A Quick Search Method for Audio and Video Signals Based on Histogram Pruning

Kunio Kashino, *Member, IEEE,* Takayuki Kurozumi, and Hiroshi Murase, *Senior Member, IEEE*

***Abstract*—This paper proposes a quick method of similarity-based signal searching to detect and locate a specific audio or video signal given as a query in a stored long audio or video signal. With existing techniques, similarity-based searching may become impractical in terms of computing time in the case of searching through long-running (several-days' worth of) signals. The proposed algorithm, which is referred to as time-series active search, offers significantly faster search with sufficient accuracy. The key to the acceleration is an effective pruning algorithm introduced in the histogram matching stage. Through the pruning, the actual number of matching calculations can be reduced by 200 to 500 times compared with exhaustive search while guaranteeing exactly the same search result. Experiments show that the proposed method can correctly detect and locate a 15-s signal in a 48-h recording of TV broadcasts within 1 s, once the feature vectors are calculated and quantized. As extentions of the basic algorithm, efficient AND/OR search methods for searching for multiple query signals and a feature dithering method for coping with signal distortion are also discussed.**

***Index Terms*—Audio fingerprinting, audio search, multimedia databases, multimedia information retrieval, video search.**

I. Introduction

THIS paper proposes a method for searching quickly through a long audio or video signal (termed a *stored signal*) to detect and locate a known reference audio or video signal (termed a *query signal*).

Audio and video data from radio, television, databases, or on the Internet has been a source of recent research interest. Among the many studies that have targeted audio or video information search, most have dealt with so-called content-based retrieval by means of indexing and classifying audio or video information. For example, in image or video retrieval tasks, a major issue has been constructing efficient indexes [1]–[4]. Similarly, in audio retrieval tasks, most works have been based on high level information such as audio content classification (e.g., indexes for speech segments and nonspeech segments), recognized speeches, or transcribed musical pieces [5]–[9].

In contrast, this study concerns a similarity-based search, which is the search of and retrieval from unlabeled audio or video archives based solely on a signal similarity measure.

Manuscript received May 2, 2001; revised June 27, 2002. The associate editor coordinating the review of this paper and aproving it for publication was Prof. Wayne Wolf.

K. Kashino and T. Kurozumi are with the NTT Communication Science Laboratories, Atsugi-shi 243-0198, Japan (e-mail: kunio@eye.brl.ntt.co.jp; kurozumi@eye.brl.ntt.co.jp).

H. Murase is with the Graduate School of Information Science, Nagoya University, Nagoya-shi 464–8603, Japan (e-mail: murase@is.nagoya-u.ac.jp).

Digital Object Identifier 10.1109/TMM.2003.813281

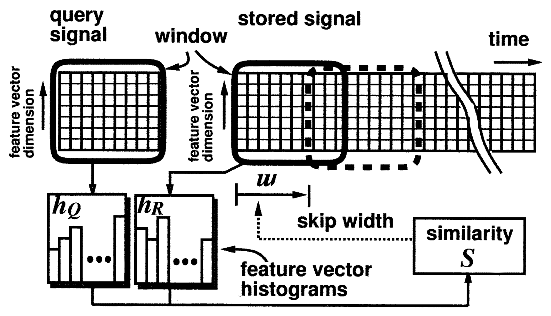


Fig. 1. Block diagram of the proposed search algorithm.

Though the range of applications for similarity-based search may seem narrow compared to content-based retrieval based on high level information, this is actually not the case: applications, such as detection and statistical analysis of broadcasted music or commercials, or copyright management on the Internet, are possible. Just as high-speed text search algorithms have come into widespread use, quick search algorithms for audio or video signals may too become basic technologies of handling multimedia information.

A conventional approach for the similarity-based search of audio or video data (hereafter time-series data) is the signal detection technique based on correlations of data itself or on feature vectors extracted from the data [10]. However, with this technique, searching may become impractical in terms of computing time in the case of long-running (e.g., several-days' worth of) stored signals or many reference signals. Search speed might be improved through a rougher matching, but that would inevitably reduce search accuracy and reliability.

The algorithm proposed in this paper offers significantly faster search with sufficient accuracy. The key to the acceleration is an effective pruning algorithm introduced in the feature matching stage using feature histograms. Through the pruning, the actual number of matching calculations is reduced by 200 to 500 times compared with exhaustive search while mathematically guaranteeing the same search result.

The rest of this paper is organized as follows. Section II describes the basic algorithm, and discusses some extension to efficient AND/OR search and feature distortion absorption. Section III evaluates the proposed algorithms under realistic circumstances. Finally, Section IV gives conclusions.

II. Theory

A. Basic Algorithm

Fig. 1 outlines the proposed algorithm. Firstly, the feature vectors are calculated from both the query signal and stored signal. The windows are then applied to both the query-signal and stored-signal feature vectors. The feature vectors over the windows are classified into a certain number of types, and the number of occurrences of each feature type is counted to create the histogram. The window length is the same as the query signal duration, though we will later discuss cases where the window is divided into multiple small windows. Thirdly, similarity between the query-signal histogram and stored-signal histogram is calculated. When the similarity exceeds a threshold value chosen in advance, the query signal is considered to be detected and located in the stored signal. In the last step, the window on the stored signal is shifted forward in time and the search proceeds. We call this algorithm "time-series active search."

B. Features

Various types of features have been proposed in the audio and image retrieval field. The widely-used audio features include zero-crossing rates of waveforms [11], [12], short-time frequency spectrum, linear predictive coding coefficients (LPC) [13], [14], and mel-frequency cepstral coefficients (MFCC) [14]–[16]. On the other hand, the image features include color distribution [17], [18], shape features [19], and spatial features such as discrete cosine transformation (DCT) coefficients [20] and wavelets [21].

Among those, our preliminary experiments showed that the short-time frequency spectrum calculated by a bandpass filter bank and the color distribution provide sufficient accuracy for our similarity-based signal search task [22], [23]. Therefore, as computationally inexpensive features, the short-time frequency spectrum and the color distribution are specifically introduced here, as audio and video features, respectively.

Audio feature vector $\boldsymbol{f}(k)$ is written as

$$\boldsymbol{f}(k) = (f_1(k), f_2(k), \ldots, f_N(k)) \tag{1}$$

where k is the sampled time. The element $f_j(k)$ is the normalized short-time power spectrum, which is given as

$$f_j(k) = \alpha(k) Y_j(k) \tag{2}$$

$$Y_j(k) = \sum_{t=k-M+1}^{k} y_j^2(t), \tag{3}$$

$$k = lM \quad (l = 1, 2, \ldots) \tag{4}$$

where $y_j(t)$ is the output waveform of bandpass filter j at time t, M the time support of the feature vector, N the number of frequency channels, and $\alpha(k)$ a normalization constant defined as

$$\alpha(k) = \frac{1}{\max_j(Y_j(k))}. \tag{5}$$

Bandpass filter $y_j(t)$ can be implemented as a 2nd-order infinite impulse response (IIR) filter and thus it is computationally inexpensive.

The video feature vector is based on colors. To extract it, the image in each video frame is divided into W subimages. Letting p designate the video frame number and j the subimage $(j = 1, \ldots, W)$, we express the video feature vector $\boldsymbol{x}(p)$ as

$$\boldsymbol{x}(p) = (x_{1r}(p), x_{1g}(p), x_{1b}(p), \ldots, x_{jc}(p), \ldots, x_{Wr}(p), x_{Wg}(p), x_{Wb}(p)) \tag{6}$$

where c denotes either r (red), g (green), or b (blue). The number of subimages W is empirically chosen. x_{jc} is the normalized color (red, green, or blue) value given as

$$x_{jc}(p) = \frac{\bar{y}_{jc}(p) - \min_i \bar{y}_{ic}(p)}{\max_i \bar{y}_{ic}(p) - \min_i \bar{y}_{ic}(p)} \tag{7}$$

where

$$\bar{y}_{ic}(p) = \frac{1}{|I_i(p)|} \sum_{q \in I_i(p)} y_{qc}(p) \tag{8}$$

where $I_i(p)$ is a set of pixels in the i-th subimage, $|\cdot|$ stands for the number of pixels, and $y_{qc}(p)$ denotes the color value of the pixel q. The color information is employed because it has been successfully applied in visual object recognition [17], [24].

C. Histogram Modeling

In this paper, a histogram is a frequency distribution of the feature vector occurrences over the window. The frequency distribution is obtained by classifying the feature vectors according to a certain vector quantization algorithm and counting the number of occurrences for each quantized code. In the vision field, histograms have been employed as image models by many researchers; for example, Swain *et al.* have shown that the histogram space provides sufficient inter-object discrimination in vision [17].

Since the feature vectors are not uniformly distributed in the feature space, feature vector density should be considered in the classification process in order to efficiently represent signals with a histogram. Thus, we use the Linde–Buzo–Gray (LBG) algorithm to create a quantization codebook. The LBG algorithm enables the number of codes assigned in the feature space to reflect the feature vector density; that is, the number of codes for regions where feature vectors are dense becomes greater than that for regions where they are sparse. Then, in the quantization stage, an input feature vector is assigned the nearest code in the codebook.

Histogram $\boldsymbol{h}$ is then defined as

$$\boldsymbol{h} = (h_1, h_2, \ldots, h_l, \ldots, h_L) \tag{9}$$

where L is the number of histogram bins, i.e., the codebook size in the above mentioned vector quantization process, and h_l is the number of feature vectors classified into l-th quantization code observed over the window. As mentioned earlier, the typical window length is the query signal duration. For simplicity, however, we will not explicitly express the window length in mathematical notations for histograms unless specifically needed.

The similarity between the query- and stored-signal histograms over the windows can be determined in several ways, for example, by using the L_1 and L_2 distance measures. Here, we employ histogram intersection, which is equivalent to the L_1 distance measure. The similarity is defined as

$$S(\boldsymbol{h}_Q, \boldsymbol{h}_R) = \frac{1}{D} \sum_{l=1}^{L} \min(h_{Ql}, h_{Rl}) \tag{10}$$

where $\boldsymbol{h}_Q$ and $\boldsymbol{h}_R$ are the histograms for the query and the stored signal respectively, and D is the window length. The histogram intersection measure is used because it is computationally simple, and it has been used successfully in visual object detection [17].

D. Window Skipping

As the window for the stored signal shifts forward in time, similarity based on the query- and stored-signal histograms shows a certain continuity from one time step to the next. The time-series active search takes advantage of this by computing an upper bound of the similarity measure as a function of the time step and skipping all intermediate time-step similarity evaluations until this upper bound exceeds the detection threshold.

The upper bound on $S(\boldsymbol{h}_Q, \boldsymbol{h}_R)$ is

$$S^u(\boldsymbol{h}_Q, \boldsymbol{h}_R(n_2)) = S(\boldsymbol{h}_Q, \boldsymbol{h}_R(n_1)) + \frac{n_2 - n_1}{D} \tag{11}$$

where $\boldsymbol{h}_R(n_1)$ and $\boldsymbol{h}_R(n_2)$ are the histograms created by the stored-signal window for frame numbers n_1 and n_2. Using (11), we can derive the skip width for the window straightforwardly:

$$w = \begin{cases} \lfloor D(\theta - S) \rfloor + 1, & \text{if } S < \theta, \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

where w is the skip width, and $\lfloor x \rfloor$ means the greatest integral value less than x. It should be noted that it is guaranteed that we will not miss any sections that will give similarity values greater than θ, even if we skip the width w given by the (12). This is because (11) states that S can not be greater than θ before the window moves forward by the width w.

In the sense that the proposed method accelerates the search guaranteeing that nothing is missed, the method is similar to the quick string matching methods such as Boyer–Moore [25] and Knuth–Morris–Pratt [26] algorithms. However, the idea of skipping here is different from those algorithms in that our algorithm is based on the similarity upper bound and property of histograms.

E. Detection Criterion

The basic assumption here is that the similarity is not usually very high, according to a certain distribution, but outstandingly high at the positions where the query signal can be considered to be detected on the stored signal.

To decide detection criterion for similarity, it is essential to model the similarity distribution. However, without simplistic and unrealistic assumptions, it is difficult to obtain the similarity distribution in an analytic form. Therefore, we model the distribution on an experimental basis.

Preliminary experiments showed that histogram intersection offered different statistical properties depending on the query signal and stored signal. For this reason, the search threshold should be set according to the query signal and stored signal. When the similarity distribution is modeled by the mean m and standard deviations σ, we determine the search threshold, θ, such that

$$\theta = m + c\sigma \tag{13}$$

where c is an empirically determined constant. The values of m and σ are calculated preliminarily for each query signal by sampling stored-signal feature vectors with respect to the query signal. In (13), c means a distance from the similarity distribution, and thus, we can take the distribution into account. Equation (13) implies the search for segments in stored signals that offer much-higher-than-average similarity to the query signal.

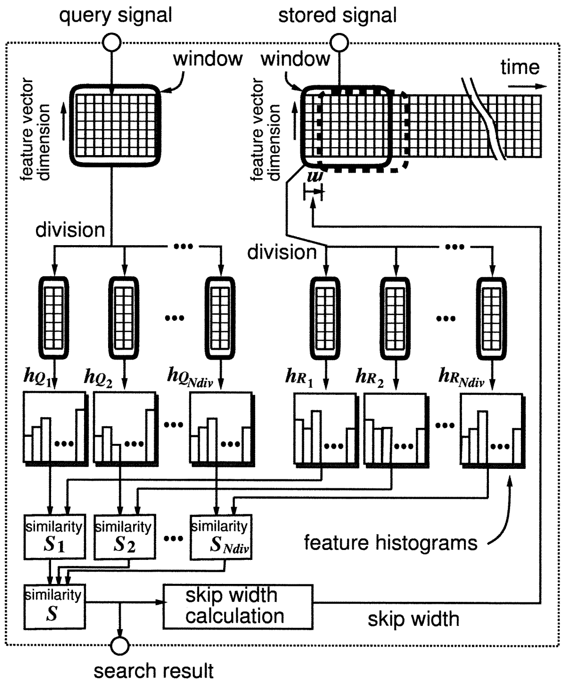


Fig. 2. Search incorporating time order.

F. Search Incorporating Time Order

Since the histogram introduced here is an accumulated representation of feature vectors over the window, the histogram does not reflect the time order of feature vectors. This may sometimes cause insufficient discrimination performance in the search. However, the time order can be considered by dividing windows into a certain number (N_{div}) of subwindows. In such cases, similarity is firstly calculated for each query- and stored-signal histogram pair in the corresponding time positions, and then the similarities are integrated, as shown in Fig. 2.

For similarity integration, we consider taking the minimum. The minimum operation corresponds to the "AND" search with respect to the individual subwindows. The AND search finds the sections where the similarities for all of the multiple query signals $Q_1, \ldots, Q_{N_{\text{div}}}$ exceed a given threshold value. When the similarity S for the total window is defined as

$$S = \min_j (S_j) \tag{14}$$

where S_j means the similarity for j-th subwindow, the problem is to find the section where S exceeds a threshold value.

A straightforward method for this search is to sequentially calculate S_j's one by one, which we call the "sequential method". Letting w_j be the skip width for the j-th subwindow when the original window is at a current position, we express the skip width for the original window as

$$w = \max_j(w_j). \tag{15}$$

Note that we can take the maximum over j rather than the minimum. This is because (14) states that S is always equal to or less than θ if at least one of the S_j's is equal to or less than θ, meaning that S cannot be greater than θ while the window moves forward by any less than the maximum of w_j with respect to j.

In reality, it is often more efficient to stop matching and skip the original window as soon as an S_j that does not exceed θ is found, rather than to calculate S_j for all j. We call this the "sequential-break method."

Meanwhile, when the subwindows are adjacent to each other (that is, when the original window is divided into subwindows as mentioned above):

$$S_{\rm org} \geq \min_j(S_j) \tag{16}$$

holds, where $S_{\rm org}$ is the similarity for the original window and S_j is the similarity for one of the divided windows (see Appendix A for the proof). Note that $S_{\rm org}$ does not always equal S; in fact, (14) and (16) lead to

$$S_{\rm org} \geq S. \tag{17}$$

Equation (17) means that the section where the similarity exceeds the threshold in the search with respect to the divided window can not be skipped by the search with respect to the original window. Hence, one can at first search with respect to the original window, and if the similarity exceeds the threshold, then search with respect to the divided windows. We call this scheme the "refinement method."

At this point, it is interesting to know which of the two methods, the sequential-break method or the refinement one, is more efficient on average. We let $w_{\rm org}$ denote the skip width for the original window, and compare $w_{\rm org}$ and w_j. For simplicity of notation, here we consider the case where $S_{\rm org} < \theta$ and omit ceiling operation in (10); thus $w_{\rm org}$ is written as

$$w_{\rm org} = D(\theta - S_{\rm org}). \tag{18}$$

Equation (18) is calculated as

$$w_{\rm org} = \sum_{j=0}^{N_{\rm div}} (1 - S_j)D_j - (1 - \theta)D \tag{19}$$

where D_j is the length of the subwindows. If we assume that the the similarities are uniformly distributed with respect to time (that is, $S_j = S_{\rm avg} = S$), we obtain

$$w_{\rm org} - w_j = (\theta - S)(D - D_j) > 0 \tag{20}$$

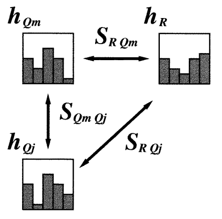


Fig. 3. Relations between the similarities.

using (19). This means that, under the above mentioned assumption, the refinement method should be more efficient than the sequential-break method.

G. Parallel Search

Typical applications of the proposed search method are assumed to include counting the occurrences of specific commercials in TV broadcasts and searching the Internet for copyright management purposes. In such applications, searching for multiple query signals is often needed, and this motivates our research on an efficient parallel searching method for multiple query signals given at the same time.

Suppose we have N query signals Q_j $(j = 1, 2, \ldots, N)$, each of which creates the histogram $\boldsymbol{h}_{Qj}$, and we have the histogram $\boldsymbol{h}_R$ created for the current window position on the stored signal R. For simplicity, we assume that D_j, the number of total votes in the histogram $\boldsymbol{h}_{Qj}$, is equal for all j and is D. Now we calculate S_{RQm}, the similarity between the m-th $(j = m)$ query-signal histogram $\boldsymbol{h}_{Qm}$ and stored-signal histogram H_R, using (10). We are then interested in knowing the upper bound of the similarity S_{RQj} without actually matching the histograms $\boldsymbol{h}_R$ and $\boldsymbol{h}_{Qj}$ $(j \neq m)$. The upper limit of S_{RQj} is given by

$$S_{RQj} \leq 1 - |S_{RQm} - S_{QmQj}|. \tag{21}$$

This inequality is derived as follows. Fig. 3 illustrates the relation between the similarities. Suppose we have just performed a matching calculation between the m-th query signal and a stored signal and obtained S_{RQm}. We assume that S_{QmQj} has been calculated for all m and j prior to the search process. We consider the following two cases:

1) Case 1 $S_{RQm} \leq S_{QmQj}$: Supposing that the m-th query signal and j-th query signal are very similar will help in understanding this case.

Now we introduce a new symbol H that refers to a set whose members are the individual feature vectors voted to a histogram. Then, $\{H_{Qm} \cap H_R\}$ denotes a set comprising the elements of H_{Qm} (that is, the feature vectors voted to H_{Qm}) contributing to the similarity to H_R. Letting $|H_{Qm} \cap H_R|$ be the number of members of the set, (10) can be written as

$$|H_{Qm} \cap H_R| = DS_{RQm}. \tag{22}$$

In the same way,

$$|H_{Qj} \cap H_R| = DS_{RQj} \tag{23}$$

holds.

Now $S_{RQm} \leq S_{QmQj}$, and the relationship shown in Fig. 4 holds. The left panel of Fig. 4 shows the relationship between

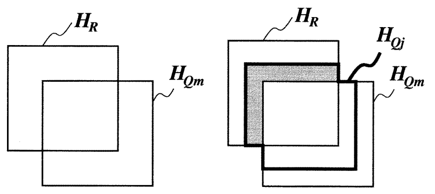


Fig. 4. Relations between histograms (1).

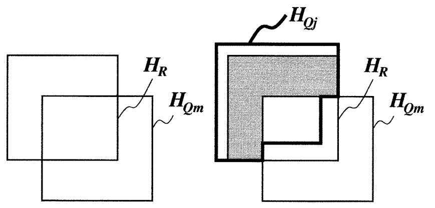


Fig. 5. Relations between histograms (2).

H_{Qm} and H_R, which was calculated by actual matching. Now we consider the case when $|H_{Qj} \cap H_R|$ is maximized with respect to the relationship between H_{Qj} and H_{Qm}. The conditions are 1) all elements of $\{H_{Qm} \cap H_R\}$ are included in H_{Qj} and 2) all elements of H_{Qj} except for the elements in $\{H_{Qm} \cap H_{Qj}\}$ (shadowed area in Fig. 4) contribute to increase the similarity to H_R, as illustrated in the thick line in the right panel of Fig. 4. Mathematically, the condition is written as

$$|H_{Qj} \cap H_R| \leq |H_{Qm} \cap H_R| + (D - |H_{Qm} \cap H_{Qj}|). \quad (24)$$

Equations (22) and (23) lead to

$$S_{RQj} \leq S_{RQm} + (1 - S_{QmQj}). \quad (25)$$

2) Case 2 $S_{RQm} \geq S_{QmQj}$: Supposing that the m-th query signal and j-th query signal are quite unsimilar will help in understanding this case.

Similar discussion to the Case 1 gives the relationship shown in Fig. 5. Thus, the condition for $|H_{Qj} \cap H_R|$ being maximized is 1) The all elements of $\{H_{Qj} \cap H_{Qm}\}$ are included in H_R and 2) all elements of H_R except for the elements in $\{H_{Qm} \cap H_R\}$ (shadowed area in Fig. 5) contribute to increase the similarity to H_{Qj}. That is

$$|H_{Qj} \cap H_R| \leq |H_{Qm} \cap H_{Qj}| + (D - |H_{Qm} \cap H_R|) \quad (26)$$

and, therefore

$$S_{RQj} \leq S_{QmQj} + (1 - S_{RQm}). \quad (27)$$

Equations (25) and (27) are written as (21).

This leads to a parallel search algorithm that comprises the following steps.

1) As a preprocessing, calculate similarities for all combinations of query signals and store them.
2) Locate the current position at the first frame of the stored signal (this is the start of the search process).
3) Choose the query signal whose skip position is closest to the current position, and update the current position to that skip position.
4) Match the chosen query signal with the stored signal at the current position to obtain a similarity value.

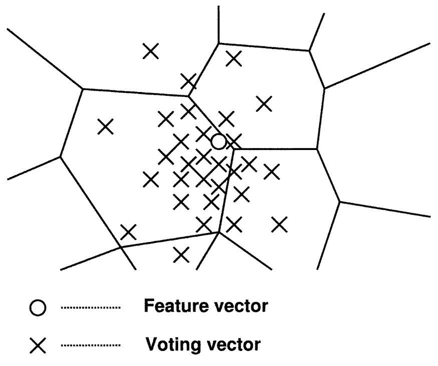


Fig. 6. Feature vectors voted with dithering.

5) Update the skip widths for all the query signals based on the obtained similarity value.
6) Return to Step 3.

In this scheme, the number of matching calculations in the search process is guaranteed to be less than or equal to the case where the query signals are separately matched with the stored signal. When D_j is not necessarily equal for all j, one can let D be a minimum of D_j, and then the above discussion is valid for the segments with the length D.

H. Feature Distortion Absorption

In realistic situations, signals can be affected by noise or distortion. For example, audio signals played back through loudspeakers and recorded with microphones, or extremely compressed to a low bit rate, can be considerably different from the original signals. To deal with such signal differences or distortion, here we propose a feature distortion absorption method.

In the method, the observed feature vector $\boldsymbol{f}$ is voted to a histogram as a probability density distribution $\boldsymbol{V}$ rather than a single deterministic vector, as illustrated in Fig. 6. Assuming that $\boldsymbol{V}$ can be sampled by a finite number of random vectors according to the distribution, feature vector voting is transformed into voting possibly-multiple dithered vectors $\boldsymbol{f}_i$ that are given by

$$\boldsymbol{f}_i = \boldsymbol{f} + \boldsymbol{v}_i \quad (i = 1, 2, \ldots, d) \quad (28)$$

where $\boldsymbol{v}_i$ is a random vector and d is the number of votes for a feature vector. We call this “probabilistic dither voting.”

The problem is then the estimation of the probability density distribution that maximizes the similarity value given by (10) when the original signals are identical. This is done by introducing a learning stage prior to the search.

In the learning stage, a pair of signals, one before and the other after the feature distortion, are prepared. The latter signal is obtained by processing the original signal actually by the target distortion model; for the learning of low-bit-rate compression, for example, the original signal is compressed at a low bit rate to obtain the distorted signal.

By comparing the features of those two signals on a frame-by-frame basis, the distortion can be learned. That is, the shift of feature vectors due to the distortion is statistically learned by subtracting the original feature vectors from the corresponding distorted feature vectors.

TABLE I
EXPERIMENTS

Exp.	Evaluation viewpoint	Feature
1	Search speed	Audio, Video
2	Search accuracy	Audio, Video
3	Search speed of time-order AND search	Audio
4	Search speed of parallel OR search	Audio
5	Feature distortion absorption effect	Audio

In our experimentation, the learning is done for each histogram bin (that is, each vector quantization code), and feature distortion is modeled by a combination of a shift and the Gaussian distribution in the feature space. That is, two parameters, the shifting vector and the standard deviation of the normal distribution for each vector quantization code, are to be learned.

The multiple kinds of distortion can be taken into account by preparing a query signal for each distortion type and searching them in parallel. In such a case, we can expect that the parallel search algorithm described in Section II-G is effective, because the query signals being searched in parallel originate in the same signal, and are likely to be similar to each other.

As the number of votes for representing the probability distribution in the feature space increases, the distribution is better represented, while the computational cost of voting is also increased. If the query signal is sufficiently long in duration, however, only one vote for each feature frame is sufficient, because the feature occurrence is accumulated in creating histograms. In our experimentation, only one vote is cast for each feature frame.

III. Experiments

Five types of experiments were conducted (Table I). Experiments 1 and 2 were conducted to evaluate the basic performance of the proposed method, and Experiments 3, 4, and 5 to evaluate the extensions. All the measurements were done on a PC (Pentium III 933 MHz, Linux).

A. Search Speed

In Experiment 1, a video recording of 48 h of TV broadcasting was the stored signal. The query signals were ten randomly chosen 15-s commercial messages, captured from other TV recordings. Thus, the task was to detect and locate specific commercial messages from two-days' worth of a TV recording. Each commercial message was played four to eight times during the stored 48 hours.

In the audio feature extraction, the audio track (VHS Hi-Fi format) of the recording was first digitized at 11.0 kHz sampling frequency and 16 bit quantization accuracy, and then analyzed by a seven-channel $(N = 7)$ 2nd-order IIR bandpass filter bank (the filter $Q = 10$). The filter center frequencies were equally spaced in a log frequency scale. The feature vectors as described in (2) were calculated every 10 ms $(M = 110)$.

TABLE II
SEARCH SPEED WHEN A 15-s QUERY SIGNAL WAS SEARCHED THROUGH 48-h STORED SIGNAL

Feature	CPU time*	♯ matches (reduction ratio)	
Audio	0.81 (69.46) s	34561	(1/501)
Video	0.17 (22.31) s	17793	(1/291)

* In (), the CPU time in case of exhaustive search (the case where w is fixed to 1) is given.

In the video feature extraction, the video signal (NTSC) was captured at 29.97 frames/s without compression. The capture size was 160 × 120 pixels. Each frame image was divided into 16 subimages $(W = 16)$ and feature vectors were calculated as in (6).

In discussing search speed, we distinguish preprocessing from searching. The preprocessing comprises processes that can be done before a query signal is given. These are 1) feature extraction for the stored signal and 2) vector quantization for the stored signal. After a query signal is given, 3) feature extraction for a query signal, 4) vector quantization for the query signal, and 5) matching between the query signal and each section of the stored signal, are performed. When we refer to search speed or search time, it specifically means the time needed for the fifth step, matching.

From the practical viewpoint, however, the time needed for processes other than the matching is also important. The CPU times needed for feature extraction for a 1-h signal were 53.2 s in the audio case and 150.2 s in the video case. The CPU times needed for vector quantization for a 1-h signal were 16.3 s for the audio feature and 62.7 s for the video feature. This means that feature extraction and vector quantization requires approximately 2% of the signal duration time in the audio case and 6% in the video case.

The search time depends on the signals to be matched, the detection threshold, and the number of histogram bins. Table II shows the CPU time averaged over ten query signals; here, the search threshold was fixed at 0.7 and the number of histogram bins was 512 for both the audio and the video features. It was verified that all the search results were correct, which means that there were neither redundant detections nor misses.

It is shown that the proposed method takes less than 1 s to search through the stored 48-h signal both in the audio case and in the video case. The number of matching calculations was reduced to approximately $1/500$ (audio) or $1/290$ (video) in comparison with the exhaustive search. The CPU time was also shortened by $1/85$ (audio) or $1/130$ (video).

Figs. 7 and 8 show a part of the corresponding similarity patterns for the audio search and the video search. In these figures the horizontal axis is time and the vertical axis is the similarity. The circles indicate the detected places whereas the horizontal dotted lines the detection threshold levels.

B. Search Accuracy

In Experiment 2, the search accuracy was evaluated using another TV recording. Firstly, a 60-m recording of TV broadcasting was captured twice; once as a source of query signals

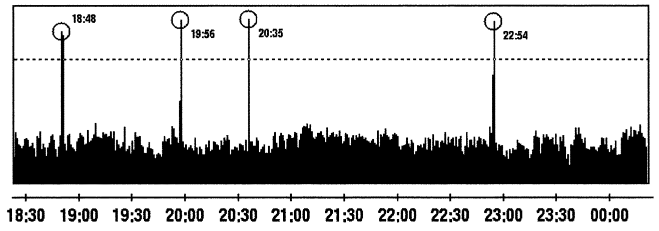


Fig. 7. Part of the similarity pattern (audio, $N_{\rm div} = 1$).

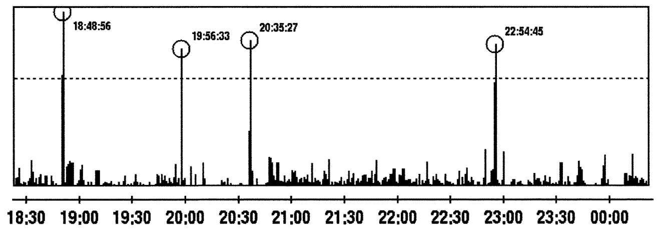


Fig. 8. Part of the similarity pattern (video, $N_{\rm div} = 1$).

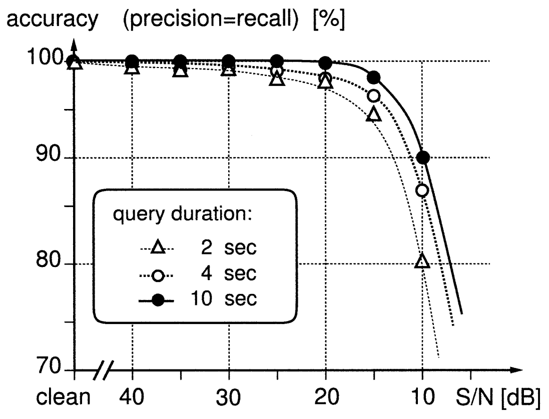


Fig. 9. Search accuracy (audio).

and then as a stored signal. The search was repeated 100 times; in each trial, a signal segment was randomly chosen from the first recording as a query signal, and the latter signal was searched through. To check for robustness, white Gaussian noise was added to the stored audio signal, and MPEG1 compression was performed on the stored video signal.

The results are shown in Fig. 9 and Table III. Here, the accuracy value is the precision rate (or the recall rate) when the precision rate equals the recall rate by changing the c value in (13). That is, for each noise condition, the c value was fixed to a certain value during the 100 repetitions, and the value to fix was adjusted so that the precision rate equaled the recall rate.

Fig. 9 shows the audio search accuracy. It is clear that if the query duration is longer than 10 s, there are no misses nor redundant detections down to the S/N ratio of 20 dB. It is also shown that even when the query duration is 2 s, the accuracy greater than 98% was achieved if the S/N ratio was greater than 25 dB. Table III shows the video search accuracy. If the query duration is longer than 4 s, there are no misses nor redundant detections even at the 100 kbps. It is also shown that even when the query duration is 2 s, better than 96% accuracy was achieved.

TABLE III
SEARCH ACCURACY (VIDEO)

Query duration [s]	Accuracy [%]		
	Original	1 Mbps	100 kbps
2	99.3	96.4	96.4
3	100	99.0	99.0
4	100	100	100
10	100	100	100

TABLE IV
NUMBER OF MATCHES AND SEARCH TIMES IN THE SEARCH INCORPORATING TIME ORDER WHEN A 15-s QUERY SIGNAL IS SEACHED THROUGH A 48-h STORED SIGNAL

Method	$N_{div} = 2$	
	♯ matches	CPU time
Individual	99289	1.57 s
Sequential	93124	1.15 s
Sequential-break	51449	0.88 s
Refinement	34836	0.83 s

(a)

Method	$N_{div} = 4$	
	♯ matches	CPU time
Individual	333630	3.41 s
Sequential	302960	2.27 s
Sequential-break	84326	0.93 s
Refinement	32361	0.81 s

(b)

C. Search Speed of Time-Order AND Search

In Experiment 3, the speed of the search incorporating time order was measured. The signals used were the same as in Experiment 1; the stored signal was a 48-h TV audio signal and the query signals were ten randomly chosen 15-s commercials. The windows were equally divided into two or four subwindows. The search threshold value was fixed at 0.7.

Table IV lists the experimental results with respect to the number of matches and the search times measured in CPU time. It is clear that the sequential method, sequential-break method, and refinement method are more efficient than the case where the divided three segments are separately searched. Note that it is guaranteed that the three methods yield the exactly same search results. The number of matches for the sequential method is less than for individual search, because of taking the maximum as in (15). The refinement method showed best performance as discussed in Section II-F.

Figs. 10 and 11 are graphical similarity patterns in this experiment. When compared with the Fig. 7, where the 15-s window is not divided into subwindows, the similarity margins in Figs. 10 and 11 are enlarged due to the time-order information incorporated.

D. Search Speed of Parallel OR Search

In Experiment 4, the search time in the parallel OR-search was measured.

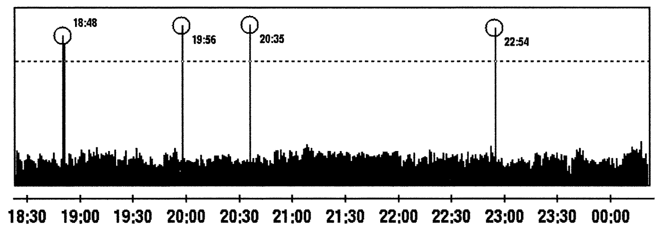


Fig. 10. Part of the similarity pattern (audio, $N_{\rm div} = 2$).

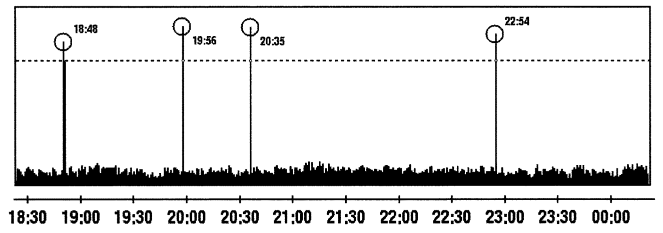


Fig. 11. Part of the similarity pattern (audio, $N_{\rm div} = 4$).

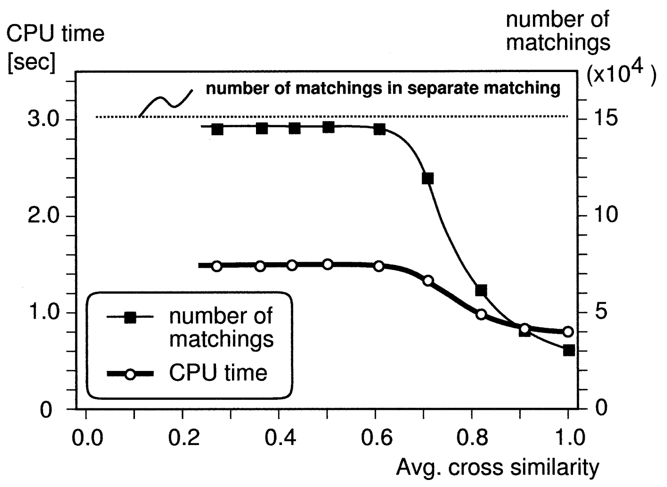


Fig. 12. Number of matches and search time for 5-parallel OR search when 15-s query signals are seached through a 48-h stored signal.

The stored signal was a 48-h audio signal captured from TV broadcasts. The query signals were five randomly chosen 15-s TV commercials.

As discussed in the previous section, the degree of efficiency in the proposed parallel search algorithm depends on the similarity between the query signals (hereafter, cross similarities). To control the cross similarities, the query signals were created by concatenating a common signal and uncommon signals; that is, by changing the proportion of the common part, the cross similarities were varied. The stored signal did not include the signals that were used in creating query signals. The search threshold value was fixed at 0.7.

The results are shown in Fig. 12, where the dotted line indicates the total number of matches when the five query signals were separately searched. The number of matches and the search times in Fig. 12 include those in the preprocessing, that is, matching among the query signals. As shown in Fig. 12, the proposed method can find the five query signals with little additional computational cost in comparison with one-query-signal search when the average cross similarity is close to 1. For example, when the average cross similarity was 0.83, the number of matches for five query signals was approximately twice, and the search time was approximately 1.24 times, as in the one-query-signal search.

In the region where the average cross similarity was less than 0.6, the number of matching calculations in the proposed method was close to that of the separate search. This means that, in terms of the number of matching calculations, there was little advantage for the proposed method in this region. The worst case in the proposed method is theoretically the same as the one-query-signal search, and in this region the situation was shown to be close to the worst case. As for CPU time, on the other hand, even when the cross similarity was as low as 0.36, the search time (1.44 s) was approximately 35% of the total of the separate search (4.05 s). This is because the computational cost for creating the histogram for stored signal was lower in the proposed parallel search than in the one-query-signal search; in our implementation, histograms were not created prior to the search, but created during the search. In this case, letting q stand for the cost of histogram creation in the OR-search, the cost of that in the one-query-signal search is nq in average, where n is the number of the query signals.

TABLE V
FEATURE DISTORTION ABSORPTION EFFECT

Microphone distance	Accuracy [%]		CPU time [ms]	
	On	Off	On	Off
10 cm	98.3	59.7	33	47
100 cm	97.3	60.8	36	42

On: feature dithering on, Off: feature dithering off

E. Feature Distortion Absorption Effect

In Experiment 5, the effect of the feature distortion absorption method was tested. The stored signal was a 60-min music signal. In an ordinary office room, the signal was played back through a loudspeaker and recorded with a microphone. The distance between the loudspeaker and the microphone was 10 cm or 100 cm. The query signals were 100 segments randomly chosen from the original music signal. The duration of the query signals was 10 s. The learning for probability density distribution for dithering was done using another 5-min music signal. That is, the 5-min signal was played back and recorded in the above-mentioned manner, and the difference between those recorded signals and the original 5-min signal was modeled using the shift vectors and standard deviation values for each VQ code. The learning was conducted for each microphone-to-loudspeaker distance. In the searching, the query signal feature vectors were first transformed by the proposed probabilistic dither voting method, and then compared with the stored signal ones. The definition of the accuracy was the same as in Experiment 2.

The results are listed in Table V. It is clearly shown that the proposed method improved the search accuracy. The average search time was also improved by the dithering; this is because the threshold margin was enlarged, and thus, the average skip width was increased, by the dithering.

F. Experiments Summary

In summary, it was shown that the proposed method, time-series active search, can detect and locate a specified 15-s audio or video signal from a 48-h audio or video signal within 1 s on a PC after the feature extraction and feature quantization (Experiment 1). This speed was achieved by our pruning method, which reduces the number of matches to $1/291$ (video) and $1/501$ (audio) in comparison with exhaustive matching, guaranteeing exactly the same search result as the exhaustive matching. Time-series active search was also shown to be accurate and reasonably robust with respect to audio noise addition and video compression (Experiment 2). Using the AND search algorithm, time order can be incorporated and thus the threshold margin can be enlarged with little additional computational cost; for example, in the refinement method, a four-division-AND search required approximately the same number of matchings as nondivided search (Experiment 3). Using the OR search algorithm, on the other hand, multiple query signals can be efficiently searched especially when they are similar to each other. When the average cross similarity was 0.83, the number of matches for five query signals was approximately twice, and the search time required was approximately 1.24 times, as the one-query-signal search (Experiment 4). In addition, the histogram modeling allows feature distortion to be considered in the feature voting process. When the probability density distribution of the feature distortion was learned prior to the search, the search accuracy improved from approximately 60% to 98% in a search using a signal recorded by a microphone, when a distance between the microphone and the loudspeaker was 10 cm (Experiment 5).

IV. Conclusion

This paper has proposed a search method that can quickly detect and locate a known query audio or video signal in a long-running audio or video signal. With the proposed algorithm, signals' feature vectors are quantized, and histograms are produced; the similarity between the query signal and stored signal is estimated by matching their respective histograms. In so doing, unnecessary matching calculations are avoided by exploiting the algebraic properties of histograms. The basic algorithm has been extended to incorporate the time order of feature vectors, and to enable efficient search for multiple query signals in a parallel way. A method to deal with feature distortions has also been presented.

The experiments showed that the proposed method can correctly detect and locate a 15-s commercial in a 48-h recording of TV broadcasting within 1 s, once feature vectors are calculated. It was also shown that the method is robust with respect to the white Gaussian noise addition to the audio signal down to the S/N ratio of 20 dB, and MPEG-1 video compression of the video signal at 100 kbps, when the query signal was 10 s.

The applications of the proposed method will include TV commercial monitoring systems and music information retrieval systems from a broadcasted music fragment. We also anticipate that it is applicable to Web search engines that are searchable by an audio or video query, while most multimedia search engines on the World Wide Web have been based on symbolic information such as automatically recognized speech contents [27].

Future research includes the development of a method to deal with even worse-quality audio or video signals, such as an audio signal transmitted over telephone lines. Such a method is expected to further broaden the application domain of the similarity-based signal searching technique.

Appendix A
Proof for (15)

When we let $D_1, D_2, \ldots, D_N$ denote the length of divided subwindows, the number of votes in the histogram created over the original window and that in the histograms created over the divided subwindows must be the same. This is expressed as

$$\mathrm{DS}_{\mathrm{org}} = D_1 S_1 + D_2 S_2 + \cdots + D_N S_N. \tag{29}$$

Now each subwindow is adjacent, and therefore, the following inequality holds:

$$S_{\mathrm{org}} \geq \frac{D_1 S_{\min} + D_2 S_{\min} + \cdots + D_N S_{\min}}{D_1 + D_2 + \cdots + D_N} \tag{30}$$

where $(S_{\min} = \min_j (S_j))$. This is equivalent to

$$S_{\mathrm{org}} \geq S_{\min} \tag{31}$$

and this leads to (16). (q.e.d.)

Acknowledgment

The authors thank Prof. K. Ishii and Dr. N. Sugamura for their help and encouragement.

References

[1] J. K. Wu, A. D. Narasimhalu, B. M. Mehtre, C. P. Lam, and Y. J. Gao, "CORE: A content-based retrieval engine for multimedia information systems," *ACM Multimedia Syst.*, vol. 3, no. 1, pp. 25–41, 1995.

[2] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Yonkani, J. Hafner, D. Lee, D. Petkovic, D. Stede, and P. Yanker, "Query by image and video content: The QBIC system," *IEEE Computer*, vol. 28, no. 9, pp. 23–32, 1995.

[3] H. D. Wactlar, "Informedia—Search and summarization in the video medium," presented at the Imagina 2000, 2000.

[4] R. Brunelli and O. Mich, "Image retrieval by examples," *IEEE Trans. Multimedia*, vol. 2, pp. 164–171, Sept. 2000.

[5] S. Pfeiffer, S. Fischer, and W. Effelsberg, "Automatic audio content analysis," in *Proc. ACM Multimedia*, 1996, pp. 21–30.

[6] E. Wold, T. Blum, D. Keislar, and J. Wheaton, "Content-based classification, search, and retrieval of audio," *IEEE Multimedia*, vol. 3, no. 3, pp. 27–36, 1996.

[7] S. J. Young, M. G. Brown, J. T. Foote, G. J. F. Jones, and K. S. Jones, "Acoustic indexing for multimedia retrieval and browsing," in *Proc. ICASSP'97*, vol. 1, 1997, pp. 199–202.

[8] J. Foote, "An overview of audio information retrieval," *Multimedia Syst.*, vol. 7, no. 1, pp. 2–11, 1999.

[9] T. Zhang and J. C. Kuo, "Hierarchical system for content-based audio classification and retrieval," *Proc. SPIE*, vol. 3527, pp. 398–409, 1998.

[10] J. C. Hancock and P. A. Wintz, *Signal Detection Theory*. New York: McGraw-Hill, 1966.

[11] B. Kedem, "Spectral analysis and discrimination by zero-crossings," *Proc. IEEE*, vol. 74, pp. 1477–1493, Nov. 1986.

[12] J. Saunders, "Real-time discrimination of broadcast speech/music," in *Proc. ICASSP'96*, vol. 2, 1996, pp. 993–996.

[13] B. S. Atal and M. R. Schroeder, "Predictive coding of speech signals," *Proc. IEEE Conf. Communication and Processing*, pp. 360–361, 1967.

[14] L. R. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[15] J. Foote, "Content-based retrieval of music and audio," *Proc. SPIE*, vol. 3229, pp. 138–147, 1997.

[16] B. Logan, "Mel frequency cepstral coefficients for music modeling," presented at the Int. Symp. Music Information Retrieval, 2000.

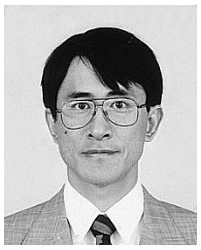
[17] M. J. Swain and D. H. Ballard, “Color indexing,” *Int. J. Comput. Vis.*, vol. 7, no. 1, pp. 11–32, 1991.
[18] Y. Gong, C. Chuan, and G. Xizoyi, “Image indexing and retrieval based on color histograms,” *Multimedia Tools Applicat.*, vol. 2, pp. 133–156, 1996.
[19] X. S. Zhou and T. S. Huang, “Edge/structural features for content based image retrieval,” *Pattern Recognit. Lett.*, vol. 22, no. 5, pp. 457–468, 2001.
[20] M. Shenier and M. Abdel-Mottaleb, “Exploiting the JPEG compression scheme for image retrieval,” *IEEE Trans. Pattern Anal. MAchine Intell.*, vol. 18, pp. 849–853, Aug. 1996.
[21] W. Ma and B. Manjunath, “Texture-based pattern retrieval from image databases,” *Multimedia Tools Applicat.*, vol. 2, pp. 35–51, 1996.
[22] G. Smith, H. Murase, and K. Kashino, “Quick audio retrieval using active search,” in *Proc. ICASSP’98*, vol. 6, 1998, pp. 3777–3780.
[23] K. Kashino, G. Smith, and H. Murase, “Time-series active search for quick retrieval of audio and video,” in *Proc. ICASSP’99*, vol. 6, Mar. 1999, pp. 2993–2996.
[24] V. V. Vinod and H. Murase, “Focused color intersection with efficient searching for object extraction,” *Pattern Recognit.*, vol. 30, no. 10, pp. 1787–1797, 1997.
[25] R. S. Boyer and J. S. Moore, “A fast string matching altorithm,” *Commun. ACM*, vol. 20, no. 10, pp. 702–772, 1977.
[26] J. H. Knuth, J. H. Morris, and V. R. Pratt, “Fast pattern matching in strings,” *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, 1977.
[27] J. V. Thong, P. J. Moreno, B. Logan, B. Fidler, K. Maffey, and M. Moores, “SPEECHBOT: An Experimental Speech-Based Search Engine for Multimedia Content in the Web,” Compaq Cambridge Res. Lab. Tech. Rep., CRL 2001/06, 2001.

Kunio Kashino (S’89–M’95) received the B.Eng. degree in electronic engineering and the M.Eng. degree in electrical engineering in 1990 and 1992, respectively, and the Ph.D. degree in electrical engineering in 1995, all from the University of Tokyo, Japan.

In 1995, he joined Nippon Telegraph and Telephone Corporation, where he is currently Senior Research Scientist. In 2002, he was a Visiting Scholar at University of Cambridge, U.K. He has been working on multimedia information retrieval, and music scene analysis. His research interests include acoustic signal processing, Bayesian information integration for real-world recognition, and sound source separation.

Takayuki Kurozumi received the B.S. degree in physics from the Tokyo Metropolitan University, Japan, in 1997, and the M.S. degree in information science from the Japan Advanced Institute of Science and Technology, Ishikawa, in 1999.

In 1999, he joined Nippon Telegraph and Telephone Corporation, Atsugi, Japan. His research interests include pattern recognition, imageprocessing, and multimedia information retrieval.



Hiroshi Murase (M’87–SM’00) received the B.Eng., M.Eng., and Ph.D. degrees in electrical engineering from Nagoya University, Japan, in 1978, 1980, and 1987, respectively.

In 1980, he joined Nippon Telegraph and Telephone Corporation (NTT), Atsugi, Japan. From 1992 to 1993, he was a Visiting Research Scientist at Columbia University, New York. From 2001 to 2003, he was Executive Manager of NTT Communication Science Laboratories. He is currently Professor of information science at Nagoya University. His research interests include computer vision, pattern recognition, and multimedia information retrieval.

Dr. Murase was awarded the IEEE CVPR best paper award in 1994 and the IEEE ICRA best video award in 1996.