Enhancements To A Pitch Detection Algorithm

Dr David Sharp

Technical Report 96/9

Dept of Computing
Imperial College of Science, Technology & Medicine
180 Queen's Gate
London SW7 2BZ

9th October 1996

Abstract

This paper describes some enhancements to a pitch detection algorithm presented in [Sharp 93]. The enhancements reduce flickering between adjacent semitones when the algorithm is used as part of a pitch to MIDI converter. The enhancements also permit less memory space to be used and minimise accidental octave jumping. The enhancements illustrate the use of voting techniques for pattern recognition applications.

1 Introduction

The pitch period of a repetitive waveform is the distance between corresponding parts of the waveform and is shown in Figure 1.

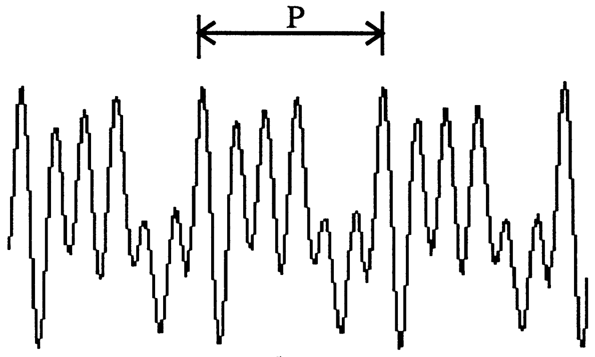

Figure 1 The Pitch Period, P, of a repetitive waveform

The pitch period of a waveform is of use in a variety of applications including the following;

- musical applications - conversion of singing and instrumental playing to music notation, real-time electronic music tutors, real-time toys that react to sound input, enhanced Karaoke machines and other electronic musical games,
- medical applications - heart beat monitoring, electrocardiogram analysis, breathing monitoring,
- engine monitoring applications - detecting faults that create a change in engine sound, and
- many other applications in which there are waveforms to be analysed or conclusions to be drawn from various pieces of evidence in which variations of the voting techniques presented here are applicable.

In [Sharp 93] an algorithm is presented which calculates the pitch period of a repetitive waveform using a technique that treats the waveform as if it were an image and votes for distances at which features of the waveform repeat. These distances are candidate pitch periods for the waveform. The voting scheme is arranged so that the candidate pitch period that receives the most votes is the pitch period of the waveform. For a sound waveform, the pitch period can be converted to a corresponding musical frequency or note. The musical frequency/note information can be as the basis for pitch to MIDI conversion, transcription of audio to music notation, feedback for a musical tutor, advanced Karaoke interactions or for other purposes.

2 Enhancing The System

The prototype system described in [Sharp 93] samples the input waveform at 11kHz so each sample is 91μs from the previous one. Each voting bucket b_i, i=0, 1, ... corresponds to a candidate pitch period i*91μs. The pitch period of the bucket that receives the most votes can be converted to a musical semitone using a simple formula based on A4=440Hz and the fact that you multiply frequency by 2^(1/12) to go up in pitch by a semitone.

The system splits the incoming audio stream into frames of a few milliseconds, determines candidate pitch periods and votes for them. The candidate pitch period that receives the most votes is winning pitch period for the frame. The votes are then cleared and the next frame is processed.

One problem with the system is that if the input pitch falls exactly between two semitones then a similar number of votes will be cast for buckets corresponding to the upper semitone and buckets corresponding to the lower semitone. This leads to some instability and there is a tendency for the winning pitch period of the system to alternate between the upper and lower semitones depending on which semitone got a few extra votes to give it a marginal win. This can produce a trilling effect on a MIDI output from the system in response to a steady input tone and this is undesirable. This paper shows how the voting can be organised to eliminate this problem.

To save memory it is possible to rearrange the voting buckets so that instead of having many buckets (one for each 91μS step), there is a voting bucket for each semitone (or quarter-tone) of the scale. This reduces the number of buckets to 12 (or 24 for quarter-tones).

A separate set of buckets are provided to accumulate votes for the octave in which the candidate pitch period lies and the buckets are arranged to avoid accidental octave-jumping as explained in the next section.

For pitch to MIDI conversion, the winning pitch bucket and octave are used to calculate which MIDI note to sound or silence at the end of each frame.

3 Enhanced Voting Strategy

The normal scale consists of 12 semitones denoted S_s = {C, C#, D, D#, E, F, F#, G, G#, A, A#, B}. We denote a quarter-tone scale by the set of 24 quarter-tones S_q = {C, C++, C#, C#++, D, D++, D#, D#++, E, E++, F, F++, F#, F#++, G, G++, G#, G#++, A, A++, A#, A#++, B, B++}. Here the symbol + may be thought of as increasing the pitch by one eight of a tone, hence ++ increases the pitch by one quarter of a tone. The note C++ may be thought of as lying in the crack between C and C# on a piano keyboard.

We also consider a scale consisting of 48 eighths of a tone as shown in Figure 1. The note one eighth of a tone above B is denoted B+ and note one eighth of a tone below B is denoted B-.

Similarly the note one quarter of a tone above C# is denoted C#++ and the note one quarter of a tone below C# is denoted C++.

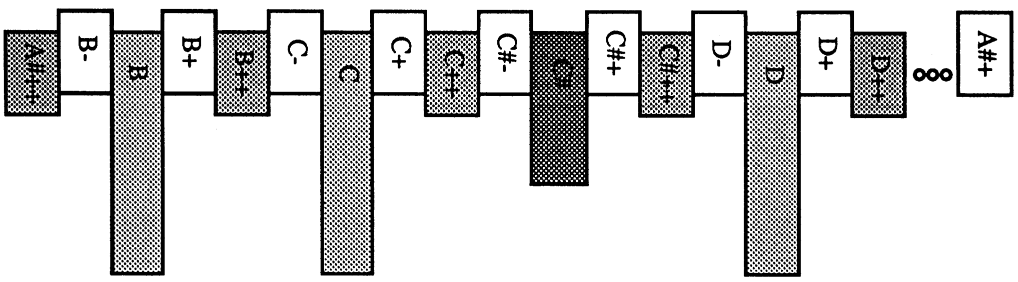

Figure 1 Eighth Tone Resolution

The pitch detection algorithm is arranged to detect when a vote is required for a candidate pitch in the set S_e = {C-, C+, C#-, C#+, D-, D+, D#-, D#+, E-, E+, F-, F+, F#-, F#+, G-, G+, G#-, G#+, A-, A+, A#-, A#+, B-, B+}. However, the voting buckets are the set of quarter-tones S_q. When a candidate pitch period in the set S_e is detected, a vote is cast for each of the two adjacent quarter-tones in the set S_q.

For example, if C- is detected as a candidate pitch period, votes are cast for the B++ below it and the C above it. Similarly if B+ is detected as a candidate pitch period, a vote is cast for the A#++ below it and the B above it.

Arranging the voting in this way smoothes the transition from one quarter-tone to the next and also enables the elimination of flickering between semitones as follows: if a steady pitch, for example F+, is input to the system then the F and F++ will receive many of the votes. It is possible that the winning pitch will alter between F and F++ on subsequent voting iterations. However, as F++ does not correspond to an ordinary musical semitone, the system can sound an F continuously and there is no flicker.

If the input pitch were to subsequently rise to F#- and the winning pitch were to change to be F#, the pitch to MIDI converter could then sound an F#. It would hold the sounding pitch at F# even if the winning pitch lowered to F++ (in response to the input pitch dropping to F#-) as F++ is not sufficiently below F# to switch to sounding an F. This strategy gives the system hysteresis of a quarter-tone and considerably stabilises its output.

4 Treatment of Octaves

It is normal to number octaves from C such that middle C is in octave 4, the C above it is in octave 5 and the C below middle C is in octave 3. Middle C is thus written C4 and the B above it is B4 and the B below it is B3.

Here we count in half octaves from C and also half octaves from F#-. Middle C is in half-octave 8 counting from C and half-octave 9 counting from F#-. Each quarter-tone is considered to be a member of an octave starting from C and also a member of an octave starting from F#- as shown in Figure 2.

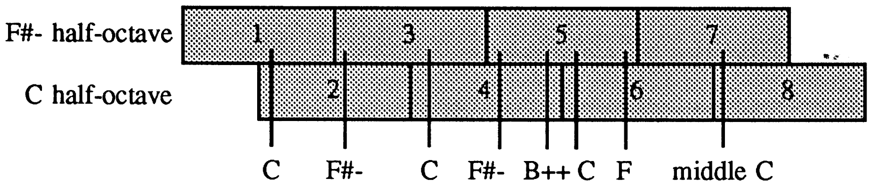

Figure 2 Arrangement of Half Octaves

When a vote is cast for a quarter-tone, a vote is also cast for each of the half-octaves in which the quarter-tone lies. For example a vote for the quarter-tone C4 (i.e. middle C, 261 Hz) is accompanied by a vote for half-octave 8 and also a vote for half-octave 7.

The winning half-octave is the half-octave that receives the most votes. It could be an F#- half-octave or a C half-octave that wins. The winning quarter-tone and winning half-octave always define a unique winning pitch with no ambiguity and no accidental octave jumping.

Overlapping the half-octaves in the above manner dramatically reduces the chances of accidentally jumping up or down an octave when the input pitch lies on the dividing line between two octaves.

For example if the input pitch is wavering between middle C- and C+ (i.e. the pitch is intermediate between C- and C+) then some votes will be cast for half-octave 8 and some for half-octave 6. If only octaves from C were used it is possible that the correct quarter tone would be consistently deduced (i.e. C) but the octave may jump between 6 and 8 accidentally and repeatedly. By also voting for octaves from F#- this problem is eliminated: each vote cast by C- or C+ for B++, C or C++ will be accompanied by a vote for F#- based half-octave 7. The votes for C based half-octaves will be split between half-octaves 6 and 8. Half-octave 7 will thus be the winning octave and middle C will be correctly determined as the winning pitch.

5 Action Taken After a Winning Quarter Tone and Octave are Determined

For a real-time pitch to MIDI converter the detection of a winning pitch results in a decision to send out appropriate MIDI information to make a synthesizer receiving the MIDI information play appropriately-pitched notes in response to the pitch input.

To decide what action to take on each iteration we utilise the following state variables:

IncomingSound	(boolean) - the user is singing/playing loud enough for a note to sound
StrongWinningPitch	(boolean) - true if the number of votes for the winning pitch are sufficient for the system to be confident it is correct. (This is a function of the octave of the winning pitch.)
MIDISounding	(boolean) - true if a note is currently being sounded. i.e. the last message sent was a Note On message.

The algorithm's decision procedure is illustrated in Figure 3 and it is suitable for a musical output that corrects the input pitch to the nearest semitone (based on A4=440Hz). This is achieved by making the "if pitch has changed" decision such that only semitones are sounded and quarter tones such as F++ are not sounded if an adjacent semitone is already sounding.

If there is silence and then the note F++ is sung then a decision needs to be made as to whether to sound the F++ as an F or an F#. This is decided as follows: an F# is sounded if the F# bucket has more votes than the F bucket, otherwise an F is sounded. A similar decision procedure is used if a jump to an F++ is made starting from a pitch more than a quarter-tone away - i.e. jumping to F++ other than from an adjacent F or F#.

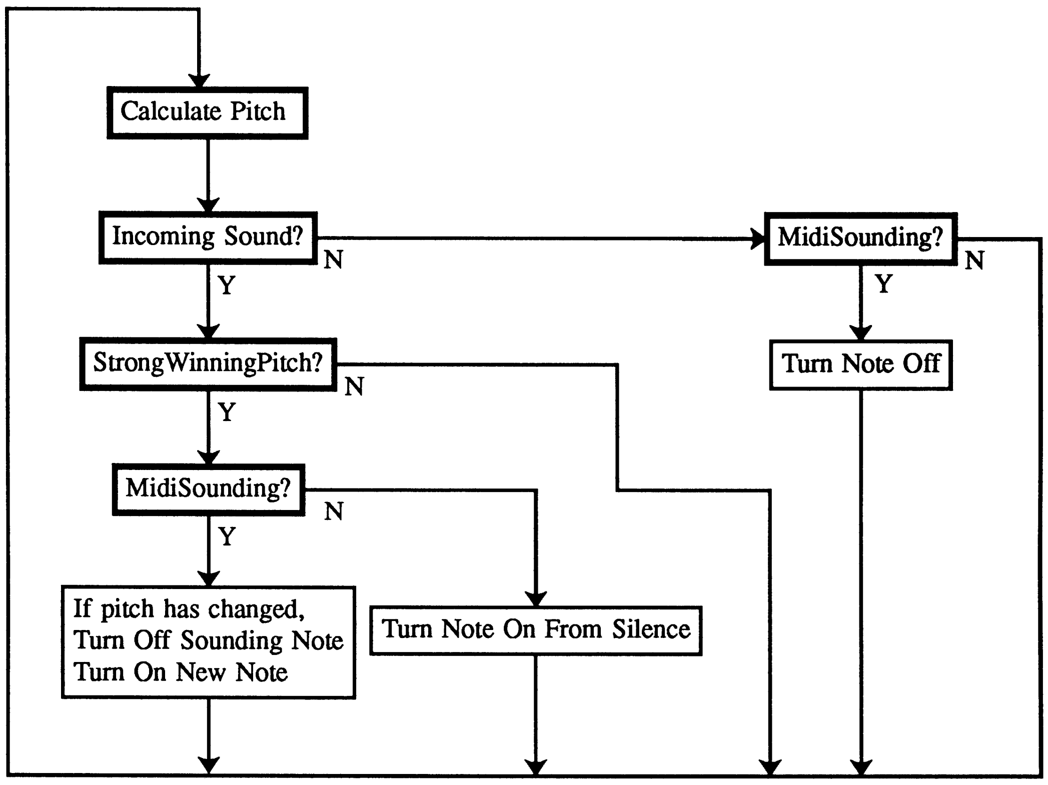

Figure 3 Algorithm Decision Procedure

6 Pitch Bend

If it is required that the Pitch to MIDI system exactly tracks the input audio pitch then MIDI pitch bend messages can be sent to adjust the output pitch to the input pitch. The output pitch can be rounded to the nearest quarter-tone or finer tuning can be obtained by interpolating between the votes in adjacent buckets.

7 Short Term Memory

To increase the robustness of the system, instead of clearing the votes after each frame, it is possible simply to halve them (or reduce them by some other ratio).

This gives the system some short term memory and allows the previous frame to influence the next one. This considerably stabilises the output.

8 Extensions When Information Is Available About The "Desired" Pitch

For some musical applications there is additional knowledge about the desired output of the system. For example the user may be singing a known song or trying to match a given pitch or pitches. Here there is information available additional to a single audio input and it is possible to bias the voting scheme to favour some candidate pitches over others. This can be achieved by enhancing or weakening the strength of the votes for the candidate pitches, by redirecting votes for some candidate pitches towards others, by adding positive or negative offsets to the accumulator buckets or by other means. Voting is a very flexible approach through which additional information can be used influence the response of the system.

9 Karaoke and Automatic-Accompaniment Applications

A Karaoke system which computes the pitch of the performer's singing could have a variety of features. For example it could provide a score showing how well the performer is doing, it could correct mistakes, add extra sounds or change the sound produced by the user, constrain the sound produced - for example to be in tune or in key, provide visual or other feedback, track the user's position in the song, accelerate or decelerate the playback of an accompaniment or respond in some other way. A sophisticated system may allow multiple performers to interact. In each case the system may use voting techniques to determine its response. One such system uses weighted voting for automatic accompaniment and is described in [Yue 96].

10 Conclusions

We have presented an enhanced voting scheme for the algorithm in [Sharp 93]. It eliminates flickering between adjacent notes and permits less memory to be used in the voting process. We have also suggested extensions to the voting scheme for use in other applications.

The algorithm enhancements described here have been implemented and tested. The resulting system provides a very good feel as you glissando upwards and downwards from one note to another: the semitone output system changes the note it sounds at just the right time and allows some wavering of the voice without wavering the output. For the pitch bend system the output pitch matches the input pitch with rapid response.

We speculate that the use of well-designed voting schemes will play an increasingly important role in systems which need to give a good "feel" when in use. Voting techniques appear to have the capability to generate appropriate responses to human and machine generated inputs.

11 References

[Sharp 93] D.W.N. Sharp, R.L. While, *Determining the Pitch Period of Speech Using No Multiplications*, **Proc. ICASSP '93**, Minneapolis, vol 2, pp.527-529, April 1993.

[Yue 96] M.K.Yue, *Score following using a voting mechanism*, **MSc Thesis**, Dept of Computing, Imperial College, London, 1996.