A Practical Query-By-Humming System for a Large Music Database

Naoko Kosugi, Yuichi Nishihara, Tetsuo Sakata, Masashi Yamamuro, and Kazuhiko Kushima
nao@isl.ntt.co.jp
NTT Laboratories, 1-1, Hikarinooka, Yokosuka-shi, Kanagawa, 239-0847, Japan

Abstract

A music retrieval system that accepts hummed tunes as queries is described in this paper. This system uses similarity retrieval because a hummed tune may contain errors. The retrieval result is a list of song names ranked according to the closeness of the match. Our ultimate goal is that the correct song should be first on the list. This means that eventually our system's similarity retrieval should allow for only one correct answer.

The most significant improvement our system has over general query-by-humming systems is that all processing of musical information is done based on beats instead of notes. This type of query processing is robust against queries generated from erroneous input. In addition, acoustic information is transcribed and converted into relative intervals and is used for making feature vectors. This increases the resolution of the retrieval system compared with other general systems, which use only pitch direction information.

The database currently holds over 10,000 songs, and the retrieval time is at most one second. This level of performance is mainly achieved through the use of indices for retrieval. In this paper, we also report on the results of music analyses of the songs in the database. Based on these results, new technologies for improving retrieval accuracy, such as partial feature vectors and or'ed retrieval among multiple search keys, are proposed. The effectiveness of these technologies is evaluated quantitatively, and it is found that the retrieval accuracy increases by more than 20% compared with the previous system [9]. Practical user interfaces for the system are also described.

1 Introduction

The use of multimedia data has spread throughout the world with the availability of high-performance, low-cost personal computers, and this has led to a need for accurate, efficient retrieval methods for large multimedia databases. General retrieval systems accept only key words as queries. However, many users experience difficulty in formulating a query in words when they want to retrieve something from a multimedia database. Content-based retrieval is seen as a solution to this problem [4, 7, 10, 15, 21].

Content-based retrieval accepts data-type queries, e.g., drawings for an image database and sung tunes for a music database. As content-based retrieval enables users to represent what they want directly, it makes the retrieval system easier to use.

The authors have developed a data-type retrieval system called *ExSight* [17, 20] for an image database. We are currently studying a content-based music retrieval system for a music database, which we call *SoundCompass* [8].

In a content-based retrieval system, it is possible for the users to make a query key from erroneous input. This makes it difficult to obtain an accurate result through a matching with the key. Thus, similarity retrieval is useful because retrieval results contain multiple alternatives ranked according to the closeness of the match. Users can select what they want from the list even though they cannot make an accurate query.

The authors have developed the *HyperMatch* engine [2], a high-speed similarity retrieval engine based on distance measurement employing multiple high-dimensional feature vectors. Each vector retains feature information, and similarity is represented as a weighted combination of the search results from individual feature vector spaces.

This paper describes our music retrieval system that accepts a hummed tune as a query. Our ultimate goal for this system is to have it retrieve the correct song for the hummed tune as the first item in the retrieved results list. This means that our system's similarity retrieval eventually should allow for only one correct answer.

When matching is done between songs in the database and the hummed tunes, we have to consider the following problems:

- We do not know which segment of the song will be hummed a priori.
- The song may be sung out of tune.
- The tempo may be wrong.
- Users may hum the wrong note, because their memory of the song is incorrect.
- The transcription may contain a mistake.

We have already described techniques for handling these problems in a query-by-humming system [9, 13]. The previous system was demonstrated at ACM Multimedia '99. Since then, the database has been enlarged, for eventual practical use of the retrieval system. As a result, the problems became more complex, and the technologies for improving performance and accuracy of the retrieval system became more challenging. This paper describes in detail the technologies for improving the performance and accuracy of the query-by-humming system for a large music database. We first report on the results of music analyses on all the songs in the database. Then, these results are incorporated in new technologies for database construction and hummed tune processing. The effectiveness of these technologies is evaluated quantitatively. Practical user interfaces for the system are also described.

2 Related Work

We'll begin by reviewing some of the research on content-based retrieval for audio databases [4]. With respect to "for what it searches", we can find research on identifying a specific rhythm or sound of an instrument [6, 12], and research on identifying a song from a melody segment given as a query [7, 9, 11]. With respect to "by what means it searches", we can find research on search by sound file in ".au" format [6], search by segment of a MIDI file [18], search by a string representing pitch direction [1], and search by hummed tune [7, 9, 11, 14]. Content-based retrieval using similarity requires that one clarify "for what" and "by what means" it searches.

String matching is the most often used method of melody and song retrieval from a music database [1, 7, 11, 14, 18]. This is because music can be represented by a sequence of notes, and this sequence can be converted into a string of letters. Pitch direction is often used [1, 7, 11, 14] instead of the pitch itself to construct the string. There are two reasons for this choice: one is that it ignores the key difference between a music in a database and an inputted tune, and the other is that for a hummed tune as input, it realizes robust retrieval even though the hummed tune may have variations in tone and tempo. However, for a large database, retrievals using only this information can not provide a high enough resolution. Thus, optional retrievals using rhythm information and accurate intervals have been provided as an advanced search feature in [11], and a more detailed specification describing pitch direction according to pitch differences has been proposed in [1].

Errors in input data also have to be considered in a searching method [7]. Errors in a hummed tune may include not only variations in tone and tempo, but also fragmentations, insertions and deletions of notes [11]. Dynamic Programming (DP) can be applied to string matching, which allows errors in inputs. Similarity is calculated by *edit distance* in DP matching. In edit distance, costs for insertions, deletions, and the substitutions of specific pairs of characters are defined. The smaller the cost, the more the two strings are regarded as being similar to each other. DP matching usually has a high retrieval accuracy [11]. However, its retrieval time depends on the size of the database because the search is done by brute force [7]. Thus, a state matching algorithm [19] has been used in [11] to improve the speed of retrieval. However, the authors of [11] also reported that the similarity cannot be sufficiently specified, after inclusion of the state matching. The authors in [18] investigate how to best represent the music data and how to calculate the similarity between two pieces of music by examining different combinations of measures (edit distance, n-gram measures, and so on) and representations (contour, exact interval, and so on).

Besides string matching, retrievals that use high-dimensional feature vectors in which similarity is represented by the distance between vectors have been proposed [6, 9]. Euclidean distance or cosine distance is used as the measure. In this case, since indices are used for retrieval, the larger the size of the database, the higher the retrieval performance.

To transcribe the inputted tune as accurately as possible, the user is asked to utter a specific syllable rather than actually hum. Thus, "humming" here does not mean the ordinary humming but instead "singing with only the syllable *ta* or *da*" [9, 11].

Many music databases for melody/song retrieval hold music data in MIDI format. The number of songs in the database usually ranges from a few hundred [7] to over 10,000 [18]. Notice that we cannot know the exact size of the database only from the number of songs it contains because its size actually depends on the length of each song and the number of the notes in each song.

3 Music Analysis

In this section, the results of an analysis of the over 10,000 songs in our database are presented. These results gave us many important clues on MIDI data processing to make a database and process hummed tunes.

3.1 MIDI Data Information

We currently have 10,069 songs stored in MIDI format. The division of all the songs is 480. Thus, the length of a quarter-note is 480 tick times. An interval is represented by a whole number between 0 and 127. An interval of 1 tone in MIDI is the same as a half step in the normal musical scale.

3.2 Note Distribution

The 10,069 songs include many musical genres. Some are short, simple songs such as nursery songs and folk songs, and some are long, complex pop and rock songs. The total number of notes is 3,676,773, and the mean number of notes per song is 365.16, which is about seven times greater than that of the average folk song that can be downloaded via the Internet. Figure 1 shows the distribution of the total number of notes in each song.

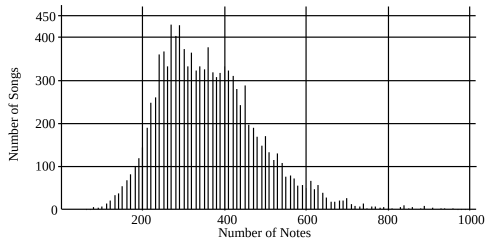


Figure 1: Distribution of the total number of notes per song.

We also investigated the distribution of the length of notes in the songs. All songs specify the length of a quarter-note as 480 in tick time, so Figure 2 shows that eighth-notes are overwhelmingly dominant. Note that the slower the tempo is, the greater the number of notes there are whose length is 120, and the faster the tempo, the greater the number of notes whose length is 480. This result implies that there is not a wide variation in the length of the notes that human beings can sing.

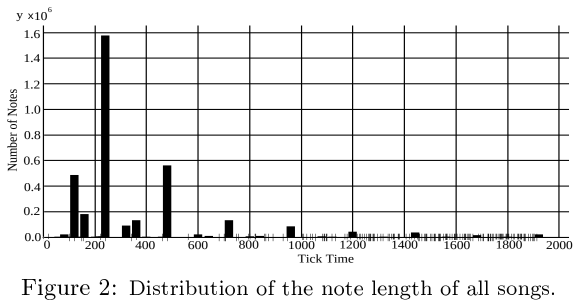


Figure 2: Distribution of the note length of all songs.

3.3 Repetition Structure of Music

Music is generally self-similar [5], and many songs have a repetitive structure consisting of two or three similar verses. Figure 3 shows the ratio of songs that contain identical parts within themselves, with the song chopped up into overlapping parts every four beats with a length of 16 beats. The figure shows that about 50% or 60% of the parts of the songs are exactly the same. This implies that many songs may

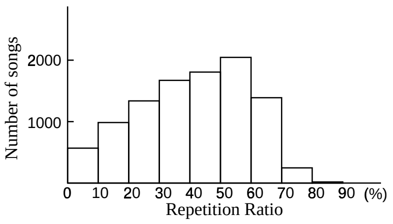


Figure 3: Ratio of songs that have a x% identical portion.

have a second verse. Most of the songs that have a high ratio of repetition (i.e., more than 80%) are highly rhythmical songs, such as dance music, and very easily learned songs, such as cheering songs for sports events.

3.4 Tempo Analysis

When people sing, they themselves decide what tempo to maintain, and it may well happen that the tempo they choose is not the same as that of the song in the database.

Figure 4 shows the distribution of the tempo of all the songs. We found that the faster a song is, the more people will use a tempo that is only half the correct one. This point is also relevant to the discussion in Section 3.2.

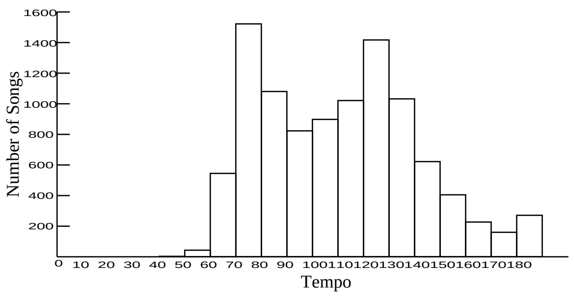


Figure 4: Distribution of tempo of all songs.

Why is that people sing a faster song at a tempo that is half the correct one? Let us consider two songs, A and B (Figure 5 – 6). The tempo of song A is 180 and that

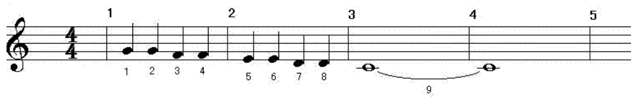


Figure 5: Segment of song A. The tempo is 180.

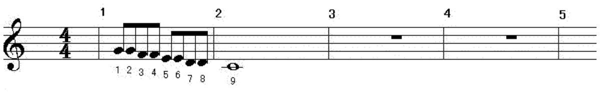


Figure 6: Segment of song B. The tempo is 90.

of song B is 90. These songs may seem to be very different from each other, but they are performed in the same way with respect to the note length. This is because an eighth-note in tempo 90 is performed with the same length as that of a quarter-note in tempo 180.

3.5 Interval Distribution

Generally, there is not a wide range of difference in the pitch between successive notes in many songs. Figure 7 shows the distribution of the pitch difference between successive notes. In MIDI data, a difference of a half step is represented with

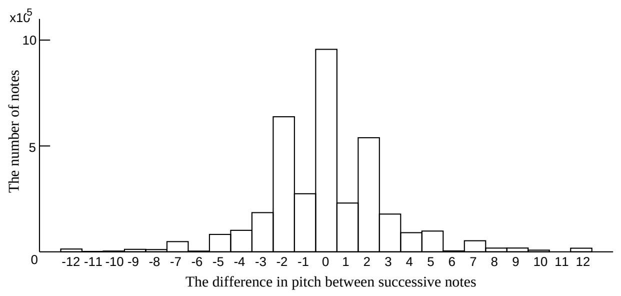


Figure 7: Distribution of the pitch difference between successive notes.

the numeral 1, so +12 represents a tone an octave higher, and -12 represents a tone an octave lower.

As shown in Figure 7, the difference in successive notes is concentrated in 0, -2, and +2. This means that most of the notes are either the same or within one step of each other. Another interesting point is that the difference for six half steps is much less than that for 5 or 7. This interval is called "tritone" and is the most dissonant interval.

4 Music Retrieval System

This section describes how we generated a query-by-humming system based on the analysis in Section 3.

4.1 Database Construction

We chose MIDI music data for the database elements. A lot of MIDI music data, including the latest pop hits, can be easily obtained in Japan because of the popularity of karaoke. MIDI music data can be regarded as musical indicators for each channel. Most karaoke recordings store the melody data on one channel. This allows the melody to be easily recognized.

To construct a music database, the melody data have to be extracted first. Currently, only melodies are used for matching because most people remember a song by its melody.

Then, the song is analyzed according to its tempo. As described in Section 3.4, for faster songs users tend to choose a tempo that is half the correct one. Thus, for fast tempos, two copies of the song data are made: one at the correct tempo and another at half that tempo. As a result, users can retrieve the same song by humming at either tempo.

Next, all the chords are deleted. A chord here means "notes that partially overlap each other in time". Chords that are found in accompaniments are not included because the melody data are extracted by the process described above. Most of the chords are made from a succession of notes that partially overlap (overlaps are included as MIDI performance effects by the manufacturer). However, most people cannot sing two tones at the same time when they hum. Thus, chord deletion is done to enable the melody data information to be coordinated with that from the hummed tunes. The note in a chord that occurred earliest is deleted. For example, if note B begins to sound while note A is sounding, the note A part of the chord is deleted (see Figure 8). If two notes start to sound simultaneously, the higher note

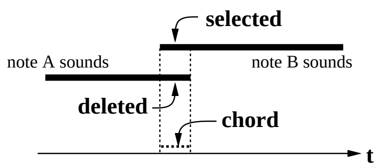


Figure 8: Chord deletion.

is kept.

In addition, double-pitch errors are corrected. This type of error exists in the transcriptions of the hummed tunes. However, we require that both the hummed tunes and melody data be processed in a similar fashion. So this correction step is performed on both data types. This correction method is described in Section 4.2.2.

The melody data is then chopped up into melody pieces of a constant length by the sliding window method[9]. The pieces are defined as *subdata*. Each subdata is given redundancy with respect to its predecessor and successor by letting the length of the slide be shorter than the length of the window. Chopping up the melody data into subdata of exactly the same length is necessary for correct similarity calculation, while chopping up the melody data into overlapping subdata allows a user to select whichever part of a song he or she wishes to hum.

Next, feature information is extracted from each subdatum. Multiple features are extracted and converted into high-dimensional feature vectors. The features are described in detail in Section 5.

Then, subdata which make the same feature vectors are deleted (duplication deletion). To put it concretely, subdata which make the same vectors for all features are detected for each song, and only one subdatum among them is kept for the database.

Finally, the feature vectors are loaded to the system's memory. The vectors are individually indexed according to feature and entered in the database's server. Indices are made based on an improved version of VAMSplit R-tree [2], which provides faster retrieval.

4.2 Hummed Tune Processing

This section describes the method of generating the query key from the hummed tune. The process is similar to the database construction process. However, before making a query, some corrections have to be made to the information obtained from the hummed tune because these tunes may contain various errors.

4.2.1 How to Hum?

The hummed tune is recorded through a microphone and is converted into MIDI format by commercial composition software [16]. The user is required to clearly hum the song notes using only the syllable "ta" . This is done so that the hummed tune can be transcribed as accurately as possible. Note that tunes hummed with the syllables "da" and "la" produce transcription errors.

Moreover, users are required to hum following the beats of a metronome. This is done to enable people to hum in constant tempo; casual singers usually have difficulty in keeping a constant tempo without any guidance. Moreover, we have to know the tempo information that the singer followed because we use beat-based processing [13]. Of course, the user may adjust the metronome to the desired tempo.

4.2.2 Modifications and Making the Search Key

Noise is deleted from the hummed tune after the tune has been converted into MIDI format. The noise deletion method is described in detail in Section 8.1.

Double-pitch errors are then corrected. In a double-pitch error, a note is transcribed at an octave lower in pitch than the correct one. Errors of this type arise in the pitch detection part of the transcription software [16]. The fact that sound waves of a relatively long wave length are weighted in processing to establish priority when the pitch is extracted by the auto-correlation function may cause these errors [3]. The method of correcting double-pitch errors is as follows: if the difference in pitch between a note and its successor falls within a certain range, raise that note one octave.

Next, the hummed tune is chopped up into hummed pieces of the same window size and sliding length as that of the subdata. During the chopping, a waver correction is also performed. The waver correction compensates for the subtle variation in tempo of a hummed tune. This correction adjusts the chop point to be at the start of the note that is nearest the chop point.

Finally, the same kinds of features as those extracted from subdata are extracted from each hummed piece. These are also converted to feature vectors and used for queries.

4.3 Similarity Measurement, Similarity Retrieval

The similarity retrieval finds vectors in each feature's vector space that are close to the vectors generated by the hummed tune. Dissimilarity is calculated for each subdatum by using the weighted linear summation of the distance between the vectors. Euclidean distance is used for the measurement. The shorter the distance to the subdata is, the higher the ranking of the retrieval results. The final retrieval result is a weighted combination of the search results from individual feature vector spaces and is presented as a ranked list of songs.

Figure 9 presents an image of the content of this section.

5 Feature Vectors

This section explains how feature vectors are generated from each subdatum and each hummed piece. The parameters used for the generation of the vectors are based on the results in Section 3. Section 7 describes these parameters in detail. The feature vectors are chosen so that a correct answer could be obtained from as little feature information as possible.

5.1 Tone Transition Feature Vector

People can usually identify a song they know by hearing a only part of it, even if the hummed tune is somewhat out of tempo or tune. They also tend to think that timewise transitions in the pitch and duration of tones are the most important feature of music. Thus, it is useful to define a feature vector that can represent a timewise transition, hereafter called a *tone transition feature vector* [9].

A tone transition feature vector is a sequence of tones in which each tone sounds within successive constant beats (resolution beat "r"). This is why we describe our query-by-humming system as being beat-based and not note-based. The value of each dimension of a feature vector is a typical tone in a succession of resolution beats. So, problems such as note fragmentation and note insertion [11] have no affect on matching. A tone is considered to be representative, if it is the longest tone in the resolution beat. Thus, a short note that is incorrectly transcribed (due to variation in tone) does not affect the generation of the feature vector.

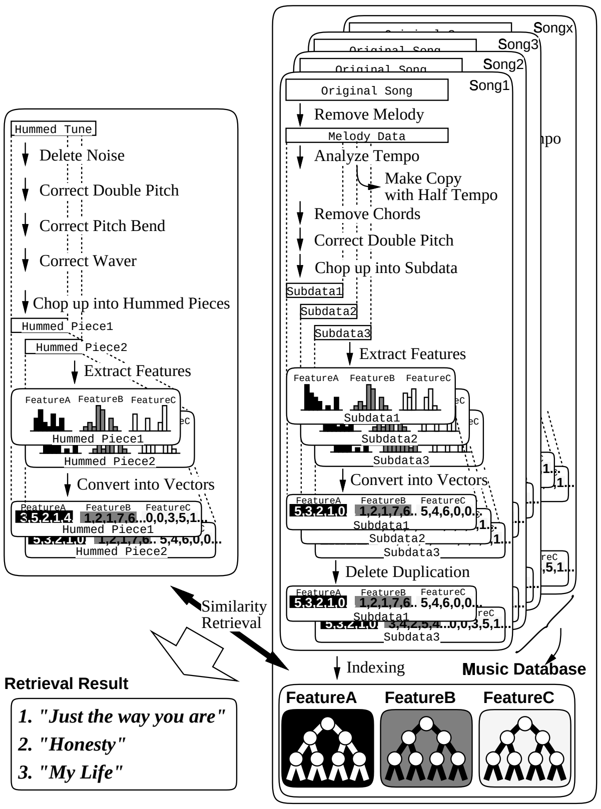


Figure 9: Steps in construction of the music database and processing the hummed tune.

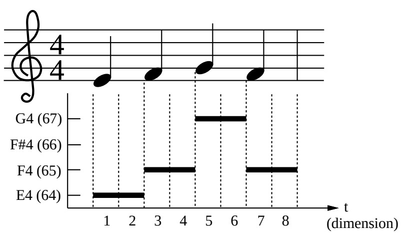


Figure 10: Image of making a tone transition feature vector.

Figure 10 shows how to make a tone transition feature vector from a tune with the resolution of an eighth-note. E4, F4, and G4 are MIDI codes, and the numbers to their right are the MIDI note number. Thus, an eight-dimensional feature vector, (64, 64, 65, 65, 67, 67, 65, 65), can be made from this four-beat tune. In addition, a vector that does not depend on the key difference between a hummed tune and a melody data in the database can be made if each value in the vector represents pitch relative to a certain tone (called *base tone*). For example, if the most common tone in the tune in Figure 10 is selected as a base tone (65) and let it be "0", the final feature vector of eight dimensions is (-1, -1, 0, 0, 2, 2, 0, 0).

5.2 Partial Tone Transition Feature Vector

As described in Section 4.1 and Section 4.2.2, both melody data in the database and hummed tunes are chopped up into subdata and hummed pieces respectively of the same length in every slide length by the sliding window method. Since we cannot know which portion of a song a user will sing a priori, this redundant chopping generally makes it possible for a user to hum any part of a song (usually users tend to hum the beginnings of a song or its phrases) in order to retrieve the song. However, this of itself is not enough, because people do not always remember a song by its beginning or its phrases. Thus, the beginning of the hummed tune is not always the beginning of the song or the beginning of a phrase. Furthermore, the beginnings of subdata are not always the beginnings of phrases or bridges. In particular, in many current pop songs, the start beat of songs is inconsistent with the start beats of the phrases and the bridges.

Let us consider the tune shown in Figure 11. The

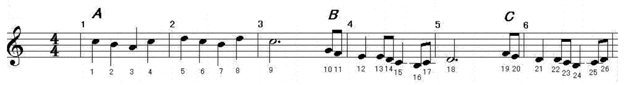


Figure 11: A tune in which the start beat of the beginning part (A) is different from the start beat of the bridge part (B).

bridge begins at the fourth beat (B) of the third bar, and the start of the song is at the first beat (A) of the first bar. This means that this tune is an example of an anacrusis with respect to the bridge. All of the starts of the subdata made from this tune are at the first beat of each bar if the slide length of the sliding window method for chopping is four beats. Thus, when the user hums the bridge, there is no subdata in the database whose start is consistent with the start of the hummed tune. That is to say, between the starts of the subdata and the starts of the hummed pieces, there can actually be a gap that is as much as half the slide length when melodies and hummed tunes are chopped up automatically in the sliding window method.

We therefore propose another tone transition feature vector to solve this problem. Of course, the length of the slide can be reduced; however, this increases the size of the database. Thus, we invented a partial tone transition feature vector to solve this problem without increasing the database. In a partial tone transition feature vector, the starts of the vectors are inconsistent with the starts of the subdata and hummed pieces.

The vector is made by picking out the values of $(w - s) \times r + 1$ dimensions from the values of $w \times r$ dimensions, where r is the resolution of the feature vector, w is the window size, and s is the slide length. The start of the vector is selected according to a certain rule from the values within s beats. For example, suppose that the tune in Figure 10 is the four beats of the beginning of a certain subdata. The value of s is four here. If we let the place where the highest tone initially appears be the beginning of the vector, the fifth place of G4 is the start of the vector and a $(w - 4) \times 2 + 1$-dimensional vector is generated.

Using the same parameters, let us make another vector from the tune in Figure 11. Vectors whose starts are consistent with the starts of the subdata are generated up to the fourth vector; however, the start of the fifth vector is different from the start of the subdata because the start of the fifth vector is the place marked C in the figure. Thus, if a user hums the bridge part including the fifth bar, the retrieval system can find a vector whose start is consistent with the start of a vector made from the hummed tune in the database.

5.3 Tone Distribution Feature Vector

The partial tone transition vector proposed in Section 5.2 can solve the inconsistency of the starts of the vectors generated by both the subdata and the hummed pieces. However, some of the starts of the vectors in the database will still be inconsistent with the starts of the vectors generated from the hummed tune. Furthermore, feature vectors that enable us to roughly grasp the characteristics of the hummed tune are needed because there may be variations in tempo and tone in the hummed tune. Thus, we have introduced another type of feature vector that represents the distribution of tones of each subdatum [9]. This type of vector can represent the total number of occurrences of each tone and represent the total number of beats that each tone sounds. Figure 12 depicts a histogram which shows the total number of beats that each tone sounds. A four-dimensional tone distribution feature vector (2,4,0,2) can be made, if we define 64 as the lowest tone and 67 as the highest.

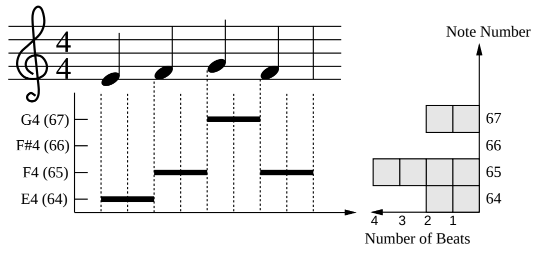


Figure 12: Image of making a tone distribution feature vector.

6 Or'ed Retrieval among Search Keys

As described in Section 4.2.2, hummed tunes are also chopped up into hummed pieces of the same length with the sliding window method. Thus, multiple search keys can be generated from a hummed tune, if the user hums for a time period longer than that specified by the window size. Since most tunes are not hummed well at the very beginning or at the very end, we made a search key for the middle part of a hummed tune by requiring users to hum for a period of time a little longer than the window size [8]. However, some people hum in tune only at the beginning, while others become more in tune the longer they hum. Moreover, some parts of songs are easy to hum, while others are more difficult.

We propose generating multiple search keys from the entire hummed tune, if a user hums for a period of time that is longer than the window size, and retrieving songs using all keys.

We also propose making the final results by or'ing each result of search keys.

Let us illustrate this with the following example. Suppose that a user hums a little longer than the time period specified by the window size, and as a result, three search keys, h_1, h_2, and h_3, are generated. There are four songs, A,B,C, and D, in the database. Suppose that these three keys retrieve the results shown in Table 1. The values in parenthesis are dissimilarities calculated from each subdata. The retrieval result can be made as follows. First, the results are gathered for each song, then we let the minimum dissimilarity within the results of each song be the dissimilarity of the song. The result is presented as a ranked list of songs in ascending order of dissimilarity. This method makes use of the fact that a vector made from a subdatum that correctly matches a vector made from a hummed piece must have minimum dissimilarity. As a result, we can obtain the final result, A(0.3) D(0.9) B(1.0) C(1.2), by processing the results in Table 1, and decide the correct answer is song A. This method is called *or'ed retrieval among search keys.*

Table 1: Retrieval Results of Each Hummed Key of a Hummed Tune.

key	result(dissimilarity)			
h_1	D(0.9)	B(1.5)	C(1.8)	A(5.8)
h_2	A(0.3)	B(1.2)	C(2.0)	D(5.9)
h_3	B(1.0)	C(1.2)	D(1.5)	A(6.0)

7 Evaluation

In this section, we present retrieval accuracy evaluation results for the database (described in Section 4.1) and for the feature vectors (described in Section 5), both of which are generated based on the analysis results given in Section 3. We also present another retrieval accuracy evaluation for the method proposed in Section 6 to determine the final result and a performance evaluation of the use of indices.

7.1 System Parameters

Experiments for the accuracy evaluation and the performance evaluation were done with the 10,069-song database.

A total of 258 tunes were hummed by 25 people (21 males and four females). Of the 258 tunes, we selected 186 tunes which are transcribed into MIDI representation and were recognizable as a part of a melody when they were heard. All the 186 tunes are a part of a song which is registered in our song database. Most of the 25 subjects were casual singers, but some of them were members of a choral group.

A window size of 16 beats was chosen for chopping up melody data and hummed tunes. This is because most of the songs in the database are in 4/4 time, and in that case, the length of 16 beats corresponds to four bars. The length of four bars was found to be the minimum length of a phrase or a unit of a part of a melody for many songs.

A slide length of four beats was chosen. With this length a few hummed pieces can be made from a tune that is hummed for a period of time that is a little longer than the window size.

For successive notes, a difference in the MIDI note number ranging between -10 and -14 was chosen to be the range for double-pitch error correction. This point is also relevant to the discussion in Section 8.1. Figure 7 shows that about 97.5% of the entire tone differences in successive notes were in the range between -12 and +12. Though more radical changes in tones can be seen when the next phrase starts, it appears that users seldom hum a tune that has overlapped phrases. As the figure also shows, the difference in successive notes is concentrated around 0, -2, and +2. Thus, a double-pitch error may be found in the range between -12-2 and -12+2 in tones of successive notes. As a result, the range between -14 and -10 of MIDI notes determines the thresholds of double-pitch error correction.

A DELL PowerEdge 2300 with twin 500MHz Pentium 3's and 1GB main memory was used as the database server for the performance evaluation.

Figure 13 shows the structure of the experimental system. A database server and a client PC were connected

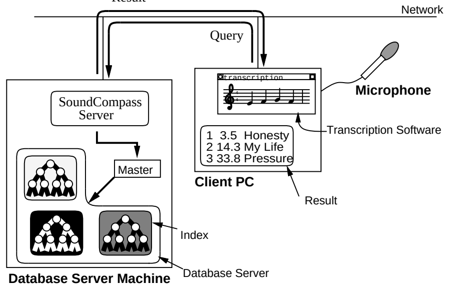


Figure 13: Experimental system structure.

through a network. The query-by-humming system server (*SoundCompass Server*), a master program that handles queries, and a database server worked together in the server machine. The database server housed the indices of feature vectors. A microphone was connected to the client PC, which included a GUI and the transcription software for the query-by-humming system. Hummed tunes input through the microphone were transcribed and converted into MIDI format and sent to the SoundCompass Server. The server processed the hummed tunes in MIDI format according to the method described in Section 4.2 and then sent a query to the master program. The master then sent the query to the database server and the server returned the retrieval result to the SoundCompass Server. The SoundCompass Server sent the result to the client PC, and the results were displayed by the GUI.

7.2 Evaluation for Size of Database

As shown in Figure 3, about 60% of the subdata of a song have a duplicate structure. By chopping up the 10,069 melody data with a 16-beat window size and a four-beat slide length, 938,028 subdata can be generated. After the duplication deletion, the number of subdata was reduced to 548,195. As a result, we were able to reduce the size of the database by 38.6%. The accuracy of the retrieval results obtained from the smaller database was the same as the results obtained from the large database.

Fast-tempo songs were copied to make half-tempo songs. According to Figure 4, the number of songs with tempo 150 or more is 1,063, which is 10% of the total number of songs in the database. Thus, our database held 11,132 song versions (10,069 songs). As a result, songs that had not been retrieved because of tempo errors could be retrieved.

The maximum number of songs that can be copied depends on the size of the database to be managed. We thought that copying only songs with tempo 150 or more might not completely ensure retrieval. However, users did not seem to make any tempo mistakes for songs with tempo 140 or less. By copying these songs, an additional 70,017 subdata units were generated but that number was reduced to 52,146 by deleting the duplication of identical subdata.

The number of subdata before duplication deletion was 938,028. However, after the deletion, it became 600,342 (a 36% reduction in size) even when subdata generated from half-tempo songs were added. As a result, we were able to obtain a smaller database and more accurate retrieval results.

7.3 Evaluation for Feature Vectors

In this section, the effects of feature vectors on retrieval accuracy are examined.

7.3.1 Evaluation of Resolution

As shown in Figure 2, the eighth-note is the most frequently occurring note. To determine the effective resolution for tone transition feature vectors, accuracy of the retrieval results is compared with the results generated through the use of tone transition feature vectors whose resolution is an eighth-note, and the results generated through the use of vectors whose resolution is a quarter-note. Figure 14 shows the results obtained. The figure shows the percentage of times in which a correct song name appears within the rank. In the figure, the x axis represents rank, and the y axis represents percentage of correct retrieval. For example, in 65% of the 186 hummed tunes the correct answer was retrieved within the 10th rank when the eighth-note was used as the resolution for tone transition feature vectors.

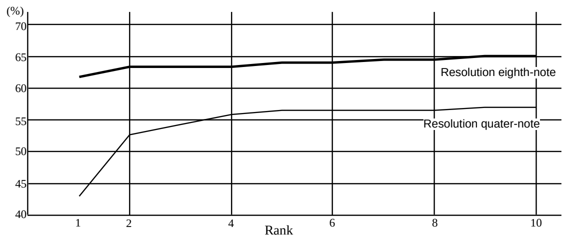


Figure 14: Evaluation of resolution for tone transition feature vectors.

Figure 14 reveals that the eighth-note is better than a quarter-note as the resolution for tone transition feature vectors.

7.3.2 Evaluation of Partial Feature Vector

A partial tone transition feature vector was proposed in Section 5.2 to solve the problem that the starts of tone transition feature vectors generated from melody subdata are not always consistent with the starts of the vectors generated from hummed pieces.

Figure 15 shows the retrieval results for examining the effect of the partial tone transition feature vectors on retrieval accuracy. The figure shows the percentage of times in which a correct song name appears within the rank. In the figure, the x axis represents rank, and the y axis represents percentage. For example, in 70% of the 186 hummed tunes the correct song name was retrieved within the second rank when the partial feature vector was used.

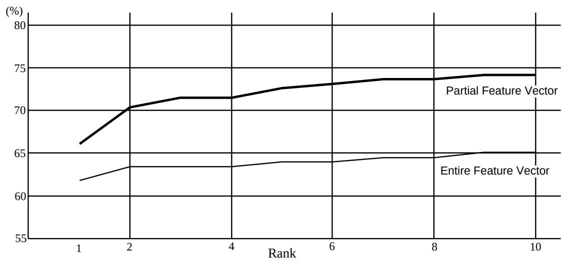


Figure 15: Evaluation of the partial tone transition feature vector.

Figure 15 reveals that the partial feature vector is an extremely good measure for retrieval. The main reason for this improvement in accuracy is that hummed tunes that could not be retrieved correctly because of the gaps between the starts of vectors became retrievable.

7.3.3 Evaluation of Combination of Tone Transition Feature Vector and Tone Distribution Feature Vector

The functions of tone transition feature vectors and/or tone distribution feature vectors were investigated based on the discussion in Section 5.

The partial tone transition vector alone seems to be good enough for retrieval, however, it may retrieve multiple possible answers with an identical similarity. On the other hand, the tone distribution feature vector alone can not retrieve as many correct answers as the tone transition feature vector does. However, through the combined use of the two, more accurate retrieval results can be obtained because the addition of the extra information helps the system narrow down the number of possible correct answers. The figure shows the percentage of times in which a correct song name

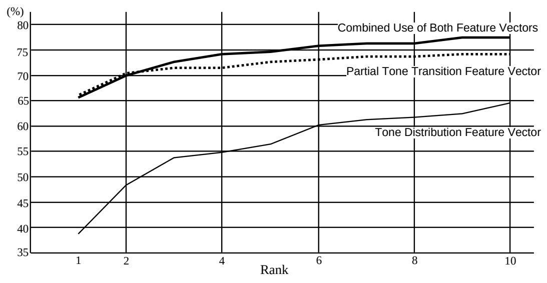


Figure 16: Evaluation of combination.

appears within the rank. In the figure, the x axis represents rank, and the y axis represents percentage of correct retrieval. For example, in 75% of the 186 hummed tunes the correct answer was retrieved within the fifth rank when both tone transition feature vector and tone distribution feature vector were used.

Thus, this figure reveals that the combined use of both feature vectors can increase the retrieval accuracy.

7.4 Evaluation for Or'ed Retrieval

In Section 6, retrieval was done by making multiple search keys from a hummed tune when a user hummed for a period of time longer than the window size for chopping, and getting the final result by or'ed retrieval among the search keys.

Figure 17 shows the retrieval results obtained by the or'ed retrieval among the search keys and those obtained only by a search key generated from the middle part of the hummed tunes. The figure shows the percentage of times

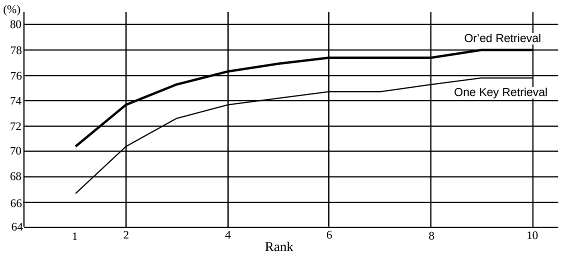


Figure 17: Evaluation for or'ed retrieval.

in which a correct song name appears within the rank. In the figure, the x axis represents rank, and the y axis represents percentage. For example, in 74% of the 186 hummed tunes the correct answer was retrieved within the second rank when or'ed retrieval among the search keys was used.

Thus, this figure reveals that or'ed retrieval among the search keys provides more accurate results than can be obtained with a single search key.

7.5 Evaluation of Performance

Figure 18 shows the execution times in the database server. One line represents the retrieval time when indices are used, and the other was obtained through the use of brute force searching. Songs to make subsets of the database for this experiment were chosen randomly.

This figure reveals that the greater the size of the database is, the higher the efficiency the indices provide. It also reveals that the database server with indices can provide a result within one second for the database with all 11,132 song versions.

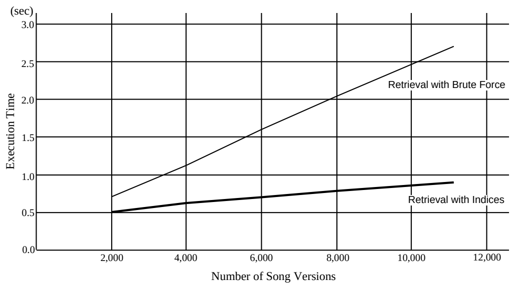


Figure 18: Performance evaluation.

7.6 Summary of Evaluations

In our music retrieval system, a database that holds copied half-tempo songs and has no duplicated subdata was generated. Partial tone transition feature vectors are used instead of the entire tone transition feature vectors. Their resolution is an eighth-note. Both a partial tone transition feature vector and a tone distribution feature vector are used. Multiple search keys are generated from a hummed tune and the result obtained by or'ed retrieval among the search keys becomes the final result. Indices are used for retrieval.

As a result, this music retrieval system is able to provide correct answers within the fifth rank for about 75% of hummed tunes that are recognizable as a part of a song. The answer, which is the combination of each result generated from multiple feature vector spaces, can be obtained in at most one second.

8 Towards Practical Use

This section describes a practical query-by-humming system that we hope to offer commercially in the near future. We visualize a system that will be used, for example, in the following situation:

Situation example:
Some people are singing happily in a karaoke room about three meters square. Person A is singing, and it's Person B's turn next. Person B knows the song he wants to sing but not its title. So while A is singing, B retrieves the song title by humming it, so now he can enter the title into the karaoke system so he can sing it.

In this section, problems related to noise and note fragmentation are discussed in addition to the problems presented in Section 4.2.

The tunes in Figure 19 and 20 are from the same song and were hummed by the same person. The one in Figure 19 was hummed in a quiet place, and the other in Figure 20 was hummed in a noisy place. These scores are used as aids in the following discussion.

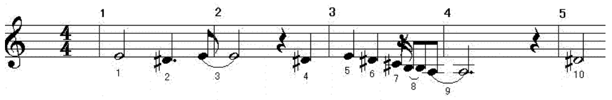


Figure 19: Hummed tune of a song C in a quiet place.

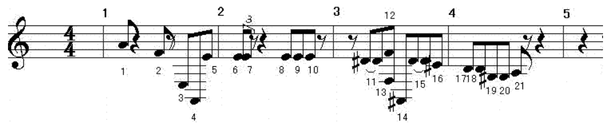


Figure 20: Hummed tune of a song C in a noisy place.

8.1 Noise Deletion

Noise is a note that was recorded when the hummed tune was recorded, but has nothing to do with the melody. Noise is caused by the singing of others, e.g. A in the situation example, and by mistakes made by the hummer. For example, in Figure 20, there are 21 notes, and three notes, 1, 4, and 14, are definitely noise notes. Noise notes must be deleted from hummed tunes to extract correct features from the tunes and then to make effective feature vectors.

As shown in Figure 7, successive notes seldom have big differences in pitch. Thus, we propose these rules for deleting noise note.

1. If there is a duration of silence of more than m beats within n beats from the beginning of the recording, the part before the silence is deleted.
2. A note which has a wide range of difference exceeding a threshold p in pitch for a mean tone is regarded as a noise note and hence is deleted. The checking of noise notes is done twice in a tune, once in timewise normal order, and then in timewise opposite order.

The mean tone is recalculated whenever the tones of the notes are checked. In this recalculation, notes whose tone exceeds the threshold p are not used for the calculation. As a result, the more the recalculation is done, the more accurate the mean tone becomes.

8.2 Note Fragmentation

The number of relatively short notes in a tune hummed in a noisy place is greater than those of one hummed in a quiet place, even if the same person hummed both in the same way. This may be caused by confusion of the transcription software, which cannot decide whether or not a tone to be recorded has been sounded in a noisy environment. Thus, note processing of the hummed tune is not appropriate when the tune is recorded in a noisy place. Fortunately, our system uses beat processing rather than note processing, as described in Section 4.2.1. Thus, note fragmentation has no effect on our system performance.

8.3 Evaluation of Noise Deletion

A query-by-humming experiment in a noisy place was conducted to evaluate the noise deletion method described in Section 8.1. In this experiment, 8, 4, and 15 were selected as parameters for m, n, and p respectively. From our experience, we had found that there are few songs with relatively long rests (four beats) at the beginning of a phrase. Thus, we selected eight beats as the threshold to detect a long rest and delete the midi data before the rest as noise notes.

A threshold of 15 in tone difference for noise deletion was chosen. This is also relevant to the discussion in Section 7.1, which showed that there may be double-pitch errors in tone difference in the range between -12-2 and -12+2. The value of ± 15, which exceeds the threshold for double-pitch error correction, was selected as the threshold for the noise note deletion.

For example, the tune in Figure 21 was obtained from the hummed tune in Figure 20 after deleting noise notes with the method described in Section 8.1 and correcting double-pitch errors with the method described in Section 4.2.2. In other words, two noise notes (1 and 4) can be deleted, but one (14) can not be. The third note (3) was a double-pitch note, and it was corrected by the method described in Section 4.2.2.

In this experiment, 31 hummed tunes were recorded in a noisy place. Two people, one male and one female, recorded their hummed tunes in a room where four kinds of background music with vocals were played very loudly. The total number of hummed notes was 1,492. There were 87 noisy notes which were definitely judged as noise, and 61 noise notes were deleted, while 20 notes that were not noise were deleted.

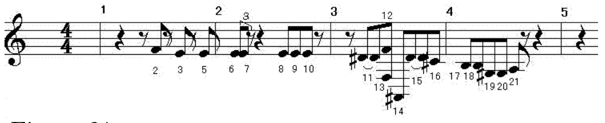


Figure 21: Hummed tune of a song C after noise removal.

Figure 22 shows the results of the accuracy evaluation of the retrieval for these 31 hummed tunes. A noise deletion routine was added to the retrieval system described in Section 7.6.

The figure shows the percentage of times in which a correct answer appears within the rank. In the figure, the x axis represents rank, and the y axis represents percentage.

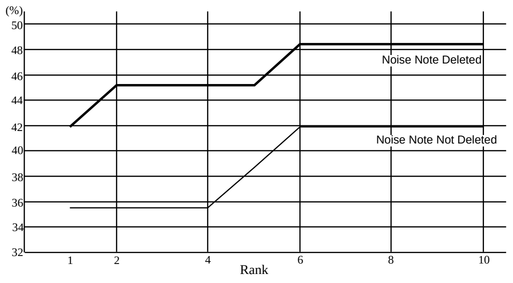


Figure 22: Evaluation for noise note deletion.

This figure reveals that the method for noise deletion proposed in Section 8.1 is efficient because accuracy was increased.

8.4 GUI and User Interface Hardware

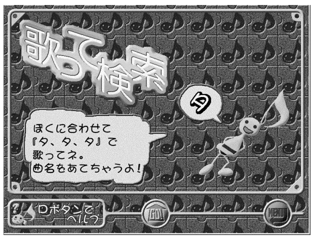
We created the graphics and prepared equipment shown in Figure 23 for practical use, where (a) to (c) show the graphics and (d) shows the equipment for the user interface.

Figure 23(a) shows a graphic for the opening. In the graphic, the title “Query by Singing” appears in large Japanese letters. The doll on the right is a character named “Onpu-chan”. Its head is an eighth-note and it is named after the term “note” in Japanese. It tells users to sing clearly with the syllable “ta” and also tells them to keep tempo by moving its body right and left.

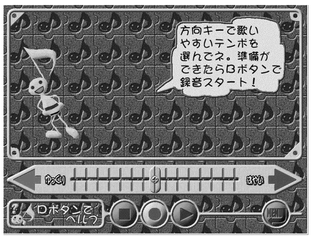
Figure 23(b) is a graphic for recording hummings. Here, Onpu-chan moves its body right and left again as a metronome. The speed can be adjusted using the direction bar under Onpu-chan. Onpu-chan requires the user to find the tempo at which it is easy for them to sing and then push the record button.

Figure 23(c) is a graphic showing the retrieval results. When the user pushes the retrieval button after recording a hummed tune in Figure 23(b), this graphic appears within a few seconds. In this graphic, the scores registered for each song are shown in addition to the song title. The score is calculated from the dissimilarity of the match using a certain function. For example, “Love Train” was retrieved as the second match in the database, and the system says the hummed tune was similar to “Love Train” with a score of 80 out of 100.

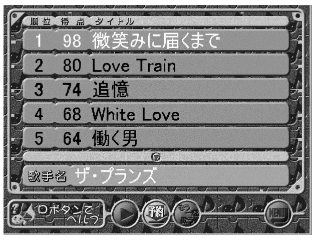
Figure 23(d) shows the equipment for the user interface: a microphone with a headphone and a game pad for user input. Tempo can be recognized by listening to the ticks of a metronome through the headphone, as well as by following the movements of Onpu-chan. The game pad is used for input because it is familiar to young people in Japan.



(a) Opening



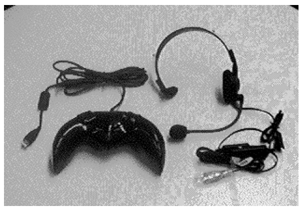
(b) Recording a Hummed Tune



順位	得点	タイトル
1	98	微笑みに届くまで
2	80	Love Train
3	74	追憶
4	68	White Love
5	64	働く男

歌手名 ザ・プランズ

(c) Retrieval Result



(d) Microphone and Game Pad

Figure 23: Practical query-by-humming system.

9 Conclusion

This paper describes a music retrieval system that accepts hummed tunes as queries. The database currently holds 11,132 song versions (10,069 songs), and the database server is able to retrieve songs from the database in at most one second. The hummed tunes can be any part of a song. In about 75% of the retrievals, the correct song name of recognizable hummed tunes is listed within the fifth rank. In addition to this result, medleys that included a part of a song that had a part similar to the hummed tune were also retrieved. The previous system [9] provided correct song names within the fifth rank for about 50% of the hummed tunes. Thus, the accuracy of retrieval increased by more than 20% over that of the previous system.

To achieve these results, we introduced partial tone transition feature vectors, implemented or'ed retrieval among the search keys, made copies of fast songs at half-tempo, decreased the size of the database, and corrected double-pitch errors based on the results of music analyses. Furthermore, in aiming towards the development of a practical service, we prepared graphics and equipment for users and proposed a noise note deletion method.

10 Future Directions

We are still researching this music retrieval system in the hope of improving its performance and accuracy. We also hope to decrease the size of the database without removing any songs, and make the retrieval system easier to use. This research represents our first step in the development of a query-by-humming system for practical use with a large music database. Further investigations and evaluations will be needed.

A very wide range of issues, e.g., from the technical side, such as the implementation of more efficient feature vectors, to fundamentals, such as what the similarities in music are, remain as future work.

It is well known that people can often identify a song by hearing only a part of it hummed, even if the humming is out of tempo or tune. Our system has significant tolerance to input errors but does not approach the ability of most people when it comes to identifying hummed tunes of music they are familiar with. Our future research will focus on making the system more human-like in this regard.

Acknowledgment

We would like to thank Daiichi Kosho Co., Ltd., in particular Mr. Yuji Namekata, for providing the 10,069 MIDI song data.

References

[1] S. Blackburn and D. DeRoure. A Tool for Content Based Navigation of Music. In *Proc. ACM Multimedia 98*, 1998.

[2] K. Curtis, N. Taniguchi, J. Nakagawa, and M. Yamamuro. A comprehensive image similarity retrieval system that utilizes multiple feature vectors in high dimensional space. In *Proceedings of International Conference on Information, Communication and Signal Processing*, pages 180–184, September 1997.

[3] J. J. Dubnowski, R. W. Schafer, and L. R. Rabiner. Real-Time Digital Hardware Pitch Detector. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, ASSP-24(1):2–8, February 1976.

[4] J. Foote. An overview of audio information retrieval. In *Multimedia Systems 7*, pages 2–10. ACM, January 1999.

[5] J. Foote. Visualizing Music and Audio using Self-Similarity. In *Proc. ACM Multimedia 99*, pages 77–80, November 1999.

[6] J. T. Foote. Content-Based Retrieval of Music and Audio. In *Proc. SPIE, vol3229*, pages 138–147, 1997.

[7] A. Ghias, J. Logan, and D. Chamberlin. Query By Humming. In *Proc. ACM Multimedia 95*, pages 231–236, November 1995.

[8] N. Kosugi, Y. Nishihara, S. Kon'ya, M. Yamamuro, and K. Kushima. Let's Search for Songs by Humming! In *Proc. ACM Multimedia 99 (Part 2)*, page 194, November 1999.

[9] N. Kosugi, Y. Nishihara, S. Kon'ya, M. Yamamuro, and K. Kushima. Music Retrieval by Humming. In *Proceedings of PACRIM'99*, pages 404–407. IEEE, August 1999.

[10] W. Y. Ma, B. S. Manjunath, Y. Luo, Y. Deng, and X. Sun. NETRA: A Content-Based Image Retrieval System. http://maya.ece.ucsb.edu/Netra/.

[11] R. J. McNab, L. A. Smith, D. Bainbridge, and I. H. Witten. The New Zealand Digital Library MELody inDEX. http://www.dlib.org/dlib/may97/meldex/05written.html, May 1997.

[12] Muscle Fish LLC. http://www.musclefish.com/.

[13] Y. Nishihara, N. Kosugi, S. Kon'ya, and M. Yamamuro. Humming Query System Using Normalized Time Scale. In *Proceedings of CODAS'99*, March 1999.

[14] P. Y. Rolland, G. Raskinis, and J. G. Ganascia. Musical Content-Based Retrieval: an Overview of the Melodiscov Approach and System. In *Proc. ACM Multimedia 99*, pages 81–84, November 1999.

[15] J. R. Smith and C. S. Li. Image classification and querying using composite region templates. In *Journal of Computer Vision and Image Understanding*, 1999.

[16] WILDCAT CANYON SOFTWARE. AUTOSCORE. http://www.wildcat.com/Pages/AutoscoreMain.htm.

[17] N. Taniguchi and M. Yamamuro. Multiple Inverted Array Structure for Similar Image Retrieval. In *IEEE Multimedia '98*, pages 160–169, 1998.

[18] A. Uitdenbogerd and J. Zobel. Melodic Matching Techniques for Large Music Database. In *Proc. ACM Multimedia 99*, pages 57–66, November 1999.

[19] S. Wu and U. Manber. Fast Text Searching Allowing Errors. *Communications of the ACM*, 35(10):83–91, October 1996.

[20] M. Yamamuro, K. Kushima, H. Kimoto, H. Akama, S. Kon'ya, J. Nakagawa, K. Mii, N. Taniguchi, and K. Curtis. ExSight – Multimedia Information Retrieval System. In *20th Annual Pacific Telecommunications Conference, PTC'98 Proceedings*, pages 734–739, 1998.

[21] A. Yoshitaka and T. Ichikawa. A Survey on Content-Based Retrieval for Multimedia Databases. *IEEE Trans. Knowledge and Data Engineering*, 11(1):81–93, Feb. 1999.