Regular Paper

# A Real-time Audio-to-audio Karaoke Generation System for Monaural Recordings Based on Singing Voice Suppression and Key Conversion Techniques

Hideyuki Tachibana[1,†1,a)] Yu Mizuno[1,†2] Nobutaka Ono[2,b)] Shigeki Sagayama[3,c)]

Received: July 31, 2015, Accepted: February 8, 2016

**Abstract:** This paper describes an automatic karaoke generation system, which can suppress the singing voice in audio music signals, and can also change the pitch of the song. Furthermore, this system accepts the streaming input, and it works in real-time. To the best of our knowledge, there have been no real-time audio-to-audio karaoke system that has the two functions above. This paper particularly describes the two technical components, as well as some comments on the implementation. In this system, the authors employed two signal processing techniques: singing voice suppression that is based on two-stage HPSS, a vocal enhancement technique that the authors proposed previously, and a pitch shift technique that is based on the spectrogram stretch and phase vocoder. The attached video file shows that the system works in real-time, and the sound quality may be practically acceptable.

**Keywords:** karaoke, music signal processing, singing voice, music application

## 1. Introduction

Karaoke is said to be invented in 1971 [56], and it is regarded as one of the earliest examples of electric-technology-based music applications for amateur music fans. It is currently one of the major ways of enjoying music, and is considered as one of major leisure activities. For example, karaoke was the 7th most populous leisure in Japan in 2011, in terms of the number of participants (38.4 million) [21]. (Note: the population of Japan was about 128 million in 2010.) This digit is comparable to those of other major leisure activities including watching movies, playing video games, and *listening* to music.

On the other hand, the songs created by amateur musicians, which are typically distributed through web sites, have become considerably popular recently, especially in the last decade. Recent growth of the web-based music community prompts many amateur musicians to upload their songs, and many listeners enjoy these songs as well as the songs created by professional musicians. (Relevant discussion on the web-based community of amateur musicians is found in Refs. [14], [15], etc.) However, it is not necessarily economic to create the karaoke data for all the songs of these kinds, remembering that the current karaoke systems require pre-made MIDI data, which are created manually by skilled craftspeople who have the ability of music dictation.

Thus the techniques that can generate karaoke signals automatically from mixed music signals, such as MP3 data, has significance. In order to create karaoke data from mixed music signals, we can consider principally two approaches as follows,

**(a) Audio → MIDI → MIDI → Audio** A possible approach may be the sequential processing as follows.
- ( 1 ) Audio to MIDI conversion, which may be based on general music transcription techniques such as multiple $F_0$ analysis (e.g., Ref. [22]).
- ( 2 ) Estimate which parts of the MIDI data are melody, and delete them from the MIDI data.
- ( 3 ) MIDI → Audio synthesis, which is a traditional technology of computer music.

However, this approach is not easy for now, because there are still many difficulties to solve the subproblems (1) and (2).

**(b) Audio → Audio** Another approach may be the direct audio-to-audio conversion, using some signal processing techniques. See also Section 1.1.1. This paper considers this approach.

On the basis of the motivations above, the authors developed an interactive music player named "Euterpe," which generates quasi-karaoke signals from ordinary music signals. The system does not require separately-recorded tracks of a song, but it only requires the already-mixed music audio signals (such as ordinary CDs, MP3 data, etc.) The system enables users to reduce the vocal part and change the key of accompanying sounds. Furthermore, the system works in real-time. Moreover, we specifically assume that the input signal is monaural because of the reasons described in Section 1.1.1. To the best of our knowledge, there have been no real-time automatic karaoke systems that integrated these two important features above: singing voice suppression

[1] The University of Tokyo
[2] National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[3] School of Interdisciplinary Mathematical Sciences, Meiji University, Nakano, Tokyo 164–8525, Japan
[†1] Presently with PKSHA Technology Inc.
[†2] Presently with Aichi Prefectural Government
[a)] tachi-hi@14.alumni.u-tokyo.ac.jp
[b)] onono@nii.ac.jp
[c)] sagayama@meiji.ac.jp

and key conversion.

In this paper, we describe the details of the processings and the architecture of the system. Note that the system is basically a combination of already-known techniques, and hence, we do not claim the novelty of each technical component. However, we would say that this paper has the following contributions: (1) we showed that our previous study, viz. singing voice enhancer [52], [53], is also promising as a singing voice suppressor, and (2) we developed the system that really works in real-time, by choosing lightweight technical components. Note also that this paper is a revision of some parts of the first author's Ph.D. Thesis [48], second author's Master's Thesis [30], as well as our short domestic non-refereed conference papers written in Japanese [29], [49], [51].

## 1.1 Related Work

### 1.1.1 Vocal Suppression

There is a widely-known simple method to generate a karaoke signal from a mixed music signal; we may obtain a quasi-karaoke signal just by subtracting the right channel $right(t)$ from the left channel $left(t)$ of a stereo signal $(left(t), right(t))$,

$$karaoke(t) = left(t) - right(t). \quad (1)$$

Many free software of automatic karaoke generators e.g. CenterPanRemover and some plugins for Audacity [1], are based on this method or its variant. This approach is based on the common practice of the presentday professional music creation procedure that all the instruments including singing voice are separately recorded, and the vocal component is placed on the center, when all the parts are mixed down by recording engineers. In other words, both $left(t)$, $right(t)$ contain almost exactly equal vocal components in many cases of professional recordings, and the subtraction can cancel the vocal component as a consequence.

A drawback of this simplest method is that it can be applied only to stereo music signals. In addition, the creation procedure should be based on the professional convention. In other words, we may not expect the technique to work effectively for e.g. live recordings, and we cannot apply it to monaural signals evidently. In order to cover a wider range of recordings above, a technique that automatically removes singing voice based on its nature, *not* on the recording convention, is needed.

To date, the number of academic literature that principally focused on this task seems limited. However, some automatic audio-to-audio karaoke generation techniques were proposed, such as the techniques based on Bayesian model [37], an $F_0$ estimation [44], Deep Leaning [46], and an RPCA (low-rank + sparse) -based model [18].

In the music signal processing community, the opposite side of the same task, namely vocal *extraction* (*enhancement*), has been attracting more interest than karaoke generation, and there have been more studies: Vembu & Baumann [54], Li & Wang [28], Hsu & Jang's unvoiced singing voice enhancement [16], the techniques based on Nonnegative Matrix Factorization [41], [55], [58], Generative model [42], robust PCA (RPCA) [17], Sprechmann's method that works in real-time [47], the authors' previous work called two-stage HPSS [52], [53], and the work by FitzGerald et al. [6], which is based on the similar idea to two-stage HPSS.

Of these, "two-stage HPSS" [52], [53] has an advantage that it performs well in terms of GNSDR (Generalized Normalized Signal to Distortion Ratio; see Ref. [38]), it is efficient, and it works in real-time with a little latency. This paper shows that the technique is effective also as a vocal *suppression* technique, and uses it as a technical component of our automatic audio-to-audio karaoke system [*1].

### 1.1.2 Another Common Function of Karaoke: Pitch Shift (Transpose)

Although the term "*karaoke*" literally translates "vacant (*kara*) orchestra (*oke*)," this literal meaning does no longer fully describe the presentday karaoke systems. Indeed, current karaoke systems have many functions other than vocal-off, such as the key conversion, the tempo conversion, showing lyrics on the monitor, scoring the quality of the singer's performance, etc.

Of these, this paper specifically considers key conversion, i.e., conversion of a music signal from a key to another singable key (e.g., C Major $\xrightarrow{\text{decrease the pitch by 3 semitones}}$ A Major). This is not a trivial problem in case of audio-to-audio conversion, contrary to the case of MIDI-based karaoke system, in which it is quite easily done just by adding/subtracting an integer to/from the MIDI note numbers.

The audio-to-audio pitch conversion techniques, viz. pitch-shift (and time-stretch), includes *Pitch Synchronous OverLap and Add* (PSOLA) [3], [5]. The technique effectively works for single source audio signals, but there is a drawback that they are not basically suitable for multi-source signals, which music signals typically are. Another well-known technique which is capable of pitch shift is *phase vocoder* [7], [24]. Other recent pitch shift techniques for music signals include the works by Schörkhuber et al. [45] and Nakamura & Kameoka [31].

Of these, the *phase vocoder* has been widely used in audio processing, probably because it is efficient, intuitive, and is compatible with the processing on spectrogram, discussed in Section 2. In this paper, we shall use a variant of phase vocoder as a pitch converter, though we additionally consider the issues of timbre; the "color" of the sound.

In an ordinary phase vocoder, as suggested by Röbel and Rodet [43], the timbre, i.e., spectral envelope, changes in the conversion procedure. As a consequence, the timbre is prone to sound differently from that of the original sounds, especially when the conversion interval is large. In other words, the resultant "transposed accompaniment" may not necessarily sound like the same instrument. Röbel and Rodet addressed the problem, and developed a new method to estimate the spectral envelope before pitch shifting to keep the spectral envelope. In their paper, it is also shown that their technique can more precisely estimate the spectral envelope than other typical techniques such as LPC and the cepstrum,

In this paper, however, we consider the simpler approach,

[*1] Major English articles on two-stage HPSS [52], [53] principally focused on singing voice *enhancement*, but the idea to exploit the technique to singing voice *suppression* has already been mentioned in the first publication of the technique (conference short paper) [51].

namely LPC-based spectral envelope extraction, since we also place emphasis on efficiency in this paper, as discussed in Section 3.

Thus our technique is based on the following two techniques. This configuration has also been considered in our previous conference paper [29].

**(1) Spectrogram Stretch** Expand/shrink the given power spectrogram in the direction of frequency axis, keeping the spectral envelope fixed.

**(2) Waveform Synthesis from Amplitude Spectrogram:** Obtain the waveform from the modified amplitude spectrogram using RTISI-LA [59], a variant of phase vocoder.

#### 1.1.3 Interactive Music Players

The concept of audio-to-audio conversion of a music signal is relevant to some of the interactive music applications based on music signal processing [23]. The proposal system may also be related to the concept "*Active Music Listening*" [12], which enables users to enjoy music interactively and actively, instead of just listening to it. Many systems, related to the concept, have been developed recently, including *Drumix* [57], *MusicKIOSK* [10], etc.

### 1.2 Paper Organization

The rest of this paper is organized as follows. Section 2 describes the fundamentals of signal processing techniques on spectrogram. Busy readers may skip this section but Section 2.2.2 may be useful to understand Section 5. Section 3 describes the requirements of the karaoke systems that we discuss in this paper. In Section 4 and Section 5, we describe the signal processing techniques which we utilized in the system. In Section 6, we discuss some issues on implementation. Finally Section 7 concludes the paper.

## 2. Signal Processing on Spectrogram

Before discussing the specific techniques of singing voice suppression and pitch shift, let us briefly recall the fundamentals of music signal processing.

### 2.1 Short-time Fourier Transform and Spectrogram

#### 2.1.1 Complex Spectrogram

In music signal processing, it is common to formulate techniques on the time-frequency plane, which is called '*spectrogram*.' (See e.g., Ref. [23].) Given a discrete signal $x(t) \in \mathbb{R}^{f_{\mathrm{samp}} T_{\mathrm{total}}}$, by applying the Short-time Fourier Transform (STFT) to it, a *complex spectrogram*. $\tilde{\boldsymbol{X}} = (\tilde{X}_{n,k})_{1 \le n \le N, 1 \le k \le K} \in \mathbb{C}^{N \times K}$, is obtained, where $\mathsf{STFT} : \mathbb{R}^{f_{\mathrm{samp}} T_{\mathrm{total}}} \to \mathbb{C}^{N \times K}$ is defined by $\tilde{X}_{n,k} = \mathsf{STFT}_L[x(t)] = \mathsf{FFT}_L[x(nS - L/2 + t)w(t)]$, where $\mathsf{FFT} : \mathbb{R}^L \to \mathbb{C}^K$ is the Fast Fourier Transform, which maps a signal $u(t)$ to a spectrum $\hat{u}(k)$. The definitions of the other symbols are shown in **Table 1**.

#### 2.1.2 Amplitude Spectrogram

The absolute value of a complex spectrogram, $\boldsymbol{X} = (X_{n,k})_{n,k} = (|\tilde{X}_{n,k}|) \in {\mathbb{R}_{\ge 0}}^{N \times K}$, is called an amplitude spectrogram. **Figure 1** shows an example of the amplitude spectrogram of a music signal.

Many audio signal processing techniques are indeed a conversion from an amplitude spectrogram $\boldsymbol{X} \in {\mathbb{R}_{\ge 0}}^{N \times K}$ into another $\boldsymbol{Y} \in {\mathbb{R}_{\ge 0}}^{N \times K}$. That is, $\boldsymbol{Y} \leftarrow f(\boldsymbol{X})$ where $f$ is some processing on the spectrogram. A famous "$f$" is *time-frequency masking*, which is not used in this paper, but is sometimes regarded as common sense in audio signal processing literature.

**Table 1** List of the symbols used in the definition of STFT.

| Symbol | Meaning |
|---|---|
| $L \in 2\mathbb{N}$ (even number) | frame length |
| $S \in \mathbb{N}$ | frame hop size (frame shift) |
| $f_{\mathrm{samp}}$ [Hz] | sampling rate |
| $T_{\mathrm{total}}$ [s] | length of the song |
| $N := \lceil f_{\mathrm{samp}} T_{\mathrm{total}} / S \rceil$ | number of time frames |
| $K := L/2 + 1$ | number of frequency bins |
| $n \in \mathbb{Z}, 1 \le n \le N$ | time index |
| $k \in \mathbb{Z}, 1 \le k \le K$ | frequency index |
| $w(t)$ | window function |

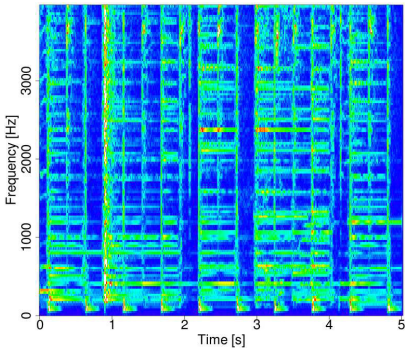



**Fig. 1** An example of the amplitude spectrogram. The music signal is excerpted from RWC database [11]. Blue bins are silent, while green, yellow, and red are louder. Higher frequencies are enhanced for visibility. (This figure is extracted from the author's previous publication [50].)

### 2.2 Inverse STFT (Wave Synthesis from a Spectrogram)

#### 2.2.1 Complex Spectrogram → Waveform

After the processing on the spectrogram, we need to transform the spectrogram back to the waveform.

A transform from a complex spectrogram to a waveform is rather trivial; we can synthesize the waveform $x(t) \in \mathbb{R}^{f_{\mathrm{samp}} T}$ by applying an inverse STFT technique to the complex spectrogram $\tilde{\boldsymbol{X}} \in \mathbb{C}^{N \times K}$.

A simple inverse STFT technique is based on a tandem connection of the inverse FFT and a simple wave concatenation which is called *overlap-add*. For the details of the inverse STFT algorithm, see e.g., Ref. [23].

#### 2.2.2 Amplitude Spectrogram → Waveform

Contrary to the case of complex spectrogram, the transformation from an amplitude spectrogram to a waveform is an ill-posed problem. Since we do not have any information on the phase $\psi_{n,k} \in [0, 2\pi)$ where $\tilde{Y}_{n,k} = Y_{n,k} e^{\sqrt{-1}\psi_{n,k}}$, we need to estimate the phase spectrogram $(\psi_{n,k})$ to apply the inverse STFT mentioned.

There are typically two methods to obtain a phase spectrogram as follows.

**(a) Diversion of the Original** Store the phase spectrogram $(\phi_{n,k})$ of the input complex spectrogram $\tilde{\boldsymbol{X}}$, where $\tilde{X}_{n,k} = |\tilde{X}_{n,k}| e^{\sqrt{-1}\phi_{n,k}}$. Then, we use it in inverse STFT assuming that $\psi_{n,k} \approx \phi_{n,k}$. That is, let $\psi_{n,k} = \phi_{n,k}$, and apply the inverse STFT, $y(t) = \mathsf{STFT}_L^{-1}[Y_{n,k} e^{\sqrt{-1}\psi_{n,k}}]$.

**Table 2** Four candidates of how a karaoke generator should be implemented; {Server-side, Client-side} × {Download, Streaming}, and whether each approach requires real-time processing or not.

| | **Server-side Vocal Suppression**<br>Technically easier, but requires a server and engineers who maintain the system. | **Client-side Vocal Suppression**<br>Not dependent on any specific service providers |
|---|---|---|
| **Download → Play**<br>Technically easier | Download a pre-made karaoke data and play it.<br>(Real-time processing not necessarily required) | Download whole the music file and process it on users' machines.<br>(Real-time processing not necessarily required) |
| **Streaming Play**<br>Rather user-friendly | Play fragments of a pre-made karaoke data one after another.<br>(Real-time processing not necessarily required) | Process the streaming input on users' machines and play it.<br>*Real-time (On-line) processing required* |

**(b) Phase Recovery Technique** Apply a phase recovery technique (phase vocoder) to the amplitude spectrogram $\boldsymbol{Y}$ to estimate a phase spectrogram ($\psi_{n,k}$). See Section 5 as well as Ref. [13].

We exploit both methods in this paper. The first approach in Section 4, and the second technique in Section 5, since we cannot expect $\psi_{n,k} \approx \phi_{n,k}$ in Section 5.

## 3. Requirements for the Karaoke System

### 3.1 Vocal Off and Key Conversion

As discussed in the introduction, karaoke is not merely a "vocal-off," but it has many other functions including key/tempo conversion, showing lyrics, rating the singing ability, etc. Of these, we focus on the key conversion, and vocal suppression of course, in this paper [*2]. That is, the system requires the following properties.

I *Vocal suppression from mixed audio* (Section 4).

II *Key conversion* (Section 5).

### 3.2 Client-side Real-time Processing

We additionally require the system to satisfy the following property.

III *Accept Streaming Input and Work in Real-time.*

Let us consider the following four possible cases: {*Download*, *Streaming*} × {*Server-side*, *Client-side*}. **Table 2** shows the details of each candidate. Although the 4 candidates in the table have their own advantages and disadvantages, the candidate "*Streaming+Client*" has following two remarkable advantages;

**Streaming:** Streaming is user friendly. A report by CNN [19] says that downloading has become less common for young listeners in USA but they prefer streaming, and that some people think that streaming is more "liberating."

**Client-side:** Client-side processing has an advantage for users that it is basically independent of the specific services. Thus users may use the method on any music streaming services in principle. In addition, client-side signal processing does not require any central server which should be maintained by some organization over a long time.

As described in Table 2, the three cases namely "*Download+Server*", "*Streaming+Server*" and "*Download+Client*", actually do not require streaming-based vocal-suppression techniques that work in real time. Nonetheless, when it comes to the case "*Streaming+Client*", real-time processing is essential.

[*2] We do not discuss the time stretch (tempo conversion) in this paper because it is not necessarily possible in streaming processing. However, it may be easily achieved in the same framework when we consider the file-input instead of streaming. Indeed, pitch shift has been discussed in tandem with time stretch, because both are essentially the same.

## 4. Two-stage HPSS for Singing Voice Suppression

This section describes singing voice suppression techniques for audio signals. The technique should be based on the nature of the singing voice, not on the recording convention, in order to cover a wider range of recordings including monaural signals.

In this paper, we exploited Two-stage HPSS [52], [53], which has two advantages: it is an effective singing voice enhancement algorithm, and it works in real-time (on-line).

### 4.1 Harmonic/Percussive Sound Separation (HPSS)

Before describing two-stage HPSS, let us first discuss HPSS [34], [35], [36], [50] briefly. HPSS is a technique that separates a spectrogram $\boldsymbol{Y}$ into two components below,

- horizontally continuous (viz. harmonic) components $\boldsymbol{H}$
- vertically continuous (viz. percussive) components $\boldsymbol{P}$

HPSS is formulated as an optimization problem, based on the three objectives below,

( 1 ) temporal difference of $\boldsymbol{H}$ should be small

( 2 ) frequency difference of $\boldsymbol{P}$ should be small

( 3 ) $\boldsymbol{H} + \boldsymbol{P} \approx \boldsymbol{Y}$

We have considered several instantiation of these three concepts. In Ref. [35], the loss function was defined as an MRF (Markov Random Field) -like model, which consists of local penalties between adjacent bins, as follows,

$$J(\boldsymbol{H}, \boldsymbol{P}; \boldsymbol{Y}) = \sum_{n,k} (H_{n,k}{}^{\gamma} - H_{n-1,k}{}^{\gamma})^2 + w \sum_{n,k} (P_{n,k}{}^{\gamma} - P_{n,k-1}{}^{\gamma})^2 + c\mathsf{KL}(\boldsymbol{Y}^{2\gamma} | \boldsymbol{H}^{2\gamma} + \boldsymbol{P}^{2\gamma}) \qquad (2)$$

where $\mathsf{KL}(\boldsymbol{x}|\boldsymbol{y}) = \sum_{n,k}(x_{n,k}\log(x_{n,k}/y_{n,k}) - (x_{n,k} - y_{n,k}))$ the generalized Kullback Leibler divergence, $\gamma$ a certain exponential factor, and $w, c$ are weight constants.

We can decrease this objective function by the coordinate descent, which is a simple but practical approach to the problems that have very large number of variables ($H_{n,k}$ and $P_{n,k}$ in our case). Instead of the ordinary gradient descent approach, our update rule is based on the MM (majorization-minimization) algorithm. The iterative update is written in the following convolutive form using certain light functions $f_H, f_P, f_\theta$,

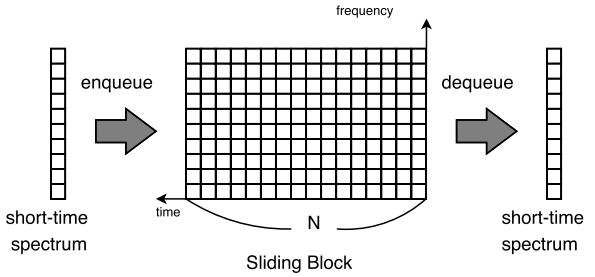



**Fig. 2** Concept of the sliding block analysis. See also Ref. [35].

$$H_{n,k} \leftarrow f_H(H_{n\pm1,k}, P_{n,k\pm1}, \theta_{n,k}) \tag{3}$$

$$P_{n,k} \leftarrow f_P(H_{n\pm1,k}, P_{n,k\pm1}, \theta_{n,k}) \tag{4}$$

$$\theta_{n,k} \leftarrow f_\theta(H_{n,k}, P_{n,k}) \tag{5}$$

where $\theta_{n,k}$ are auxiliary parameters. For the specific forms, see Appendix A.1. Applying these formulae for each coordinate orderly (known as CCD: Cyclic Coordinate Descent), we can find a local optimum of the objective function.

If we finally obtain spectrograms $\boldsymbol{H}, \boldsymbol{P}$, then we apply the inverse STFT to $\boldsymbol{H}$ and $\boldsymbol{P}$, using the original phase spectrogram (see Section 2.2.2 (a)), and obtain the wave forms $h(t)$ and $p(t)$.

### 4.2 Streaming HPSS

We actually do not need the spectrogram of the entire song to execute HPSS, but we only need short period of it to obtain a practical solution at a bin $(n, k)$, because of the reasons below:

- The iterative updating formulae Eqs. (3), (4) and (5) are quite localized. That is, we only require the adjacent bins $(n \pm 1, k), (n, k \pm 1)$ as well as itself $(n, k)$ to update the components at $(n, k)$, namely $H_{n,k}, P_{n,k}$, and $\theta_{n,k}$.
- Intuitively, little of the effects from far distant bins pass through the 'barriers' of the 'silent' bins $\{(n', k')\}$ s.t. $H_{n',k'} \approx P_{n',k'} \approx 0$, which are constrained to be silent because of the three constraints: (1) $Y_{n',k'} \approx 0$ (there are many silent bins in the input spectrogram; see Fig. 1), (2) $H_{n',k'} + P_{n',k'} \approx Y_{n',k'}$ and (3) $H_{n',k'}, P_{n',k'} \geq 0$.

Let us extract the short period from the entire song, and call this window as a "sliding block," [35], which is shown in **Fig. 2**. In the procedure below which exploits a sliding block, the updating formulae Eqs. (3), (4), and (5) are applied $N \times I$ times for each component of the spectrogram. In this setting the latency is evaluated as approximately $NS/f_{\mathsf{samp}}$ [s], where $S/f_{\mathsf{samp}}$ [s] is the frame shift of STFT.

**(1) Enqueue/dequeue:** Enqueue/dequeue the short-time spectrum to/from the sliding block of length $N$.

**(2) Update:** Apply updating formula Eqs. (3), (4), and (5) to the sliding block $I$ times ($I \in \mathbb{N}$).

The above procedure is basically similar to RTISI-LA, a technical component of key conversion, which is described in Section 5.

For those who are interested in the convergence of streaming HPSS, see Appendix.

### 4.3 Vocal Off Technique: Two-stage HPSS

Singing voice may be regarded as a stationary signal in a very short period (such as 1 [ms]), while it may not be so in a very long period (such as 1 [s]), owing to its fluctuation. It is a matter of the relative time-scale whether singing voice appears as 'rather harmonic signal' or 'rather percussive (non-stationary) signal.' Exploiting this nature of singing voice we can roughly extract it from a music signal just by twice applying the local convolutive operation, namely HPSS [52].

In an ordinary condition, HPSS separates a music signal into harmonic components and percussive components. That is,

$$(\textit{Input}) \xrightarrow{\mathsf{HPSS}(L_1)} (\textit{Harmonic including Vocal}), (\textit{Percussive})$$

where $L_1$ is a frame length of STFT. To the contrary, if we apply it to a spectrogram of long frame length $L_2 \gg L_1$, HPSS separates the same signal quite differently in the following way,

$$(\textit{Input}) \xrightarrow{\mathsf{HPSS}(L_2)} (\textit{Harmonic}), (\textit{Fluctuated + Percussive}).$$

Since singing voice often has fluctuation, vocal components tend to be separated into "percussive" component in this case [*3]. Therefore, applying HPSS twice on differently-resolved spectrograms, a signal can be separated to the three components: the *harmonic* (e.g., guitar), the *fluctuated* (e.g., vocal), and the *percussive*.

$$(\textit{Input}) \xrightarrow{\text{Two-stage HPSS}} (\textit{Harmonic}), (\textit{Vocal}), (\textit{Percussive}).$$

The purpose of our previous work [52] was to *extract the vocal*, but what we are interested in in this paper is the residuals, i.e., *harmonic* and *percussive*. In this paper, we generated the "vocal-off" signal just by adding thus obtained *harmonic* and *percussive* multiplying certain weights $\alpha_h, \alpha_p \in \mathbb{R}$. (See Fig. 6). **Figure 3** shows an example of the result of two-stage HPSS.

An advantage of this approach is that we do not need any explicit models, nor training data, of singing voice, which may be complicated or may be infeasible for real-time processing. In two-stage HPSS, we just need two operations: FFT and the convolutive operation (namely HPSS), which are sufficiently efficient. Moreover, it works in real-time (on-line) since the streaming HPSS is basically a one-pass technique.

### 4.4 Performance of Singing Voice Suppression

In Ref. [52], it was shown that the two-stage HPSS extracts singing voice from a mixed music signal effectively; the performance was about 4 dB GNSDR (weighted average of SDR improvement) in some conditions. In this section, let us similarly evaluate the performance of two-stage HPSS as a singing voice *suppressor* using SDR, and our ears.

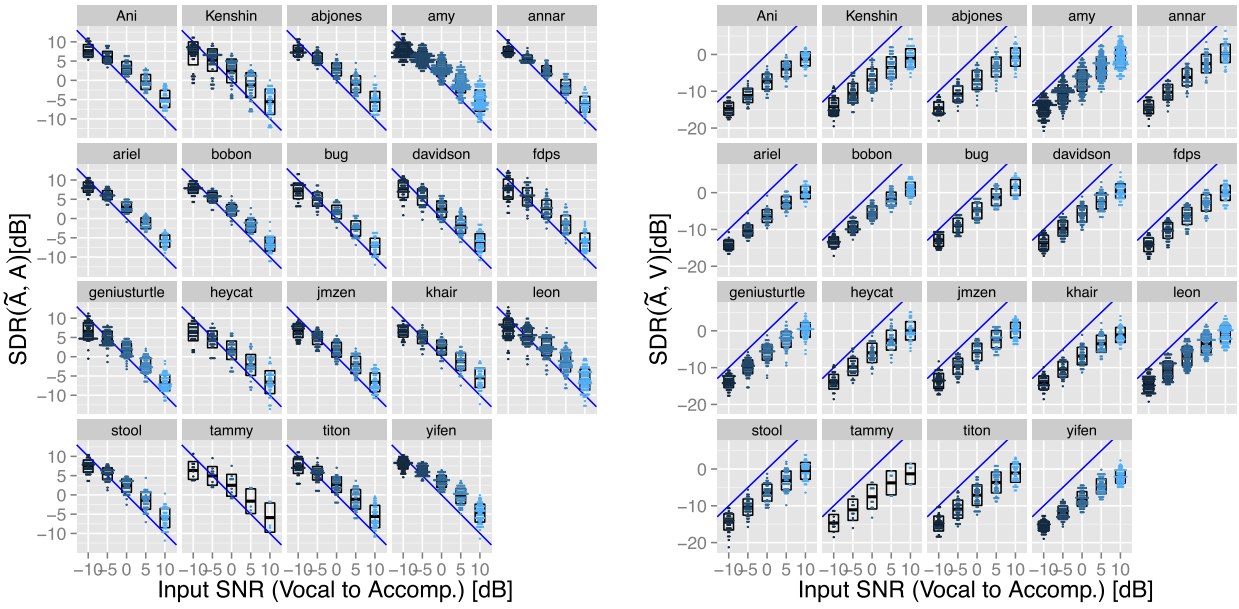
#### 4.4.1 Evaluation Criterion: SDR (Signal to Distortion Ratio)

Given an estimated signal $E(t)$ and the ground truth $X(t)$, SDR is defined as follows,

$$\mathsf{SDR}(E(t); X(t)) := 10 \log_{10} \frac{\|aX(t)\|^2}{\|E(t) - aX(t)\|^2} \tag{6}$$

$$= 10 \log_{10} \frac{\langle X(t), E(t)\rangle^2}{\|X(t)\|^2 \|E(t)\|^2 - \langle X(t), E(t)\rangle^2} \tag{7}$$

[*3] Of course, *vocal* ⇔ *fluctuated signal*. For example, (1) singing voice does not always have sufficient fluctuation, and (2) there are many instruments that has fluctuation such as violin. However, we roughly identified them because this assumption practically works.




(a) (b)

**Fig. 4** (a) Histogram of $\mathrm{SDR}(\tilde{A}; A)$ of the 5000 music clips (1,000 clips × 5 input SNRs). Each dot indicates a music clip. The averaged value for each input SNR is 7.4 dB (−10 dB input SNR), 5.5 dB (−5 dB), 2.5 dB (0 dB), −1.4 dB (5 dB) and −5.9 dB (10 dB), respectively. (b) Histogram of $\mathrm{SDR}(\tilde{A}; V)$, similarly. Blue lines indicate the baseline (the levels of the input signals), and the black boxes indicate the mean ± sd. The averaged value for each input SNR is −14.2 dB (−10 dB),−10.4 dB (-5 dB), −6.6 dB (0 dB), −3.1 dB (5 dB) and −0.4 dB (10 dB), respectively. Ani, Kenshin, . . . are the IDs of the singers.

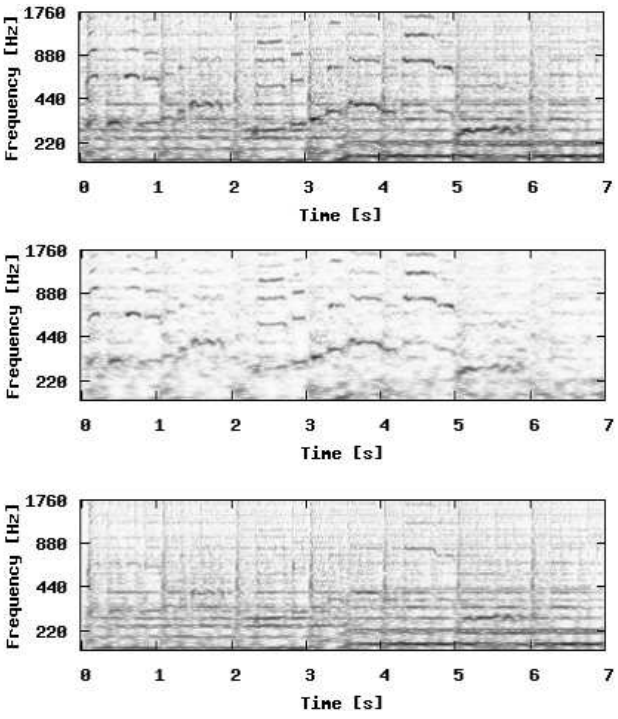



**Fig. 3** An example of the result of two-stage HPSS. Top figure shows the spectrogram of a mixed music signal. Middle figure shows the spectrogram of extracted singing voice by two-stage HPSS. Bottom figure shows the spectrogram of the residual component, i.e., the sum of the 'harmonic' and 'percussive,' which is roughly equal to the accompaniment.

where $\langle \cdot, \cdot \rangle$ denotes the inner product, $\|x\|^2 := \langle x, x \rangle$, and $a \in \mathbb{R}$ is a certain constant that satisfies $\langle X(t), E(t) - aX(t) \rangle = 0$. Intuitively, $\mathrm{SDR}(E(t); X(t))$ indicates the strength of the target signal $X(t)$ in the estimated $E(t)$. Evidently, "SDR = $r$ dB" means that the target signal $X(t)$ is $10^{r/10}$ times louder than the distortion $E(t) - aX(t)$.

**Table 3** Parameters of the two-stage HPSS.

| | Parameter | Value |
|---|---|---|
| | Sampling Rate $f_{\mathrm{samp}}$ | 16,000 [Hz] |
| Short Frame | Frame length $L_1$ | 256 (=16 [ms]) |
| | Frame hop size $S_1$ | 128 (=8 [ms]) |
| | Parameter $w$ | 1 |
| | Parameter $c$ | 0.2 |
| | Parameter $\gamma$ | 1 |
| | Sliding block size $N$ | 30 |
| Long Frame | Frame length $L_2$ | 4,096 (=256 [ms]) |
| | Frame hop size $S_2$ | 2,048 (=128 [ms]) |
| | Parameter $w$ | 1 |
| | Parameter $c$ | 0.2 |
| | Parameter $\gamma$ | 1 |
| | Sliding block size $N$ | 30 |

Let $V(t), A(t)$ be the ground truth vocal and accompaniment, and $\tilde{V}(t), \tilde{A}(t)$ are the estimated, respectively. What we want to evaluate here is $\mathrm{SDR}(\tilde{A}(t); A(t))$, i.e., the strength of the ground truth in the obtained "accompaniment." [*4]

### 4.4.2 Experiment & Result

Let us evaluate SDRs using real music signals. The parameters we used in this experiment are shown in **Table 3**. We evaluated the SDR of the accompaniment using the MIR-1K dataset, similar to Ref. [52].

[*4] Since singing voice *enhancement* and *suppression* are "two sides of the same coin," the result, i.e. $\mathrm{SDR}(\tilde{A}(t); A(t))$ may be basically similar to the value of $\mathrm{SDR}(\tilde{V}(t); V(t))$ in Ref. [52]. Indeed, it is easily verified

$$\mathrm{SDR}(\tilde{A}(t); A(t)) = \mathrm{SDR}(\tilde{V}(t); V(t)) - \text{input SNR}$$

where input SNR is $\mathrm{SDR}(V + A; V)$, if we can assume $\langle V(t), A(t) \rangle = \langle V(t), \epsilon(t) \rangle = \langle A(t), \epsilon(t) \rangle = 0$, and $\tilde{V}(t) = V(t) + \epsilon(t), \tilde{A}(t) = A(t) - \epsilon(t)$ where $\epsilon(t)$ is the distortion. Nevertheless, it is not necessarily the case.

**Figure 4** (a) shows the distribution of $\mathsf{SDR}(\tilde{A}(t); A(t)))$, i.e., the strength of the accompaniment in the output signal. The histogram is plotted by the singers. The values of SDR were higher than the levels of the input signals on average if input SNR is 0, 5, and 10 dB. This suggests that the accompaniment $A(t)$ became relatively louder in the output $\tilde{A}(t)$, compared with the input signal $Input(t) = A(t) + V(t)$, in these conditions. This figure shows that the trend does not strongly depend on the individual singer. On the other hand, if input SNR is $-5$ and $-10$ dB (i.e., the cases that the accompaniment is louder than the vocal), the output quality was lower than the baseline on average. This is possibly because the deterioration of the accompaniment has more impact on the digits than the vocal reduction does in the low input SNR case.

Figure 4 (b) shows the distribution of $\mathsf{SDR}(\tilde{A}(t); V(t)))$, i.e., the strength of the singing voice in the output signal. This shows that the singing voice tends to reside quite little in the output signals on average. It is seen also in this figure that the trend does not strongly depend on the individual singer.

#### 4.4.3 Comparison to Other Methods

The digits, however, are not as good as the state-of-the-art techniques e.g., Ref. [18], according to the digits reported in the Figure in the literature [18]. Figure 5 in Ref. [18] shows that the performance (GNSDR) of RPCA [17] is about 1.5 dB, and that that of Ikemiya's method (RPCA-F0) is about 6 dB. The corresponding GNSDR value of our method is about 2.5 dB (averaged $\mathsf{SDR}(\tilde{A}; A)$ when input SNR = 0 dB). These digits indicate that the performance of our method is a little better than RPCA, while much worse than RPCA-F0. Nevertheless, our method has the advantage that it is suitable for real-time processing.

### 4.5 Audio Examples

In the attached video file, the examples of the output are shown. Listen to the attached Audio 1a, 1b (RWC-MDB-P-2001 No.7), 2a, 2b (No.13), 3a, 3b (No.98). The songs 1a, 2a, 3a were excerpted from RWC database [11], and converted to mono 16 kHz. The clips 1b, 2b, 3b are the result.

Listening to these examples, we can find that the output signals contain less vocal components than the input signals, as the experiment above suggests. Although the output signals contain some distortions, they may be largely acceptable as karaoke signals.

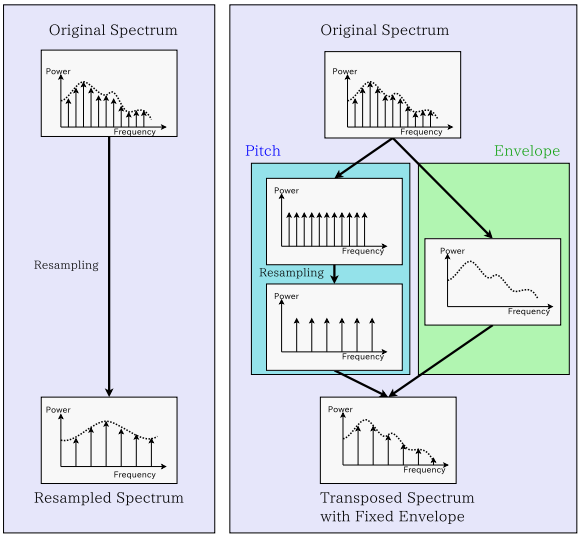
## 5. Key Conversion with Fixed Spectral Envelope

As discussed in Section 1 and Section 2, karaoke is not merely a vocal-off, but it has much more functions including key conversion. This section describes a simple technique to convert the *key* (e.g., "C Major," "D Major," etc) of a music signal.

### 5.1 Key Conversion Based on Resampling and Phase Vocoder

Before considering the timbre, let us consider the simpler method, shown in **Fig. 5** (a).

Pitch shift is achieved by the resampling of the waveform and the concatenation of the resampled waveform segments. This is




(a) Straightforward Resampling (b) Resampling with Fixed Envelope

**Fig. 5** Concept of pitch shift. (a) Straightforward "pitch shift" based on resampling, in which the shape of spectral envelope changed. (b) Fixed-envelope pitch modification. Harmonic series are expanded, but spectral envelope is kept almost fixed. (Note: in order to make the discussion intuitive, we displayed the signals on the *frequency* domain, but the actual processing was done on the *waveform* domain based on the LPC framework.)

intuitively similar to stretching the short-time spectrum in the direction of the frequency. This section describes the key conversion technique based on the resampling, as well as the wave concatenation technique.

#### 5.1.1 Resampling of a Short Wave Segment

The resampling from the short wave segment $s'(t)$ of length $L'$ to the another one $s(t)$ of length $L$ is done by the following procedure.

$$s'(t) \xrightarrow{\mathsf{FFT}_{L'}[\cdot]} \hat{s}'(\omega) \xrightarrow{\text{projection}} \hat{s}(\omega) \xrightarrow{\mathsf{FFT}_{L}^{-1}[\cdot]} s(t) \tag{8}$$

The ratio of $L'$ and $L$ gives the interval of the conversion,

$$L'/L \approx 2^{n/12}, \tag{9}$$

where $n$ [semitone] is the interval [*5].

The size of the output, i.e. $L$, should be a simple integer, preferably a power of 2, since we shall execute FFT of size $L$ many times in the subsequent processing (i.e., RTISI-LA). In contrast, FFT of size $L'$ is less frequently executed. Therefore, it is better to set $L$ a fixed simple integer such as 1024, 2048, and regulate the interval $n = 12 \log_2 L'/L$ via the value of $L'$.

Nevertheless, it is preferable that $L'$ is also an efficient number for FFT. For example, FFTW3 [9], a famous implementation of FFT, prefers that the size of FFT is a product of small primes smaller than 13, namely

$$L' = 2^{e_2} 3^{e_3} 5^{e_5} 7^{e_7} 11^{e_{11}} 13^{e_{13}} \text{ where } e_i \in Z_{\geq 0} \tag{10}$$

In addition, it also requires $e_{11} + e_{13} \leq 1$ for efficiency [8]. Thus we need to find the suitable $L'$ that approximate $2^{n/12}L$ in the integers above. For example, if $L' = 1440 (= 2^5 \times 3^2 \times 5)$ and

[*5] Note that we considered $-12 \leq n \leq 12$ in Table 4 but the range or practical $n$ is narrower. In our experiments, $-4 \lesssim n \lesssim 4$ seemed practical.

$L = 1024$, the ratio $L'/L = 45/32 \approx 2^{5.902/12}$ approximates $2^{6/12}$ finely. (Cf. $\lfloor 1024 \times 2^{6/12} \rfloor = 1448 = 2^3 \times 181$, and 181 is prime.) **Table 4** shows such approximate values we have found.

#### 5.1.2 Phase Vocoder (Concatenation of Inconsistent Wave Segments)

On the basis of the above procedure, we convert the input wave segment

$$s'_n(t), \quad c_n - L'/2 \le t < c_n + L'/2 \tag{11}$$

to the output wave segment, whose key is $(12 \log_2 L'/L)$ semitones higher,

$$s_n(t), \quad c_n - L/2 \le t < c_n + L/2 \tag{12}$$

where $c_n$ is the center of each frame, i.e., $c_n = nS$ where $S$ is the frame shift of STFT.

The wave segments obtained above, however, have a problem that the neighboring wave segments $s_n(t)$ and $s_{n+1}(t)$ are inconsistent. Then, in order to connect these wave segments, we need to employ a technique to reconstruct the waveform from a modified amplitude spectrogram, as discussed in Section 2.2.2.

We then used the phase vocoder here, which is efficient and intuitive. A basic method is the one proposed by Griffin & Lim [13], which is a simple iteration of the STFT and inverse STFT. Some variant of the technique are known such as the work by Le Roux et al. [25], [26], [27], Perruadin et al. [39], but in this paper, we used a method proposed by Zhu et al. [59], which is called *Real-Time Iterative Spectrum Inversion with Look Ahead* (*RTISI-LA*). As the name suggests, the method works in real time. RTISI-LA is based on the similar architecture to the streaming HPSS discussed above, i.e., it is basically the iteration of the STFT and inverse STFT on the "sliding block."

### 5.2 Spectrogram Modification with Fixed Spectral Envelope

A drawback of the method discussed in the previous subsection is that the timbre, i.e. spectral envelope, of the spectrum also changes, as claimed in Ref. [29], [43]. In order to avoid this problem, we first decomposes a spectrum into the envelope and the pitch by LPC analysis [20], [40] to obtain the two data. Then the system applies the pitch shifting technique based on the resampling described above, while the spectral envelope is kept fixed, as described in Fig. 5 (b).

**Table 4** Approximate values of $2^{n/12}$ which makes $L' = 2^{n/12}L$ a simple integer written as a product of small primes when $L$ is power of 2 e.g., 1024, 2048.

| positive $n$ | | negative $n$ | |
|---|---|---|---|
| $2^{1/12}$ | $\approx 135/128$ | $2^{-1/12}$ | $\approx 15/16$ |
| $2^{2/12}$ | $\approx 9/8$ | $2^{-2/12}$ | $\approx 225/256$ |
| $2^{3/12}$ | $\approx 1215/1024$ | $2^{-3/12}$ | $\approx 27/32$ |
| $2^{4/12}$ | $\approx 5/4$ | $2^{-4/12}$ | $\approx 405/512$ |
| $2^{5/12}$ | $\approx 675/512$ | $2^{-5/12}$ | $\approx 3/4$ |
| $2^{6/12}$ | $\approx 45/32$ | $2^{-6/12}$ | $\approx 45/64$ |
| $2^{7/12}$ | $\approx 3/2$ | $2^{-7/12}$ | $\approx 675/1024$ |
| $2^{8/12}$ | $\approx 405/256$ | $2^{-8/12}$ | $\approx 5/8$ |
| $2^{9/12}$ | $\approx 27/16$ | $2^{-9/12}$ | $\approx 75/128$ |
| $2^{10/12}$ | $\approx 225/128$ | $2^{-10/12}$ | $\approx 9/16$ |
| $2^{11/12}$ | $\approx 15/8$ | $2^{-11/12}$ | $\approx 135/256$ |

## 6. Does the System Really Work in Real-time? — Implementation of the System

### 6.1 Overview of the Implementation

In this section we describe the implementation of the system. The source code of the system is available on the author's GitHub [*6]. Note we implemented the system to receive the input from the *audio input jack*, instead of the *streaming input* via the web, but they are essentially similar. Almost all of the system is coded in C++ except the GUI which is written in Tcl/Tk.

The system consists of the cascade connection of five processing blocks, shown in **Fig. 6**. The total architecture is based on the pipeline model as shown in the figure.

**Figure 7** shows the user interface of the system. The system has two sliders, "key" and "volume." By dragging the "key" slider, the users can change the value of $n$ in the pitch shift algorithm, while the "volume" slider is linked to the values of $\alpha_h, \alpha_p, \alpha_v$ in Fig. 6.

**Figure 8** shows a photo of the experiment system. We connected the output jack of a CD player (SONY CD Walkman ®) to the audio input jack of PC.

**Table 5** shows the parameters of each technical component. Note that the frame lengths $L_1, L_2$ of HPSS are different from those used in Section 3. The parameters in Section 3 are advantageous in audio quality, but when we used them in real-time processing, the system tended to be less stable. Quantitatively, the parameter set shown in Table 5 ($I = 1$) gives poorer performance: GNSDR (averaged $\mathsf{SDR}(\tilde{A}; A)$ ) was about 1.1 dB when

**Audio Input**
Input audio signals from streaming audio input (or file).

**Two-stage HPSS (§4)**
Similar to [52].
- HPSS 1: Original signal $\xrightarrow{\text{HPSS(Short)}} h'(t), p(t)$
- HPSS 2: $h'(t) \xrightarrow{\text{HPSS(Long)}} h(t), v(t)$

**Synchronization and Add**
Synchronize the output of Two-stage HPSS $h(t), v(t), p(t)$, and add them. i.e., $x(t) = \alpha_h h(t) + \alpha_v v(t) + \alpha_p p(t)$, where $\alpha_h, \alpha_v, \alpha_p$ are certain coefficients, controlled by the users through the user interface.

**conversion (§5)**
Apply key conversion and wave synthesis.

**Audio Output**
Output the signal to the audio device. The real-time audio input/output was achieved by PortAudio [2].

**Fig. 6** Five blocks in the system.


**Fig. 7** User interface.

[*6] https://github.com/tachi-hi/euterpe

**Fig. 8** Photograph of experiment system. There are a laptop PC, in which the system works, and CD player which is connected to the line input of the PC, from which an audio signal is input into the the system.

**Table 5** Parameters of each technical component.

| | Parameter | Value |
|---|---|---|
| | Sampling Rate $f_{\text{samp}}$ | 16,000 [Hz] |
| Short HPSS | Frame length $L_1$ | 512 (=32 [ms]) |
| | Frame hop size $S_1$ | 256 (=16 [ms]) |
| | Parameter $w$ | 1 |
| | Parameter $c$ | 0.2 |
| | Sliding block size $N$ | 7 |
| | Iteration $I$ | 1, 2, 5 or *flexible* |
| Long HPSS | Frame length $L_2$ | 2,048 (=128 [ms]) |
| | Frame hop size $S_2$ | 1,024 (=64 [ms]) |
| | Parameter $w$ | 1 |
| | Parameter $c$ | 0.2 |
| | Sliding block size $N$ | 7 |
| | Iteration $I$ | 1, 2, 5 or *flexible* |
| Key Conversion | Frame length $L$ | 2,048 (=128 [ms]) |
| | Frame hop size $S$ | 256 (=16 [ms]) |
| | LPC coefficient | 15 |
| | Sliding block size $N$ | 7 |
| | Iteration $I$ | 1, 2, 5 or *flexible* |

input SNR = 0 dB, while it is about 2.5 dB if we used parameters in Section 3, though the value is still comparable to that of RPCA [17].

### 6.2 Subsidiary Comments on Implementation

#### 6.2.1 On the Pipeline Model

In a very simple audio application, the whole processing can be written in the callback function of audio I/O API such as PortAudio [2]. In contrast, in our more complicated system, we need to consider a little more sophisticated architecture, in order to avoid the "audio glitch" and the "underflow" (frame rate drop), etc.

In this paper, instead of applying HPSS and key conversion in the callback function in audio I/O, we designed the callback function to just copy the already-processed data stored in the buffer to the audio output. The core algorithms, namely HPSS and key conversion, work in other threads independently of the I/O; see e.g., Ref. [33] in Section 2.2.3.

#### 6.2.2 Iteration Strategy: Flexible Iteration

In Section 4.2 and RTISI-LA, although iterating the updating formula once ($I = 1$) results in decent audio quality, larger number of iteration basically results in better audio quality.

A simple strategy to increase the number of iteration is setting $I$ a constant which is bigger than 1, such as $I = 5$. However, we can expect that there should be some more chances to update the spectrogram more than $I$ times, especially if we have a computer of higher performance. In order to increase the number of iteration $I$ as large as possible, we considered to make $I$ more flexible, instead of setting $I$ by a constant.

This is simply achieved by the following architecture: we implement the two processes (namely "enqueue/dequeue" and "update") loosely-coupled, and make them work in individual threads[*7]. In this architecture, "update" threads are working pauselessly, while "enqueue/dequeue" threads work only when it is required. Intuitively speaking, "update" thread concentrates solely on the quality of their main occupation, and turns over its output to the subsequent component only when it is required, especially when the amount of the stock in the buffer between them is lower than the predetermined threshold.

**Table 6** Machine Specification.

| | | | |
|---|---|---|---|
| PC | DELL PRECISION M4500 | ASUS Eee PC | Apple MacBook Air |
| Category | Laptop Workstation | Netbook | Laptop |
| CPU | Intel Core i7 Q 740 1.73 GHz | Intel Atom N455 1.66 GHz | Intel Core i5 1.6 GHz |
| RAM | 2 GB | 1 GB | 4 GB |
| OS | Linux Mint 13 Maya (Ubuntu 12.04 Precise) | Xubuntu (Ubuntu 14.04 Trusty) | OS X 10.10.5 Yosemite |
| Compiler | g++ 4.6.3 | g++ 4.8.4 | clang++ 7.0.0 |
| Release | 2010 | 2009 | 2015 |
| Audio Interface | Built-in 3.5 mm audio jack | iBUFFALO BSHSAU01BK USB/3.5 mm converter | |
| Note | On VMware Player (Host OS Windows 7) | – | – |

### 6.3 Experiment

We implemented the system on the basis of the architecture above, and verified that it works on the computers shown in **Table 6**, though we sometimes observed some "underflow" and some other errors of ALSA (Advanced Linux Sound Architecture) on Linux.

#### 6.3.1 Video Example

The attached video file (Video 1) shows the system that we implemented. The song used in the video is extracted from RWC database [11] (RWC-MDB-P-2001 No.7).

Watching the video, we can find that the system in the left PC accepts the audio signal from the right PC, and it processes the signal in real-time. Moving the "volume" slider, the volume of singing voice is reduced first. Moving the "key" slider, the key is converted. We can find there is a little latency, but it may be almost negligible practically. The video also shows both vocal suppression and key conversion work simultaneously.

#### 6.3.2 Throughput

**Table 7** shows the approximate throughput of the system; it displays the approximate number of iterations of each technical component per a second. "Minimal" $U$ is the inverse of the frame shift $S$. That is, $S = 16$ [ms] implied that we should iterate the

[*7] To be specific, we implemented both HPSS and pitch shifter as a class of C++, and both "enqueue/dequeue" and "update" as the member function of the classes. The functions "enqueue/dequeue" and "update" can access the resource of the class, such as the spectrogram, mutually exclusively (MUTEX). For example, when "update" is running, the resource is locked, and "enqueue/dequeue" should wait until "update" terminates. This strategy is a classic, and there may be more sophisticated strategies.

**Table 7** Throughput of each technical component. (top): $U$ [Times/s] being the number of updates per a second. The values are empirically evaluated. (middle): Number of iterations $I$, being the value of the designed value if the iteration strategy is "Fixed," or the value evaluated by $I \equiv U/S^{-1}$ if the iteration strategy is "Flex." (bottom): "Flex." and "Fixed" are the iteration strategy of the component.

| | Short HPSS | Long HPSS | key conv. (RTISI-LA) | Note |
|---|---|---|---|---|
| DELL | $U \approx$4000<br>$I \approx$64<br>*Flex.* | $U \approx$1000<br>$I \approx$64<br>*Flex.* | $U \approx$1000<br>$I \approx$16<br>*Flex.* | |
| DELL | $U \approx$312<br>$I = 5$<br>Fixed | $U \approx$78<br>$I = 5$<br>Fixed | $U \approx$1400<br>$I \approx$22<br>*Flex.* | Video example |
| ASUS | N/A<br>$I = 1$<br>Fixed | N/A<br>$I = 1$<br>Fixed | N/A<br>N/A<br>*Flex.* | Not stable enough |
| ASUS | $U \approx$124<br>$I = 2$<br>Fixed | $U \approx$31<br>$I = 2$<br>Fixed | $U \approx$124<br>$I = 2$<br>Fixed | |
| MacBook Air | $U \approx 32000$<br>$I \approx 512$<br>*Flex.* | $U \approx 7700$<br>$I \approx 493$<br>*Flex.* | $U \approx 3800$<br>$I \approx 61$<br>*Flex.* | |
| Minimal Requirements | | | | |
| $U$[Times/s]<br>$I$[Times] | $U \geq 62.5$<br>$I \geq 1$ | $U \geq 15.6$<br>$I \geq 1$ | $U \geq 62.5$<br>$I \geq 1$ | rhs= $S^{-1}$ |
| $S$ [ms] | 16 [ms] | 64 [ms] | 16 [ms] | |

updating formulae at least 62.5 times per second. This is the minimal number of iterations which is required to apply the updating formulae to all the bins at least once, keeping up with the real-time.

The table shows that powerful machines, namely the workstation (DELL 2010) and MacBook Air (2015), can execute the updating formulae of HPSS and RTISI-LA luxuriously many times, much more than practically required (see Appendix). The table also shows that even an old netbook (ASUS 2009) can catch up with the real-time, executing the updating formulae twice as many times as the minimal requirement, though it is not stable enough if we employ the flexible iteration strategy.

# 7. Concluding Remarks

## 7.1 Summary and Conclusion

We developed an automatic karaoke generating system "Euterpe," which aimed at generating a karaoke signal automatically from a wider range of music signals, especially monaural music signals, to which we cannot apply the simple center-removal-based "karaoke generator," contrary to the professionally-created stereo signals.

This system has two functions: singing voice suppression and key conversion.

- The singing voice suppression is based on two-stage HPSS, which is effective as a vocal enhancer in our previous study. Simple evaluation and sample audio files showed that the technique also works effectively as a vocal suppressor.
- The key conversion is based on the pitch shift technique based on phase vocoder, but has a little improvement from G&L algorithm: it works in real-time because it is based on RTISI-LA, and it keeps the timbre of the sound because of the LPC analysis.

We also described the architecture of the implementation of the whole systems, which is based on the pipeline model. Because of the architecture, we achieved a real-time karaoke generating system, which is especially important when we consider a client-side streaming-based karaoke generating system. The attached video file may be showing that the latency and the sound quality is acceptable for now.

## 7.2 Future Work

There is still room for improvement of the sound quality such as the following.

- The obtained "accompaniment" has ambiguous articulation. This problem basically comes from the long HPSS, which is, however, the most important component of vocal suppression.
- The volume of the sound is uneven through the song. Especially the "harmonic" components are occasionally quite loud.
- The sound quality is rather poor especially when the interval of key conversion is large, such as $n \lesssim -5, 5 \lesssim n$.

These issues should be addressed in the future, but we should note that too complicated techniques may impair the real-time nature.

Another possible future work will be the development of a system which is more faithful to ordinary karaoke systems. Adding some other functions such as showing lyrics is a challnege for the future.

Another future work will be the development of more intelligent karaoke systems, such as one that follows the user's singing, similar to some of the automatic accompaniment systems [4], [32] that follow the MIDI piano. This may be a dreamy technical challenge in the future.

**Acknowledgments** Part of this work was supported by Grant-In-Aid for JSPS Research Fellows (22-6961) and Kakenhi Grant-In-Aid for Scientific Research (A). In the process of the development of the real-time system we were supported by Mr. Takuho Nakano (program test) and Dr. Takuya Nishimoto (advice on real-time audio I/O).

## References

[1] Audacity Vocal Removal Plug-ins, available from ⟨http://wiki.audacityteam.org/wiki/Vocal_Removal_Plug-ins⟩ (accessed 2016-02).
[2] Bencina, R. and Burk, P.: PortAudio – An Open Source Cross Platform Audio API, *Proc. 2001 International Computer Music Conference* (*ICMC*), pp.263–266 (2001).
[3] Charpentier, F.J. and Stella, M.G.: Diphones Synthesis using an Overlap-add Technique for Speech Waveforms Concatenation, *Proc. ICASSP*, pp.2015–2018 (1986).
[4] Dannenberg, R.: An On-line Algorithm for Real-time Accompaniment, *Proc. ICMC*, pp.193–198 (1984).
[5] Moulines, E. and Charpentier, F.: Pitch-synchronous Waveform Processing Techniques for Text-to-speech Synthesis using Diphones, *Speech Communication*, Vol.9, No.5-6, pp.453–467 (1990).
[6] FitzGerald, D. and Gainza, M.: Single Channel Vocal Separation using Median Filtering and Factorisation Techniques, *ISAST Trans. Electronic and Signal Processing*, Vol.4, No.1, pp.62–73 (2010).
[7] Flanagan, J.L. and Golden, R.M.: Phase Vocoder, *Bell System Technical Journal*, Vol.45, pp.1493–1509 (1966).
[8] Frigo, M. and Johanson, S.G.: FFTW users manual, available from ⟨http://www.fftw.org/doc/⟩, ⟨http://www.fftw.org/fftw3.pdf⟩.
[9] Frigo, M. and Johanson, S.G.: The design and implementation of FFTW3, *Proc. IEEE*, Vol.93, No.2, pp.216–231 (2005).
[10] Goto, M.: SmartMusicKIOSK: Music Listening Station with Chorus-Search Function, *Proc. 16th Annual ACM Symposium on User Inter-*

*face Software and Technology*, Vol.5, pp.31–40 (2003).
[11] Goto, M.: Development of the RWC Music Database, *Proc. 18th International Congress on Acoustics* (*ICA 2004*), pp.I–553–556 (2004).
[12] Goto, M.: Active Music Listening Interfaces Based on Signal Processing, *Proc. 2007 IEEE International Conferences on Acoustics, Speech, and Signal Processing*, pp.1441–1444 (2007).
[13] Griffin, D.W. and Lim, J.S.: Signal Estimation From Modified Short-Time Fourier Transform, *IEEE Trans. Audio, Speech, and Signal Processing*, Vol.32, No.2, pp.236–243 (1984).
[14] Hamasaki, M., Takeda, H., Hope, T. and Nishimura, T.: Network Analysis of an Emergent massively Collaborative Creation Community – How Can People Create Videos Collaboratively without Collaboration?, *Proc. 3rd International ICWSM Conference, AAAI*, pp.222–225 (2009).
[15] Hamasaki, M., Takeda, H. and Nishimura, T.: Network Analysis of Massively Collaborative Creation of Multimedia Contents – Case Study of Hatsune Miku Videos on Nico Nico Douga, *Proc. 1st International Conference on Designing Interactive User Experiences for TV and Video* (*UXTV*), pp.165–168, ACM (2008).
[16] Hsu, C.-L. and Jang, J.-S.R.: on The Improvement of Singing Voice Separation for Monaural Recordings using The MIR-1K Dataset, *IEEE Trans. Audio, Speech, and Language Processing* (2010).
[17] Huang, P.-S., Chen, S.D., Smaragdis, P. and H.-Johnson, M.: Singing-voice separation from monaural recordings using robust principal component analysis, *Proc. 2012 IEEE International Conference on Audio, Speech, and Signal Processing* (*ICASSP 2012*), pp.57–60 (2012).
[18] Ikemiya, Y., Yoshii, K. and Itoyama, K.: Singing Voice Analysis and Editing based on Mutually Dependent F0 Estimation and Source Separation, *Proc. ICASSP* (2015).
[19] Imam, J.: Young Listeners Opting to Stream, Not Own Music (2012), available from ⟨http://edition.cnn.com/2012/06/15/tech/web/music-streaming/index.html⟩.
[20] Itakura, F. and Saito, S.: Digital Filtering Techniques for Speech Analysis and Synthesis, *Proc. International Conference on Independent Component Analysis and Blind Signal Separation*, Vol.25-C-1, pp.261–264 (1971).
[21] Japan Productivity Center: *White Paper of Leisure 2011* (2011) (in Japanese).
[22] Klapuri, A.: Multiple Fundamental Frequency Estimation based on Harmonicity and Spectral Smoothness, *IEEE Trans. Speech Audio Process.*, Vol.11, No.6, pp.804–816 (2003).
[23] Klapuri, K. and Davy, M.: *Signal Processing Methods for Music Transcription*, Springer (2006).
[24] Laroche, J. and Dolson, M.: Improved Phase Vocoder Time-Scale Modification of Audio, *IEEE Trans. Speech and Audio Processing*, Vol.7, No.3, pp.323–332 (1999).
[25] Le Roux, J.: Exploiting Regularities in Natural Acoustical Scenes for Monaural Audio Signal Estimation, Decomposition, Restoration and Modification, Ph.D. Thesis, The University of Tokyo (2009).
[26] Le Roux, J., Ono, N. and Sagayama, S.: Explicit Consistency Constraints for STFT Spectrograms and Their Application to Phase Reconstruction, *Proc. Workshops on Statistical and Perceptual Audition* (*SAPA*) (2008).
[27] Le Roux, J. and Vincent, E.: Consistent Wiener Filtering for Audio Source Separation, *IEEE Signal Processing Letter*, pp.217–220 (2013).
[28] Li, Y. and Wang, D.L.: Separation of Singing Voice From Music Accompaniment for Monaural Recordings, *IEEE Trans. Audio, Speech, and Language Processing*, Vol.15, No.4, pp.1475–1487 (2007).
[29] Mizuno, Y., Le Roux, J., Ono, N. and Sagayama, S.: Real-time Time-scale/Pitch Modification of Music Signal by Stretching Power Spectrogram and Consistent Phase Reconstruction, *Proc. ASJ Spring Meeting* (2009) (in Japanese).
[30] Mizuno, Y.: Master Thesis, The University of Tokyo (2012) (in Japanese).
[31] Nakamura, T. and Kameoka, H.: Fast Signal Reconstruction from Magnitude Spectrogram of Continuous Wavelet Transform based on Spectrogram Consistency, *Proc. DAFx* (2014).
[32] Nakamura, T., Nakamura, E. and Sagayama, S.: Automatic Score Following to Musical Performance with Errors and Arbitrary Repeats and Skips for Automatic Accompaniment, *Proc. SMC*, pp.299–304 (2013).
[33] Nichols, B., Buttlar, D. and Farrell, J.P.: *Pthreads Programming: A POSIX Standard for Better Multiprocessing*, O'Reilly Nutshell (1996).
[34] Ono, N., Miyamoto, K., Kameoka, H., Le Roux, J., Uchiyama, Y., Tsunoo, E., Nishimoto, T. and Sagayama, S.: Harmonic and Percussive Sound Separation and its Application to MIR-related Tasks, *Advances in Music Information Retrieval*, Ras, Z.W. and Wieczorkowska, A. (eds.), *Studies in Computational Intelligence*, Springer 274, pp.213–236 (2010).
[35] Ono, N., Miyamoto, K., Kameoka, H. and Sagayama, S.: A Real-time Equalizer of Harmonic and Percussive Components in Music Signals, *Proc. International Symposium on Music Information Retrieval* (*ISMIR2008*), pp.139–144 (2008).
[36] Ono, N., Miyamoto, K., Le Roux, J., Kameoka, H. and Sagayama, S.: Separation of a Monaural Audio Signal into Harmonic/Percussive Components by Complementary Diffusion on Spectrogram, *Proc. European Signal Processing Conference* (*EUSIPCO2008*) (2008).
[37] Ozerov, A., Philippe, P., Bimbot, F. and Gribonval, R.: Adaptation of Bayesian Models for Single-Channel Source Separation and Its Application to Voice/Music Separation in Popular Songs, *IEEE Trans. Audio, Speech, and Language Processing*, Vol.15, No.5, pp.1564–1578 (2007).
[38] Ozerov, A., Philippe, P., Gribonval, R. and Bimbot, F.: One Microphone Singing Voice Separation using Source-Adapted Models, *Proc. 2005 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics* (*WASPAA*) pp.90–93 (2005).
[39] Perraudin, N., Balazs, P. and Søndergaard, P.L.: A fast Griffin-Lim algorithm, *Proc. WASPAA* (2013).
[40] Rabiner, L. and Juang, B.-H.: *Fundamentals of Speech Recognition*, Prentice Hall (1993).
[41] Rafii, Z. and Pardo, B.: A Simple Music/Voice Separation Method Based on The Extraction of The Repeating Musical Structure, *Proc. 2011 IEEE International Conference on Audio, Speech, and Signal Processing* (*ICASSP 2011*), pp.221–224 (2011).
[42] Raj, B., Smaragdis, P., Shashanka, M. and Singh, R.: Separating a Foreground Singer from Background Music, *Proc. International Symposium Frontiers of Research Speech and Music* (*FRSM*) (2007).
[43] Röbel, A. and Rodet, X.: Efficient Spectral Envelope Estimation and its Application to Pitch Shifting and Envelope Preservation, *Proc. DAFx 2005* (2005).
[44] Ryynänen, M., Virtanen, T., Paulus, J. and Klapuri, A.: Accompaniment separation and karaoke application based on automatic melody transcription, *Proc. 2008 IEEE International Conference on Multimedia and Expo* (*ICME 2008*) pp.1417–1420 (2008).
[45] Schörkhuber, G., Klapuri, A. and Sontacchi, A.: Audio Pitch Shifting using Constant-Q Transform, *Journal of Audio Engineering Society*, Vol.61, No.7/8, pp.562–572 (2013).
[46] Simpson, A.J.R., Roma, G. and Plumbley, M.D.: Deep Karaoke: Extracting Vocals from Musical Mixtures using a Convolutional Deep Neural Network (2015). arXiv:1504.04658.
[47] Sprechmann, P., Bronstein, A. and Sapiro, G.: Real-time Online Singing Voice Separation from Monaural Recordings using Robust Low-rank Modeling, *Proc. ISMIR* (2012).
[48] Tachibana, H.: Music Signal Processing Exploiting Spectral Fluctuation of Singing Voice using Harmonic/Percussive Sound Separation, PhD Thesis, The University of Tokyo (2014).
[49] Tachibana, H., Mizuno, Y., Ono, N. and Sagayama, S.: Euterpe: A Real-time Automatic Karaoke Generation System based on Singing Voice Suppression and Pitch Conversion, *Proc. ASJ autumn meeting* (2012) (in Japanese).
[50] Tachibana, H., Ono, N., Kameoka, H. and Sagayama, S.: Harmonic/Percussive Sound Separation Methods Based on Anisotropic Smoothness of Spectrograms, *IEEE Trans. ASLP*, Vol.22, No.12, pp.2059–2073 (2014).
[51] Tachibana, H., Ono, N. and Sagayama, S.: Vocal Sound Suppression in Monaural Audio Signals by Multi-stage Harmonic-Percussive Sound Separation (HPSS), *Proc. ASJ Spring Meeting* (2009) (in Japanese).
[52] Tachibana, H., Ono, N. and Sagayama, S.: Singing Voice Enhancement in Monaural Music Signals Based on Two-stage Harmonic/Percussive Sound Separation on Multiple Resolution Spectrograms, *IEEE Trans. ASLP*, Vol.22, No.1, pp.228–237 (2014).
[53] Tachibana, H., Ono, T., Ono, N. and Sagayama, S.: Melody Line Estimation in Homophonic Music Audio Signals Based on Temporal-Variability of Melodic Source, *Proc. International Conference on Acoustics, Speech, and Signal Processing* (*ICASSP*), pp.425–428 (2010).
[54] Vembu, S. and Baumann, S.: Separation of Vocals from Polyphonic Audio Recordings, *Proc. International Symposium on Music Information Retrieval* (*ISMIR 2005*), pp.337–344 (2005).
[55] Virtanen, T., Mesaros, A. and Ryynänen, M.: Combining Pitch-Based Inference and Non-Negative Spectrogram Factorization in Separating Vocals From Polyphonic Music, *Proc. SAPA*, pp.17–20 (2008).
[56] Xun, Z. and Tarocco, F.: *Karaoke: The Global Phenomenon*, Reaktion Books (2007).
[57] Yoshii, K., Goto, M., Komatani, K., Ogata, T. and Okuno, H.G.: Drumix: An Audio Player With Real-Time Drum-Part Rearrangement Functions for Active Music Listening, *IPSJ Journal*, Vol.48, No.3, pp.1229–1239 (2007).
[58] Zhu, B., Li, W., Li, R. and Xue, X.: Multi-stage Non-negative Matrix Factorization for Monaural Singing Voice Separation, *IEEE Trans.*

*ASLP*, Vol.21, No.10, pp.2096–2107 (2013).
[59] Zhu, X., Beauregard, G. and Wise, L.: Real-Time Iterative Spectrum Inversion With Look Ahead, *IEEE Trans. Audio, Speech, and Language Processing*, Vol.15, No.5, pp.1645–1653 (2007).

# Appendix

## A.1 HPSS Updating Formula

The functions Eqs. (3), (4), and (5) are specifically written as follows,

$$H_{n,k} \leftarrow \frac{H^{\mathrm{ave}}_{n,k} + \sqrt{(H^{\mathrm{ave}}_{n,k})^2 + (2+c)c\theta_{n,k}Y^2_{n,k}}}{2+c}$$

$$P_{n,k} \leftarrow \frac{P^{\mathrm{ave}}_{n,k} + \sqrt{(P^{\mathrm{ave}}_{n,k})^2 + (2+cw^{-1})cw^{-1}(1-\theta_{n,k})Y^2_{n,k}}}{2+cw^{-1}}$$

$$\theta_{n,k} \leftarrow \frac{H^2_{n,k}}{H^2_{n,k}+P^2_{n,k}}$$

where

$$H^{\mathrm{ave}}_{n,k} = (H_{n+1,k} + H_{n-1,k})/2, \qquad P^{\mathrm{ave}}_{n,k} = (P_{n,k+1} + P_{n,k-1})/2$$

This update never increases the objective function.

The curve in **Fig. A·1** shows the relation between the number of iteration ($\tau$) and the compensated objective function $J^{(\tau)} - J^{(\infty)}$ *8 i.e., difference from the local optimum, in batch HPSS. It shows that the objective function quite rapidly decreases, and several dozens of the iterations gives a solution sufficiently close to the local optimum (note that $y$-axis is logarithmic scale.)

## A.2 Difference between Batch HPSS and Streaming HPSS

The difference between the batch HPSS and streaming HPSS is the matter of the order of the coordinates to apply the update formulae in the coordinate descent.

The solutions are not exactly identical, but our empirical experiment suggests that streaming HPSS also sufficiently rapidly decreases the objective function, as shown in Fig. A·1. Each point in Fig. A·1 shows the terminal values of the objective function after applying the streaming HPSS of parameter $(N, I)$: $N$ being the size of the sliding window, and $I$ being the number of iteration in the window. They will not converge to the identical local optimum even if $I$ increases, but it shows that the terminal values are still much smaller than the levels of those of the initial several updates of the batch HPSS.

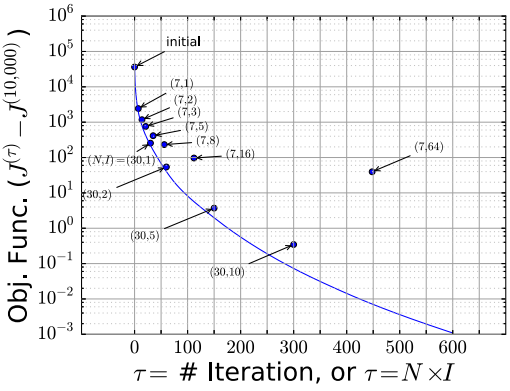
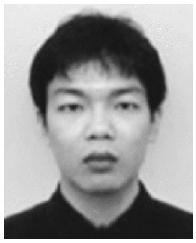



**Fig. A·1** Values of the objective function after the $\tau$-th update, where $\tau$ is the number of iteration of the batch HPSS (curve), or $\tau = N \times I$ of streaming HPSS of parameter $(N, I)$ (each point). We used `Ani_1_01.wav` in MIR-1K dataset as a sample input.

*8 As we cannot iterate the procedure infinite times, we used $J^{(10,000)}$ instead of $J^{(\infty)}$.



**Hideyuki Tachibana** received his B.E. degree in mathematical engineering and information physics, M.E. and Ph.D. degrees in information physics and computing, from the University of Tokyo in 2008, 2010 and 2014 respectively. He was with the Faculty of Interdisciplinary Mathematical Sciences, Meiji University in 2014. He has been working for a startup as an ML & NLP research scientist since 2015. He has been interested in signal processing, AI, and their applications including but not limited to music. He is a member of IEEE, ACM, IPSJ, IEICE, and ASJ.

**Yu Mizuno** received his B.E. degree in mathematical engineering and information physics, M.E. degree in information physics and computing, from the University of Tokyo in 2009 and 2012 respectively. He is currently working at Aichi Prefectural Government.

**Nobutaka Ono** received his B.E., M.S., and Ph.D. degrees in Mathematical Engineering and Information Physics from the University of Tokyo, Japan, in 1996, 1998, 2001, respectively. He joined the Graduate School of Information Science and Technology, the University of Tokyo, Japan, in 2001 as a Research Associate and became a Lecturer in 2005. He moved to the National Institute of Informatics, Japan, as an Associate Professor in 2011. His research interests include acoustic signal processing, specifically, microphone array processing, source localization and separation, and optimization algorithms for them. He is the author or co-author of more than 180 articles in international journal papers and peer-reviewed conference proceedings. He was a Tutorial speaker at ISMIR 2010, a special session chair in EUSIPCO 2013 and 2015, a chair of SiSEC (Signal Separation Evaluation Campaign) evaluation committee in 2013 and 2015. He was an Associate Editor of the IEEE Transactions on Audio, Speech and Language Processing during 2012 to 2015. He has been a member of IEEE Audio and Acoustic Signal Processing (AASP) Technical Committee since 2014. He is a senior member of the IEEE Signal Processing Society, and a member of the Acoustical Society of Japan (ASJ), the Institute of Electronics, Information and Communications Engineers (IEICE), the Information Processing Society of Japan (IPSJ), and the Society of Instrument and Control Engineers (SICE) in Japan. He received the Sato Paper Award and the Awaya Award from ASJ in 2000 and 2007, respectively, the Igarashi Award at the Sensor Symposium on Sensors, Micromachines, and Applied Systems from IEEJ in 2004, the best paper award from IEEE ISIE in 2008, Measurement Division Best Paper Award from SICE in 2013, the best paper award from IEEE IS3C in 2014, the excellent paper award from IIHMSP in 2014 and the unsupervised learning ICA pioneer award from SPIE.DSS in 2015.

**Shigeki Sagayama** received his B.E., M.E. and Ph.D. degrees from the University of Tokyo, Japan in 1972, 1974 and 1998, respectively, all in mathematical engineering and information physics. He joined Nippon Telegraph and Telephone Public Corporation (currently, NTT) in 1974 and started his career in speech analysis, synthesis and recognition at NTT Labs in Musashino, Japan. From 1990, he was Head of Speech Processing Department, ATR Interpreting Telephony Laboratories, Kyoto, Japan to pursue an automatic speech translation project. From 1993, he was responsible for speech recognition, synthesis and dialog systems at NTT Human Interface Laboratories, Yokosuka, Japan. In 1998, he became a professor of Graduate School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan. In 2000, he was appointed Professor, Graduate School of Information Science and Technology, the University of Tokyo, Japan. He is an author or co-author of approximately 700 technical papers on processing and recognition of speech, music, acoustic signals, hand-writing and images. Prof. Sagayama received National Invention Award from the Institute of Invention of Japan in 1991, Chief Officials Award for Research Achievement from the Science and Technology Agency of Japan in 1996 and other academic awards including Paper Awards from the Institute of Electronics, Information and Communications Engineers, Japan (IEICEJ), in 1996 and from Information Processing Society of Japan (IPSJ) in 1995. He is the Chair of IEEE Signal Processing Society Japan paper, a fellow of IEICEJ and a member of ASJ (Acoustical Society of Japan) and IPSJ.