean(a(1,:));
globalMax = max(a(1,:));
globalMin = min(a(1,:));
maxThresh = globalMaxThresh*(globalMax-aver) + aver; % Adjust for DC Offset
minThresh = globalMaxThresh*(globalMin-aver) + aver; % Adjust for DC Offset

%% Begin pitch detection %%

for (i = 2:lev)
  newWidth = dataLen/2^(i - 1);

  %% Perform the FLWT %%

  j = 1:newWidth;
  d(i,j) = a(i-1,2*j) - a(i-1,2*j-1);
  a(i,j) = a(i-1,2*j-1) + d(i,j)/2;

  %% Find the maxes of the current approximation %%

  minDist = max(floor(fs/maxFreq/2^(i-1)),1);
  maxCount(i) = 0;
  minCount(i) = 0;

  climber = 0; % 1 if pos, -1 if neg
  if (a(i,2) - a(i,1) > 0)
    climber = 1;
  else
    climber = -1;
  end

  canExt = 1;    % Tracks whether an extreme can be found (based on zero crossings)
  tooClose = 0; % Tracks how many more samples must be moved before another extreme

  for (j = 2:newWidth-1)
    test = a(i,j) - a(i,j - 1);

    if (climber >= 0 && test < 0)
      if(a(i,j - 1) >= maxThresh && canExt && ~tooClose)
        maxCount(i) = maxCount(i) + 1;
        maxIndices(i,maxCount(i)) = j - 1;
        canExt = 0;
        tooClose = minDist;
      end
      climber = -1;
```

```
    elseif (climber <= 0 && test > 0)
      if(a(i,j - 1) <= minThresh && canExt && ~tooClose)
        minCount(i) = minCount(i) + 1;
        minIndices(i,minCount(i)) = j - 1;
        canExt = 0;
        tooClose = minDist;
      end
      climber = 1;
    end

    if (a(i,j) <= aver && a(i,j - 1) > aver) || (a(i,j) >= aver && a(i,j - 1) < aver)
      canExt = 1;
    end

    if(tooClose)
      tooClose = tooClose - 1;
    end
  end

  %% Calculate the mode distance between peaks at each level %%

  if (maxCount(i) >= 2 && minCount(i) >=2)

    % Calculate the differences at diffLevs distances

    differs = [];
    for (j = 1:diffLevs)       % Interval of differences (neighbor, next-neighbor...)
      k = 1:maxCount(i) - j;   % Starting point of each run
      differs = [differs abs(maxIndices(i,k+j) - maxIndices(i,k))];
      k = 1:minCount(i) - j;   % Starting point of each run
      differs = [differs abs(minIndices(i,k+j) - minIndices(i,k))];
    end

    dCount = length(differs);

    % Find the center mode of the differences

    numer = 1;   % Require at least two agreeing differs to yield a mode
    mode(i) = 0; % If none is found, leave as zero

    for (j = 1:dCount)

      % Find the # of times that distance j is within minDist samples of another distance
      numerJ = length(find( abs(differs(1:dCount) - differs(j)) <= minDist));

      % If there are more, set the new standard
      if (numerJ >= numer && numerJ > floor(newWidth/differs(j))/4)
        if (numerJ == numer)
          if oldMode && abs(differs(j) - oldMode/(2^(i-1)) ) < minDist
            mode(i) = differs(j);
          elseif ~oldMode && (differs(j) > 1.95*mode(i) && differs(j) < 2.05*mode(i))
            mode(i) = differs(j);
          end
        else
          numer = numerJ;
          mode(i) = differs(j);
        end
      elseif numerJ == numer-1 && oldMode && abs(differs(j)-oldMode/(2^(i-1))) < minDist
        mode(i) = differs(j);
      end
    end

    %% Set the mode via averaging %%

    if (mode(i))
      mode(i) = mean(differs(find( abs(mode(i) - differs(1:dCount)) <= minDist) ));
    end

    %% Determine if the modes are shared %%

    if(mode(i-1) && maxCount(i - 1) >= 2 && minCount(i - 1) >= 2)

      % If the modes are within a sample of one another, return the calculated frequency
      if (abs(mode(i-1) - 2*mode(i)) <= minDist )
        freq = fs/mode(i-1)/2^(i-2);
        return;
      end
    end
  end
end
```