Real-Time Pitch Extraction of Acoustical Signals Using Windowing Approach

Farshad Arvin, Shyamala Doraisamy

Department of Multimedia, Faculty of Computer Science, University of Putra Malaysia
43400, Serdang, Selangor, Malaysia

Abstract: This paper presents a real-time signal processing technique based on a hardware interface using a microcontroller to process audio music signals for pitch extraction. A technique for transcribing music signals by extracting note parameters is described. The audio signal is divided into smaller sections known as windows to obtain samples of the signals for transcription. In general, two different approaches using static and dynamic window sizes to convert the voice samples for real-time processing are used. However, the transcription process involves complex calculations and in this paper we proposed a simple technique to estimate fundamental frequency of given sound signals. The transcribed data generated shows the feasibility of using microcontrollers for real-time MIDI generation hardware interface.

Key words- Signal processing, MIDI, Microcontroller, Transcription, Pitch extraction.

INTRODUCTION

Before the invention of digital based music notation systems, paper-based musical notations and scores have been used for communicating musical ideas and compositions. Digital-based music encoding has simplifying editing, processing, and communication of musical scores. Music data are multi-dimensional; musical sounds are commonly described by their pitch, duration, dynamics and timbre. Most music database and data management systems use one or two dimensions and these vary based on types of users and queries. There are many formats in which music data can be digitally encoded. These formats are generally categorized into:

- Highly structured formats such as Humdrum (Eleanor Selfidge-Fied, 1997) where every piece of musical information on a piece of musical score is encoded.
- Semi-structured formats such as MIDI in which sound event information is encoded.
- Highly unstructured raw audio which encodes only the sound energy level over time.

Most current digital musical data management systems adopt a particular format and therefore queries and indexing techniques are based upon the dimensions of music information that can be extracted or inferred from that particular encoding method.

Systems such as voice to MIDI, whereby format conversion and transcription of real-time audio signals to MIDI data stream are required, are currently being research and developed widely (Mark Nelson, 2006; Hai et al., 2002; Ghias et al., 1995). MIDI is a popular format in computer-based music production such as composition, transcription, etc. Size of MIDI files are generally smaller than other music formats, because MIDI data consists of text messages that are defined instructions only and not sounds signal representation. It is a suitable data format to be utilized in software application development. With Voice to MIDI system, we will be able to generate MIDI data of analog input rather easily. This system will enable conversion of a melody input with a microphone to digital scores like MIDI. It extracts acoustical characteristics such as pitch, volume, and duration by intelligent algorithms and converts them into a sequence of notes for producing music scores. Thus, melodies will be translated into chromatic pitches without human intervention (Itou and Nishimoto, 2007).

Challenges faced commonly in voice to MIDI systems include the clarity of input data for acquiring suitable results. In some systems such as (Jun et al., 2004), we must sing music simply with “ta ta ta…” expressively singing to prevent many inaccurate outputs. So, in those methods we are being forced to sing unnaturally.

The quality of MIDI transcribed would also depend on the hardware capabilities. Also, some systems are
based on software that needs intelligent algorithms for providing better quality music transcriptions. However, most of these systems use Digital Signal Processor (DSP) to process audio signals. A microcontroller as the main processor to process real-time audio signals is investigated in this study. The proposed technique is implemented completely with this microcontroller without a need for much complex calculations.

In this paper, we introduce an audio to MIDI transcription module encompassing a microcontroller and a pitch tracking algorithm. A hardware based real-time converter is offered which uses a real-time technique for implementing relatively good MIDI generator for transcription. The main aim is to extract music information from the voice signals to convert to MIDI representation. This hardware must be able to capture input signals that would enable estimation of parameters such as pitch, note onset time, and duration, from the audio signals and generates MIDI messages. Two different approaches to provide the MIDI events from real-time signals are proposed. Both approaches are implemented with the microcontroller processor and comparison of outputs will be described in this paper.

**Windowing Approaches:**

Two differing approaches for determining sample window sizes are proposed for our real-time signal processing technique. The signal processing technique uses the pitch extraction algorithm introduced in (Arvin and Doraisamy, 2009). When a music note is played, there are several repetitions of a given frequency as shown in Fig. 1 (a). The captured signals are converted to rectangular pulses based on wave’s energy level using defined constant threshold as shown in Fig.1 (b). The musical sound is a composite of many harmonics. While the fundamental frequency gives the sound its pitch, the harmonics give the sound its characteristic timbre. The sounds of a violin and piano are different even if they are all playing the similar pitch. The difference is caused by the complex mixture of harmonics from each instrument as shown in Fig. 2. Identifying the fundamental frequency is an important part of the music transcription. The main idea of proposed technique is extracting fundamental frequency of music sounds. Hence, captured samples will be saved in time duration which is called windows. The note extraction function processes the captured samples during each window.

**Static Window Size:**

In this windowing approach, captured samples will be divided to fixed size windows as shown in Fig. 3. In each window, $\Delta t$ is the time duration between a rising and falling edge of a given pulse. One of the microcontroller’s timers is used for counting the duration of each pulse. A two-dimensional array is used for recording captured samples. The counted durations are saved as the first dimension and the start time of each pulse as the second dimension. At the end of each window’s duration, the frequency calculation function will be called to estimate the fundamental frequency. This function will find several maximum $\Delta ts$ in the samples array. The distance of these maximum values are used for the frequency calculation. The fundamental frequency of each played note will be calculated by the distance of two similar maximum pulse widths. This sequence of the static sized windowing approach is shown in Fig. 4.

The size of each window is a significant parameter for estimating of fundamental frequency in this approach. The played notes durations are the other important parameter in extract processing function. When the size of windows gets bigger, a lot of samples will be captured and output will be near to real frequency. But if the window size is large, the processor will not have enough time to process the captured data. In addition, selecting a big size window, result to losing notes with short durations.

**Dynamic Window Size:**

In this second windowing approach, the window size is not a constant. It may differ in each sampling period as illustrated in Fig. 5. The fundamental frequency extraction function in this approach is similar with the static window approach. The distance of each maximum pulse is used for estimating fundamental frequencies. The implementation of the dynamic window approach is more complex than the static window approach. Several parameters will be used to change window’s size.

The first reason for changing window sizes is to capture similar maximum pulse widths. When two equal maximum pulses are captured in one period, the window can be closed and a new window can begin. In this approach, the notes with short durations will not be ignored. This function is implemented with a simple counter variable and a maximum pulse finding process will be added to the real-time algorithm.

The second reason that helps find the ideal window size is the gap between two played notes. This function will detect silent areas between played notes when it can’t find any normal pulse width within the timer period. The normal pulse width is a constant value which is defined in the microcontroller source programs.
One other parameter that will also help to estimate window sizes is using a time-out approach for each window. For the implementation of this algorithm, another timer of the microcontroller will be employed for time-out management. This timer will work with windowing approach simultaneously. The function will end the current window processing if it does not receive any maximum pulse width its timer period. Consequently, if all captured pulses are smaller than the music note frequencies, the active window will be closed and new window will begin for capturing process.

Three parameters, i) similar maximum pulse width, ii) gap detection, and iii) time-out value, were described for changing the window sizes. Furthermore, several complex parameters can be used for changing window size but implementation of these algorithms are complicated and would need very high speed processors such as DSP.

\[ N(p) = 40 \log(f(p)) / 261.6} + 60 \]  

The MIDI note number 60 is the musical note name C4 with frequency value 261.6 Hz that is middle C note in musical instruments. This formula indicates, if the value \( f(p) \) is increased to 2 times, the value of 12 which is an octave interval will be added to the \( N(p) \).

Each calculated section is equivalent to one musical note, and will generate MIDI codes based on the standard MIDI file rules (Modegil and Lisaku, 1998). In general there are two types of commands in MIDI
event data, which are a Note-On and a Note-Off, and a Delta Time value must be calculated before each command as follows:

\[
\text{Delta Time 1, Note-On, Note Number 1, Velocity 1} \\
\text{Delta Time 2, Note-Off, Note Number 2, Velocity 2}
\]

The defined MIDI representation for Note-On command is the hexadecimal value “9x”. “8x” is defined for the Note-Off command. In these commands, \(x\) shows channel number of notes, Note Number 1 and 2 is the value calculated using formula (1), and Velocity 1 and 2 is given by formula (2) below:

\[
sqrt{\left(V_{\text{max}} (s)\right)} - 127
\]  

(2)

Architecture of Hardware:

Several simple hardware components are required for our proposed architecture in implementing the algorithms. The architecture includes an ATMEGA88 microcontroller as the main processor with an external 20 MHz oscillator employed to execute the functions for MIDI data extraction. The programs were implemented with C programming language and several low-level functions were implemented with assembly language. Several tests preformed show that, the assembly routines provide better results than C functions for certain critical routines. Thus, timers and ADC (Analog to Digital Converter) interrupt routines were written with assembly instructions to prevent probable errors in extraction calculations.

The sequence for the real-time processing is shown in Fig. 6. The samples captured with microphone, will be amplified with a pre-amplifier unit. After amplification of the input signal, the digital state of the signal will be extracted. The digital state of each signal is pulses with different pulse widths are an important parameter in implementing the note extraction. These modules were implemented with analog electronic components. The next stage in implementation is processing units based on the microcontroller’s source programs. These codes include several functions that work sequentially for calculating fundamental frequencies within the windows.

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. This device is used for transmitting MIDI events to PC. The serial RS232 protocol is selected for communicating between the designed hardware and PC [9]. The hardware sends MIDI contents as serial bits in 38.4 Kbps. Therefore, a TXD interrupt should be enabled to prevent data loss. Fig. 7 illustrates hardware of designed interface.

Fig. 4: Block diagram of static size windowing approach
Experimental Results:

The outputs of the proposed approaches are shown in Table I. In these experiments, duration times of 20 and 100 ms were selected to process the static window capturing technique. In each algorithm, 200 notes were played and results were obtained. The percentage of correctly transcribed notes are shown in two note duration, short duration notes and long duration notes. The static window capturing technique has different results for both note durations. The dynamic window capturing technique provides ideal results in all experiments. This technique improves transcription performance for both short and long duration notes.

Fig. 8 illustrates played notes and the output of proposed algorithms. Fig.8 (a) is sample notes which were played with electronic keyboard. We used two note durations at different frequency levels. Fig.8 (b) shows results of static window approach of 20 ms period. The transcription result of this window size is quite close to the actual played notes. However it is still not close enough. Fig.8 (c) is the result of static capturing algorithm with 100 ms period. This window size provides good results for long duration notes but it is a lossy for short duration notes. Fig.8 (d) is the dynamic capturing algorithm results. This algorithm has good performance in the transcription process. The dynamic windowing approach can extract pitch numbers more precisely for higher frequency notes. In addition, we will obtain best results with this technique if several dynamic window size calculation functions are employed.

Conclusion:

This paper presents a real-time signal processing technique for note transcription based on the MIDI format using a simple hardware interface. Using simple algorithms for signal processing and a digital circuit based on microcontroller without the need of a DSP, is shown to be feasible in this paper. The experimental results show that the proposed hardware generates good transcribed outputs for played notes with electronic keyboard. Static and dynamic window size techniques were proposed in this paper. The dynamic capturing technique provides transcribed notes that are close to the actual played notes. This hardware will be used for other real-time voice processing applications. For future work, other windowing functions will be defined to obtain better results in all note durations. This technique and a learning function to detect silent areas using previous pulse widths can also be improved for a multi-track MIDI format. In addition, it can also solve voice processing problems in robotic environments with this MIDI transcription ability.
Fig. 7: Main board of design interface

Fig. 8: (a) original played notes with electronic keyboard (b) generated notes with static window of 20ms (c) generated notes with static window of 100ms (d) extracted notes with dynamic window approach

Table 1: Percentage of Correctly Transcribed Notes

<table>
<thead>
<tr>
<th>Played Notes Duration</th>
<th>Static Window Size</th>
<th>Dynamic Window Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td>Short Duration</td>
<td>67%</td>
<td>50%</td>
</tr>
<tr>
<td>Long Duration</td>
<td>70%</td>
<td>75%</td>
</tr>
</tbody>
</table>

REFERENCES


