Study for a Real-Time Voice-to-Synthesised-Sound Converter

Claudio Galmonte, Dimitrij Hmeljak
Dipartimento di Elettrotecnica, Elettronica ed Informatica,
Universita' di Trieste, via A.Valerio 10, 34127 Trieste, Italy

Abstract

In this paper we describe a system capable of a real time conversion of pitch and dynamics from a vocal melodic input to a synthesised sound output, with on-screen graphical visual control.

Different pitch recognition algorithms (Fast Fourier Transform, Average Magnitude Difference Function and Fundamental Period Measurement) are implemented in the system. Voice signal sampling/buffering and sound synthesis routines are based on OS-level techniques necessary to achieve a real-time performance and compatibility among general-purpose machines. A version with optimised routines for lower-level machines has been written in assembly. FPU or DSP coprocessors are not needed.

1 Introduction

1.1 Problem Statement

Despite the availability of powerful hardware for Digital Signal Processing, high-performance voice-to-synthesised-sound conversion is still an open point. The effectiveness of a practical solution depends on many mutually interconnected factors. The problems we decided to address are, in order of importance:

1. *Real time pitch recognition*. Algorithms used for the extraction of pitch from a vocal signal must perform adequately, according to logarithmic precision, matching the resolution of the human ear. The computational load should not prevent real-time use of the system, both for the time necessary for the pitch value computation as for the minimal resolution time period required.
2. *Real time sound synthesis*. The sound output controlling process should perform smoothly, concurrently to the input signal sampling and signal processing processes.
3. *Portability* and *low system requirements*. To be a viable alternative to dedicated hardware devices, a software tool performing voice-to-synthesised-sound conversion should run on a wide base of general-purpose machines. Additional hardware might be not easily available or too expensive, therefore limiting the usefulness of the software system.

1.2 Related Research

Effective solutions to the pitch recognition problem are to be found mainly in classical voice analysis studies (for authoritative references, see [5], [4], [2]) for non-musical purposes. The methods and results of this sort of research, however, will have to be carefully adapted to

musical use, if they are to be part of a solution to our goals, because of different requirements (e.g. logarithmic vs. linear precision).

Some of the past research in the musical field has addressed the similar problem of tracking different melodies from a polyphonic source [8], [10]. These works, although useful from a theoretical point of view, are not practically useful for a real-time implementation on general-purpose hardware. An interesting method for melody tracking has been presented in [12], and result data from this research suggest a precise extraction of the melody - although they require a computational load beyond the limits posed by our goals.

A useful study for a real-time pitch recognition for music applications is [7]. The author presents a conceptually simple yet effective solution (the Fundamental Period Measurement method), that can be implemented balancing precision and speed - we used this algorithm as one of the pitch extraction options in our system.

1.3 Our Work

We approached the problem of real-time conversion by developing a system on which to perform both testing and real-time implementation. The complete system has been developed in software (C language and assembly) on Macintosh Quadra computers running MacOS, and tested on different Macintosh models. A research activity partially resulting from the work conducted on this system is presented in a different section of this Conference [6].

Distinct pitch-recognition algorithms have been tested and compared: Fast Fourier Transform, Average Magnitude Difference Function and Fundamental Period Measurement. Voice signal sampling/buffering and sound synthesis have been implemented to run concurrently as background processes, while analysis algorithm communicate with them through semaphores and circular buffers. Sound Manager routines of the MacOS perform these tasks, assuring compatibility among different A/D and D/A hardware used on different Macintosh models. Minimal requirements for the system are separate A/D and D/A hardware built-in subsystems (i.e. MacIIvx-class, but not LCIII-class machines).

The computational-intensive routines have then been manually optimised for the real-time performance on lower-class machines: while the FFT+AMDF method-based routines still require a MC68040-class machine, FPM routines have been ported on MC68030 machines.

We compared our system with some commercial voice-to-MIDI hardware converters connected to MIDI synthesisers, experimenting with various voices. In terms of speed, precision and response time, our system performed better than the low-class hardware devices, being a cheaper and portable software-only product.

2 Pitch Recognition Algorithms

A possible classification of pitch extractors can be based on different definitions of the pitch of the human voice. We will not go into details of perception-based definitions that can lead to concepts as *spectral pitch* and *virtual pitch*. It is more useful for our purposes to focus on these two possible definitions:

• *pitch-period* T_0 is defined as the mean of time periods measured during a short emission of pulses by the larynx

• *pitch-frequency* f_0 is defined as the fundamental frequency of a short-term spectral representation of the voice signal

It is clear from these definitions that 1) a pitch extractor must operate on short portions of the signal, extract a pitch value and then repeat the calculation for another portion, until the end of the voice signal is reached (therefore a *short-term* analysis); and that 2) a pitch extractor can operate directly on the time-representation of the signal or on data obtained from a transformation of the original signal.

Due to the real-time constraint imposed by our goals, we could not use more computation-consuming methods that operate entirely on frequency-domain representation of the signal. Nevertheless, using a combination of time-domain and frequency-domain algorithms, some interesting results have been achieved.

2.1 Speed, Accuracy and Resolution

The main issue in a real-time system is speed: the minimum requirements must be satisfied in a time not exceeding time constrains.

A first requirement to obtain a pitch estimation is caused by the non-linear resolution of the human ear - the tempered chromatic scale in western music is composed of logarithmically-spaced (half tones are *5% f* intervals) rather than evenly-spaced frequency intervals. To discern single semitones, a constant-percent precision of less than *2%* must be maintained throughout the operative range (we can safely limit the pitch range to the 3 octaves from f_l=87Hz to f_h=784Hz [7]). This requires a careful tuning of frequency-domain algorithms in the lower range of the spectrum, while time-domain algorithms show their limits in the higher range of the spectrum.

The other resolution we have to deal with is caused by the ability of the human ear to discern rapid variations in a melody. For our system, we arbitrarily chose a 4/4 90 beat/minute melody, with sixteenth notes to track, to be the fastest changing recognisable input. Two samples per note, the minimum to perform the tracking, would lead to 12 samples/second. Sampling the pitch at a global f_{pitch} = 20 Hz rate would be a safer choice - although over-exceeding this value can lead to other problems: if $f_{pitch} > 3/2\ f_l$, a pitch period can not be discerned. We encompassed these constraints choosing f_{pitch} = 50 Hz for the fastest-needed computations, maintaining the global f_{pitch} above 20 Hz.

Algorithm processings, in real-time implementation, have one more constraint: the delay occurring between an input (trigger) signal event and the output (effect) result. For practical musical use, this delay can be disturbing to the performing musician even when it reaches a 50 msec length [7], [3] - independently from the global f_{pitch} sample rate achieved by the tracking routines.

2.2 FFT+AMDF Algorithm

This method uses a combination of two popular pitch-recognition techniques. The first one is based on the FFT algorithm used to compute the *power spectrum* of the sampled input signal, from which the pitch-frequency is estimated [9]. To gain the desired frequency resolution in the lower range of the spectrum, however, a 1024-point FFT has to be performed. This results in an overwhelming 409.600 multiply-and-add operations per second to maintain the necessary f_{pitch}.

Yet, a time-domain technique based on the *optimum-comb filter* search by means of the AMDF function calculation can be usefully combined with the FFT algorithm. The AMDF operates on time-shifts of the input signal, not requiring domain transformations. Again, the logarithmic precision poses a challenging constraint. To gain it at the higher end of the frequency range, a high signal-sampling frequency f_s is needed: f_s = 22 kHz corresponds to a difference of two samples between adjacent semitones in the 750 Hz range. At that f_s value, the AMDF requires about 1.000.000 operations per second to maintain the necessary f_{pitch}.

We combined the two methods in a sequential processing scheme: first, a 512-points FFT is used to give a rough estimation of the pitch-frequency. This value is then considered as the centre of a frequency range where the AMDF can be used to gain the necessary precision. In this way both algorithms operate at a reduced load and, above all, another shortcoming of the AMDF is addressed: the inherent inability to discern octaves. Using the FFT as range limiter, this problem is happily by-passed.

2.3 FPM Algorithm

This method, described in [7], is based on the assumption that the singing-voice signal is quasi-periodic. Using a bank of bandpass or sequentially-combined lowpass filters, a set of pseudo-sinusoidal signals is obtained. The fundamental period of these signals is computed measuring the number of samples in zero-crossing time frames.

The method is particularly interesting for real-time applications, since it can be optimised in many ways, to obtain a fast implementation. The cascade combination of low-pass filters allows a progressive interpolation of the input signal sampling frequency. Filters can be realised in

integer arithmetic, and subsequently optimised for a particular target processor. This optimisation is explained in detail in a following paragraph.

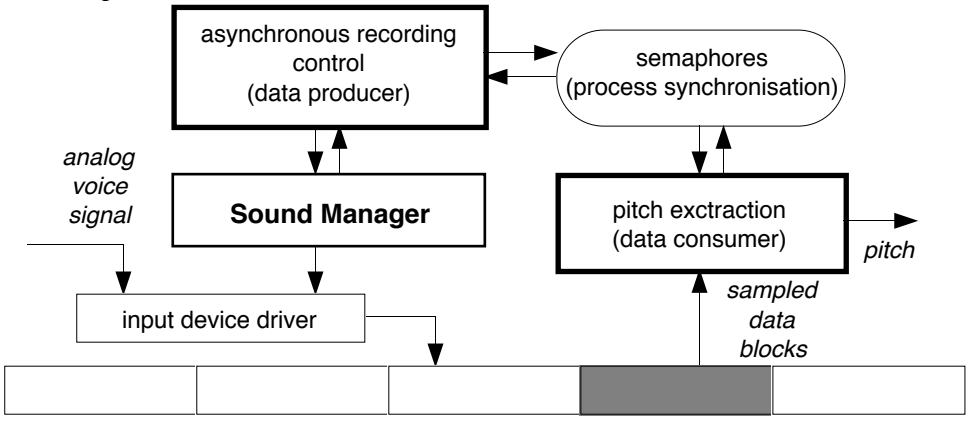
3 Architecture and Implementation of the Converter

The converter can be functionally decomposed into main blocks:
• *Input signal interface and pre-processing*. The signal, sampled and stored digitally in an input buffer, has to be subsequently prepared for easy retrieval for pitch-extraction routines. A circular-buffer implementation provides an efficient exchanging mechanism for this purpose.
• *Pitch-extraction*. Being the computational-intensive part of the system, critical for the correct timings, this block operates as the main foreground process of the system, triggered by input data, passing output values to the output routines.
• *Sound synthesis routines*. The amount of input data to this block is relatively low, compared to the information processed in previous parts: the output block deals with higher-level values for pitch, duration and volume.

3.1 Signal-Sampling and Pitch-Extraction Synchronisation

Sampling routines have been implemented using the *asynchronous recording* mechanism built into the Sound Manager (*SM*) part of the MacOS operating system. Using the Sound Manager, we maintain the system's independence from underlying A/D hardware: drivers provided for the MacOS, together with SM calls, act as interface to the programmer. All that is required from the input device and driver is a sampling rate of at least 22 kHz, 8 bit linear resolution (after AGC compression of the signal), and the ability of operating in background. All Macintoshes provided with sound input interface support these characteristics, as well as most common third party add-on sound interfaces.

SM routines perform accordingly to configuration stored in the Sound input Parameter Block. Most importantly, in the *SPB* are stored:
• the *location* in memory where sampled data has to be stored
• the *duration*, in bytes or milliseconds, of data to be *recorded* (sampled analogue input signal)
• the *completion* and *interrupt* routines starting addresses
These data are first used by the device driver activation routine SPBOpenDevice, which sets the operation mode of the sound input hardware. Particularly, the *interrupt* and *completion* routines permit a safe process synchronisation of the sampling routines with the main data processing to be performed. Synchronisation is implemented with a typical producer-consumer algorithm and *generalised semaphores* [13].

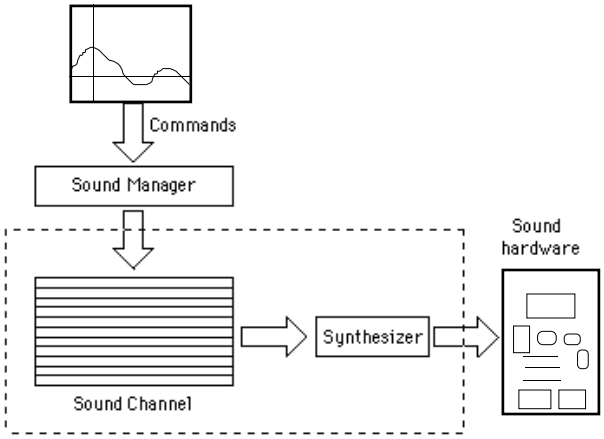


Whenever the internal buffer of the sampling hardware is full, the interrupt routine is triggered, which can access globals in the main program data space. The interrupt routine sets the producer semaphore to tell the consumer process (pitch extractor) if and where fresh data has been stored in the buffer, and subsequently returns. The producer process waits in idle state until enough data is produced, then it starts the pitch extraction routine. If the CPU is not fast enough to process all the data in time, there is no easy way around: due to the nature of pitch-

extracting algorithms used, it is not desirable to skip sampled data that can not be analysed. In that case, there is simply no output.
The circular buffer is easily implemented without worrying about sample loss at the end of the linear memory space, when some pointer reinizialization occurs, thanks to another feature of the SM: the *continuous* recording mode. When the sound recording background routine reaches the end of the buffer, it signals this fact by calling the completion routine, and releases control - continuing to sample data in the input device's internal buffer. If the recording is restarted before this buffer is filled, there is no loss of data.

3.2 Sound Synthesis and Control

Sound synthesis is performed by hardware controlled through Sound Manager output routines. While all current Macintosh computers incorporate sound output capable of asynchronous synthesis, freeing the CPU to perform other tasks, not all of them allow simultaneous background sound input and output. We were able to successfully test these routines on most Quadra, as well as MacIIvx-class machines, but not on LCIII-class machines. On the latter, sound input inhibits sound output and vice versa.
Sound output control is allowed at different Sound Managers. The method used in our work consists in storing sound commands in a FIFO queue called Sound Channel [1].



This queue is read by one to four software *synthesisers,* which can be initialized in different ways: as a *sampleSynth* which plays arbitrarily-length samples at a desired frequency, as a *wave table* synthesiser which generates sounds based on 512-sample data, or as simple *beeps* of desired height.
Best results are achieved using the *sampleSynth*, although the sound quality depends on the capability of the connected or built-in hardware, which varies among the range of machines.

3.3 Performance Optimisations

To perform a smooth real-time pitch conversion, the computing-intensive routines of the system have been optimised in successive steps. The FPM algorithm is well suited to this type of improvement [7]. The fastest implementation of the filters bank is achieved using a cascade of low-pass second-order filter: the subsequent lowering of the highest frequency in the sampled signal allows a drastic reduction of the required computations, by decimating the sampling frequency. This observation alone reduces the number of required multiple-and-add operations to almost a quarter of the original load.
A further step towards real-time execution is the utilisation of integer-only arithmetic. While errors due to filter coefficient quantization are minimised by the consistent use of second-order filters [11], overflows in multiply operations can only be avoided by carefully coding the filters. Shift operations on signal data before and after each arithmetic operation have to preserve most of less-significant digits and completely avoid the overflow of most-significant digits.
After the first successful C coding of the filters for a real-time performance on MC68040-based computers, we have addressed the problem of obtaining the same result on less powerful

MC68030 CPUs. In the MC68040, the multiply operation does not weight much more than the add operation, in terms of machine cycles required. Not so on the MC68030, where 44 cycles are needed to multiply two numbers, vs. only 2 cycles for an add operation. An analysis of total needed cycles for the complete scan of the circular buffer (200 kB of data) proves that the multiply operations represent the biggest computational load:

operation	+	shift	*	/	=	if
# called	5631750	3862800	3668850	7200	5117400	1349100
CPU cycles	2	6	44	90	2	8
total	11263500	23176800	161429400	648.000	10234800	10792800

To achieve real-time performance on these CPUs, multiply operands have been substituted by shift-and-add operations, coding the filters directly in assembly language. By coding each multiply operation with two to four shift-and-adds, the average load has been reduced to 58% of the original code, thus permitting real-time execution on MC68030 machines.

4 Conclusions and Future Work

The converter has been compared in real-time use to two low-class commercial hardware products: the Digigram MidiMic and the Roland CP-40, both driving a MIDI synthesiser. Although these products can track also non chromatic voice expressions, their ability to track fast pitch variations is considerably limited, probably due to less available computing resources and MIDI protocol overhead. In terms of speed, precision and response time, our system performed better, being a cheaper and portable software-only product.
The development of the system used for our research was conducted on machines equipped with MacOS Sound Manager version 2.0. The release of the version 3.0 expands the possibility of sound sampling hardware, and the subsequent release of QuickTime 2.0 allows Macintosh computers to emulate a MIDI synthesiser using on-board hardware only. A future redesign of our system might lead to a better use of the available sound hardware resources, while maintaining portability and moderate load on the processor.

References

1 • Apple Computer, Inc. Inside Mac vol.VI. Cupertino, USA: Addison-Wesley; 1991.
2 • Boite, Rene'; Kunt, Murat. Traitement de la parole. Lausanne: Presses Polytechniques Romandes; 1987.
3 • De Furia, Steve; Scacciaferro, Joe. The MIDI Book, Using MIDI and Related Interfaces. Pompton Lakes, N.J.: Third Earth Publishing, Inc.; 1988.
4 • Furui, Sadaoki; Sondhi, Mohan M.; et al. Advances in Speech Signal Processing. Murray Hill, NJ, USA: Marcel Dekker, Inc; 1991.
5 • Hess,Wolfgang.Pitch Determination of Speech Signals.Berlin:Springer-Ferlag; 1983.
6 • Hmeljak, Dimitrij; Zanon, Domenico. Tools for the analysis of alterations in vocal musical performance. Ferrara: CIARM95; 1995.
7 • Kuhn, William B. A Real-Time Pitch Recognition Algorithm for Music Applicatons. Computer Music Journal; 1990; 14(3).
8 • Moorer, James A. On the Transcription of Musical Sound by Computer. Computer Music Journal; 1977; (3): Menlo Park, USA.
9 • Oppenheim,A.;Shafer, R. Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall; 1975.
10 • Piszczalski, Martin; Galler, Bernard A. Automatic Music Transcription. Computer Music Journal; 1977; (3): Menlo Park, USA.
11 • Proakis, John G.; Manolakis, Dimitris I. Introduction to Digital Signal Processing. New York: MacMillan; 1988.
12 • Rodet, Xavier; Doval, Boris. Estimation of Fundamental Frequency of Musical Sound Signals. ICASSP Proceedings; 1991: 3657-3660.
13 • Wirth, Niklaus. Algorithms + data structures = programs. Englewood Cliffs ,N.J.: Prentice - Hall; 1976.