Singing Voice Impersonator Application for PC

Maarten de Boer, Jordi Bonada, Pedro Cano, Alex Loscos, Xavier Serra
Audiovisual Institute, Pompeu Fabra University
Rambla 31, 08002 Barcelona, Spain
{mdeboer, jboni, pcano, aloscos, xserra}@iua.upf.es
http://www.iua.upf.es

Abstract

This paper presents the implementation aspects of a real-time system for morphing two voices in the context of a karaoke application. It describes the software design and implementation under a PC platform and it discusses platform specific issues to attain the minimum system delay.

1. Introduction

Singing voice conversion can be defined as the conversion of the voice quality from one singer to another. The impersonator application we present in this paper makes use of a conversion that gives a specific individuality to the synthesized voice. The goal is to come up with a karaoke-type application for PC in which the user can sing like his/her favorite singers.

In order to incorporate to the user's voice the corresponding characteristics of the "target" voice, the "target" voice is recorded and analyzed beforehand. Then the system performs a real-time morphing between the user's voice and the pre-stored analysis of the target voice. The user is able to control the degree of morphing, thus being able to choose the level of "impersonation" that he/she wants to accomplish.

The application relies on two main algorithms that define and constrict the architecture: the Spectral Modeling Synthesis (SMS) (Serra, 1997) and a Speech Recognizer. This paper presents the practical implementation aspects of such an application. The theoretical background can be found in other papers (Cano, Loscos, 1999; Loscos, Cano, Bonada, 1999; Cano, Loscos, Bonada, de Boer, Serra, 2000).

2. Implementation

The application has been implemented mostly in ANSI C++, in order to be easily portable to different operating systems. For some time, development has been focusing on the Windows platform only, but now it compiles under Linux as well, thus adding a second development platform. One of the purposes was to compare latencies between Windows and Linux, using the same hardware, in non-real-time and real-time implementations. A flow diagram with the most important classes, structures, and procedures is shown in figure 1.

On both platforms the application uses multi-threading. The approach shown in figure 1 turned out to be the most efficient, in terms of latency, as well as the least sensitive for underruns. The Sound Thread makes use of a tight loop in which the audio is read and written from the soundcard. This ensures that when the processing does not keep up with the sound input, something that occasionally happens, sound throughput continues. Every read buffer of input samples is passed to the Analysis Thread, and when a new frame has been analyzed, the Synthesis Thread is signaled, which writes into the output buffer, that has been prepared for playback. The GUI is running in a third thread with low priority, and can modify the synthesis and analysis parameters used by the Analysis Thread and Synthesis Thread. Thread priority for the Sound Thread, Analysis Thread and Synthesis Thread are set to maximum, and memory pages are locked.

3. Graphical User Interface

The graphical interface of the application has been thought as a test platform from which all relevant parameters could be modified in real time. As shown in figure 2, the application has got six main sections. The analysis section controls some of the analysis parameters such as the lowest and highest pitch, the analysis window size or the residual model used.

In the morph section five sliders control the interpolation values of the attributes involved the main part in the morph. These are the pitch, the amplitude of the sinusoidal part, the amplitude of the residual part, the spectral shape of the sinusoidal part, and the spectral shape of the residual part.

The sine and residual sections control the most important synthesis parameters of SMS and also include some graphic filtering capabilities. The two sections left, the harmonizer section and the IR section, were included to test some of the effects we were working on.

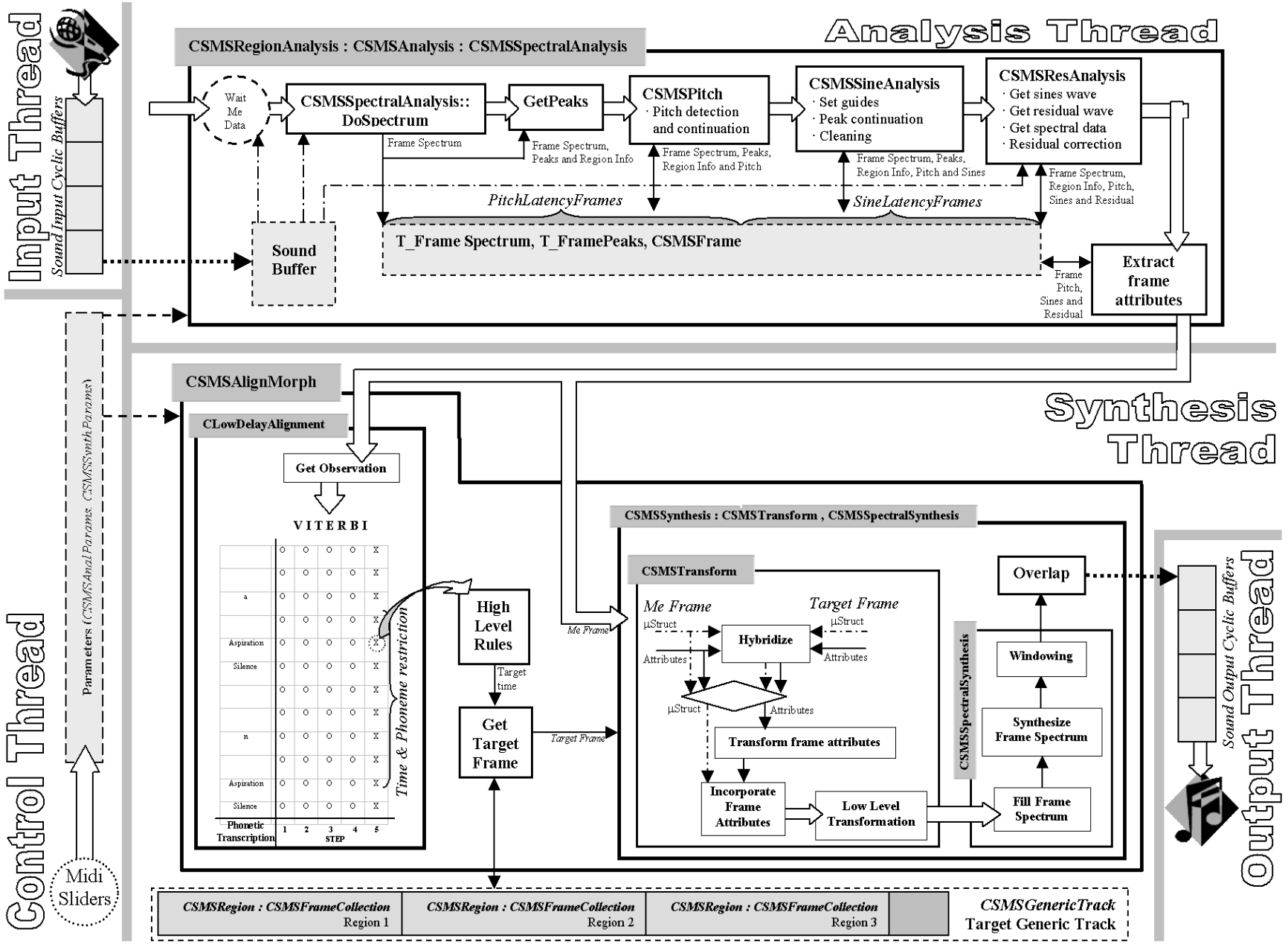

Figure 1. System architecture diagram.

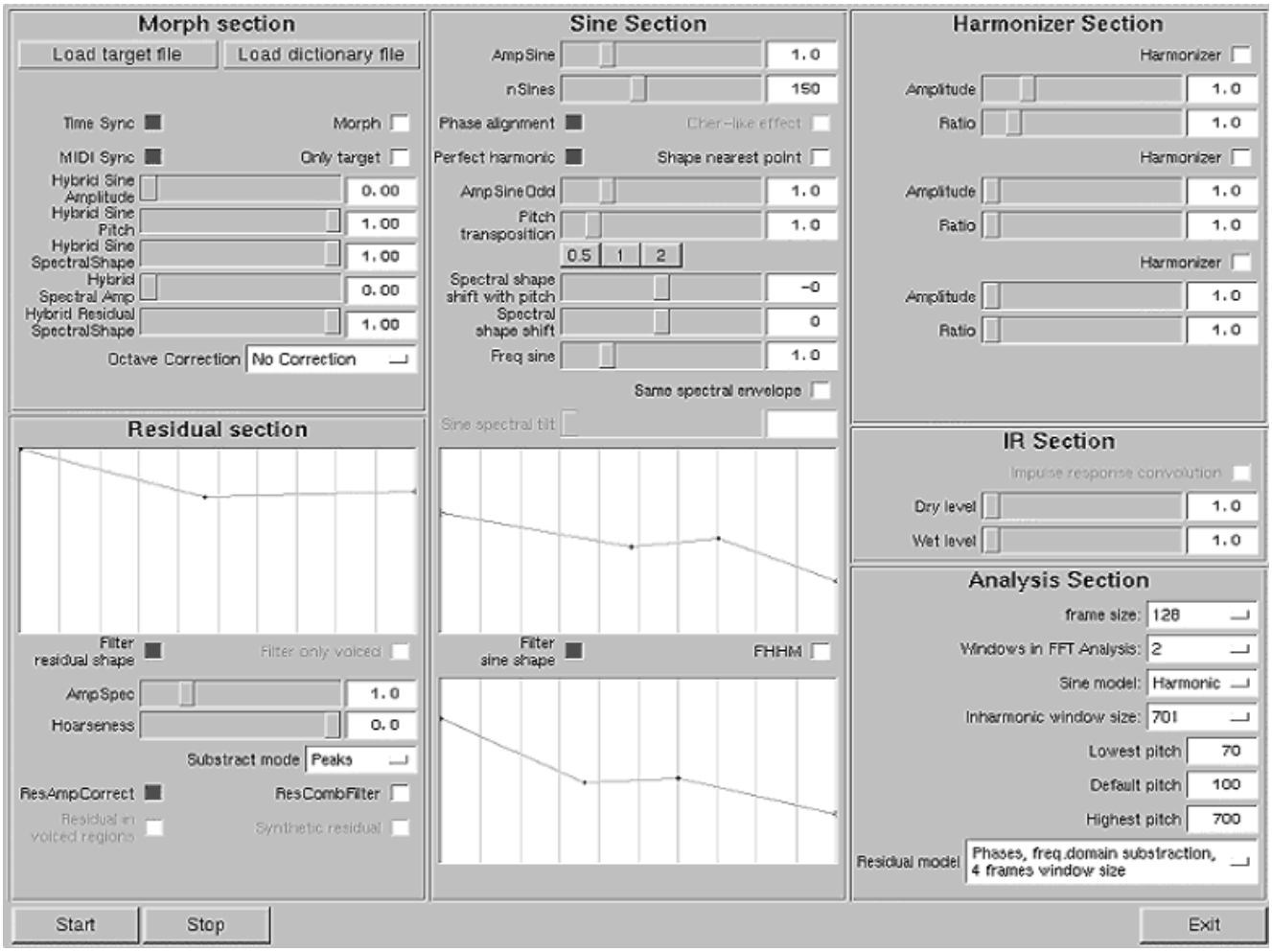

Figure 2. GUI screenshot.

4. Scoring of pitch and timing accuracy

We included in our application one of the features that many karaoke systems have; a score indicator to show how well the user is singing at each instant. To do so the system measures the pitch and timing of the user, comparing it with some reference values. The pitch and timing chosen as reference are not from the target (the target singer does not necessarily sing perfect) but the ones included in the song's MIDI representation.

In order to do the comparison, we have to know which is the note that the user is singing. The morph-alignment module tells us which region of the target the user is at. From there we should be able to find out the corresponding note information, by giving every region a note reference. Instead of doing this by hand, this is done automatically be adding the phonetic label found in each of the regions of the target to the MIDI file containing the correct melody.

Representation of the regions with the corresponding phonetic transcription (the dot means silence):

Note information from midi file:

id	begin (sec)	duration (sec)	pitch (Hz)
0	0.20	0.23	174.61
1	0.43	0.22	196.00
2	0.65	0.22	207.65
3	0.88	0.22	261.63
4	1.11	1.04	349.23

Add phonetic lyrics to the midi file:

id	begin (sec)	duration (sec)	pitch (Hz)	transcription
0	0.20	0.23	174.61	na
1	0.43	0.22	196.00	ki
2	0.65	0.22	207.65	na
3	0.88	0.22	261.63	ka
4	1.11	1.04	349.23	la

The automated mapping matches regions and id's in the following way:

From the analysis/synthesis loop, we obtain the user's fundamental frequency and time, and the target region used for the morph. This target region contains a reference to a note ("note id") with the pitch and timing information. A history of these note id's is kept, so when a certain number of the same note id's has been reached, we presume that this is the note the user is singing (current note id). We use the time the current note id changes, as the begin time of the user note, and compare it with the reference note, and can compare the pitch continuously. The scoring is then calculated from the pitch difference in semitones:

$$Dif_{semitones} = \left| \left(12 \log \left(freq_{user} / freq_{reference} \right) - 6 \right) \% 12 \right|$$

When the timing error is within a 0.1 second, maximum score is assigned. If the error is bigger than that, score goes from 1 (0.1 sec) to 0 (0.5 sec). A lowpass filter is applied on these scores to filter out sudden changes that are likely to be caused by erroneous analysis. Apart from this current score, also an overall score is kept.

We tried to improve the visual presentation of the scoring. The presentation with sliders is neither visually attractive nor inspiring for the singer. Because of that we implemented an animation-style scoring indicator, with seven different levels of scoring, each with a small variation to make it less static. The character in the animation is an animation-style Elvis-alike figure that takes the role of the singer as shown in figure 3. The character is sad/desperate when the user is singing badly and happy when the singer is doing well.

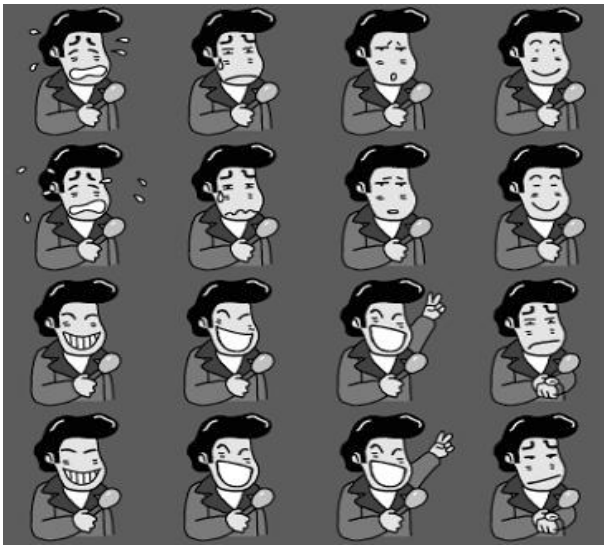

Figure 3. Animated-scoring indicator of the user's performance.

5. Conclusions

The running system has been tested with several songs and many users. Even though there are still many improvements to be made, both in the algorithms and in the implementation, the application has proved to be musically useful.

We have measured the hardware latency (with a SB AWE64 PCI), recording a stereo signal where one channel was the original sound, and the other channel was the sound through adc-dac read/write with small buffers (128 samples, 22100 kHz, 16 bit, mono). The difference was 6 ms, with constant throughput. To this, we have to add the latency that is inherent to the FFT's and pitch detection, which around 20 ms. Thus the delay between the sound input and the final sound output in the running system is less than 30 milliseconds. This delay is just good enough to make the user have the feeling of being in control of the output sound.

Some underrun problems occur by unpredictable performance problems, when the analysis or synthesis takes longer than expected. This might be because of memory allocation. Since the occurrence of these kinds of underruns is very rare, it is not a real problem for the current set up.

Currently, the scoring is only related to the pitch accuracy. We experienced that the timing accuracy is more difficult to translate to a significant scoring, at least to a continuously changing one. It is in some extend visualized with an extra image with the animation character looking at his watch.

Acknowledgments

We would like to acknowledge the support from Yamaha Corporation and the contribution to this research of the other members of the Music Technology Group of the Audiovisual Institute.

References

Cano, P., A. Loscos. 1999. *Singing Voice Morphing System based on SMS*, *UPC,* 1999

Cano, P., A. Loscos, J. Bonada, M. de Boer, X. Serra. 2000. "Singing Voice Impersonator Application for PC", *Proceedings of the ICMC* 2000.

Loscos, A., P. Cano, J. Bonada. 1999. "Low-Delay Singing Voice Alignment to text". *Proceedings of the ICMC* 1999.

Serra, X. 1997. "Musical Sound Modeling with Sinusoids plus Noise". G. D. Poli and others (eds.), *Musical Signal Processing*, Swets & Zeitlinger Publishers, 1997.