# Query By Humming

## Musical Information Retrieval in An Audio Database

Asif Ghias Jonathan Logan David Chamberlin
Brian C. Smith

Cornell University
{ghias,bsmith}@cs.cornell.edu, logan@ghs.com, chamber@engr.sgi.com

**ABSTRACT**
The emergence of audio and video data types in databases will require new information retrieval methods adapted to the specific characteristics and needs of these data types. An effective and natural way of querying a musical audio database is by humming the tune of a song. In this paper, a system for querying an audio database by humming is described along with a scheme for representing the melodic information in a song as relative pitch changes. Relevant difficulties involved with tracking pitch are enumerated, along with the approach we followed, and the performance results of system indicating its effectiveness are presented.

**KEYWORDS:** Musical information retrieval, multimedia databases, pitch tracking

### Introduction

Next generation databases will include image, audio and video data in addition to traditional text and numerical data. These data types will require query methods that are more appropriate and natural to the type of respective data. For instance, a natural way to query an image database is to retrieve images based on operations on images or sketches supplied as input. Similarly a natural way of querying an audio database (of songs) is to hum the tune of a song.

Such a system would be useful in any multimedia database containing musical data by providing an alternative and natural way of querying. One can also imagine a widespread use of such a system in commercial music industry, music radio and TV stations, music stores and even for one's personal use.

In this paper, we address the issue of how to specify a hummed query and report on an efficient query execution implementation using approximate pattern matching. Our approach hinges upon the observation that melodic contour, defined as the sequence of relative differences in pitch between successive notes, can be used to discriminate between melodies. Handel[3] indicates that melodic contour is one of the most important methods that listeners use to determine similarities between melodies. We currently use an alphabet of three possible relationships between pitches ('U', 'D', and 'S'), representing the situations where a note is above, below or the same as the previous note, which can be pitch-tracked quite robustly. With the current implementation of our system we are successfully able to retrieve most songs within 12 notes. Our database currently comprises a collection of all parts (melody and otherwise) from 183 songs, suggesting that three-way discrimination would be useful for finding a particular song among a private music collection, but that higher resolutions will probably be necessary for larger databases.

This paper is organized as follows. The first section describes the architecture of the current system. The second section describes what pitch is, why it is important in representing the melodic contents of songs, several techniques for tracking

pitch we tried and discarded, and the method we settled on. Next we discuss pattern matching as it is used in the current implementation of the database. The last two sections describe our evaluation of the current system and specifie some future extensions that we are considering incorporating in the existing system.

### System Architecture

There are three main components to the our system: a pitch-tracking module, a melody database, and a query engine. The architecture is illustrated in Figure . Operation of the system is straight-forward. Queries are hummed into a microphone, digitized, and fed into a pitch-tracking module. The result, a contour representation of the hummed melody, is fed into the query engine, which produces a ranked list of matching melodies.

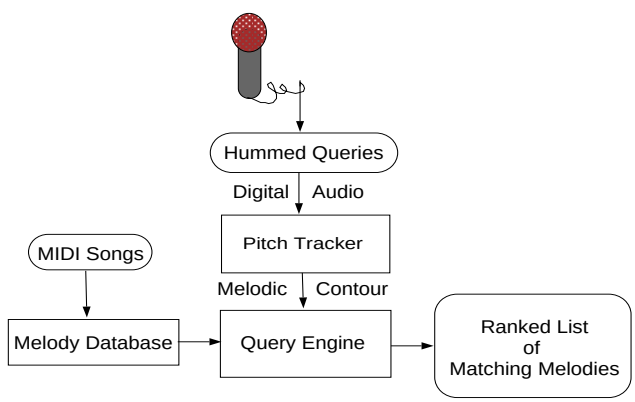


Figure 1: System Architecture

The database of melodies was acquired by processing public domain MIDI songs, and is stored as a flat-file database. Pitch tracking is performed in Matlab, chosen for its built-in audio processing capabilities and the ease of testing a number of algorithms within it. Hummed queries may be recorded in a variety of formats, depending upon the platform-specific audio input capabilities of Matlab. We have experimented with 16-bit, 44Khz WAV format on a Pentium system, and 8-bit, 8Khz AU format on a Sun Sparcstation. The query engine uses an approximate pattern matching algorithm[1], described in below, in order to tolerate humming errors.

### Tracking Pitch in Hummed Queries

This section describes how user input to the system (humming) is converted into a sequence of relative pitch transitions. A note in the input is classified in one of three ways: a note is either the same as the previous note (S), higher than previous note (U), or lower than the previous note (D). Thus, the input is converted into a string with a three letter alphabet (U,D,S). For example, the introductory theme Beethoven's 5th Symphony would be converted into the sequence: - S S D U S S D (the first note is ignored as it has no previous pitch).

To accomplish this conversion, a sequence of pitches in the melody must be isolated and tracked. This is not as straight-forward as it sounds, however, as there is still considerable controversy over exactly what pitch is. The general concept of pitch is clear: given a note, the pitch is the frequency that most closely matches what we hear. Performing this conversion in a computer can become troublesome because some intricacies of human hearing are still not understood. For instance, if we play the $4^{th}$, $5^{th}$, and $6^{th}$ harmonics of some fundamental frequency, we actually hear the fundamental frequency, not the harmonics even though the fundamental frequency is not present. This phenomenon was first discovered by Schouten in some pioneer investigations carried out from 1938 to 1940. Schouten studied the pitch of periodic sound waves produced by an optical siren in which the fundamental of 200Hz was canceled completely. The pitch of the complex tone, however, was the same as that prior to the elimination of the fundamental. [12]

Since we were interested in tracking pitch in humming, we examined methods for automatically tracking pitch in a human voice. Before we can estimate the pitch of an acoustic signal, we must first understand how this signal is created, which requires forming a model of sound production at the source. The vibrations of the vocal cords in voiced sounds are caused as a consequence of forces that are exerted on the laryngeal walls when air flows through the glottis (the gap between the vocal cords[1]). Hess [5] describes a model of the vocal cords as proposed by Hirano [6]. For the purposes of this paper though, it is sufficient to know that the glottis repeatedly opens and closes thus providing bursts of air through the vocal tract.

The vocal tract can be modeled as a linear passive transmission system with a transfer function $H(z)$. If we add an additional transfer function $R(z)$ which takes into account the radiation, the output impedance of the vocal tract can approximately be set to zero. In the neutral position where the vocal tract can be regarded as a uniform tube, the resonances of the vocal tract occur at sound wavelengths of

$$\lambda = \frac{4L}{(2k-1)c}; \qquad k = 1, 2, 3, \ldots \tag{1}$$

With $L = 17$cm (average value of vocal-tract length) and a sound propagation speed of $c = 340\frac{m}{s}$, the frequencies of these resonances will be:

$$F_k = (2k-1) \cdot 500Hz; \qquad k = 1, 2, 3, \ldots \tag{2}$$

The frequencies $F_k$ are called formant frequencies.

The resulting sound that we hear is considered to be the convolution of the excitation pulse created by the glottis and the formant frequencies. Therefore, if we want to model a speech signal, we start with a train of excitation pulses as shown in figure 2. For the formant frequencies, use equation (2) with $k \in \{1, 2, 3\}$. This gives formant frequencies: $F_1 = 500Hz$, $F_2 = 1500Hz$, and $F_3 = 2500Hz$. Combining these frequencies and adding an exponential envelope produces the formant structure shown in figure 3. By convolving the train of excitation pulses with the formant structure, we get a synthesized pitch as shown in figure 4.

[1]The terms vocal folds and vocal chords are more or less used as synonyms in the literature

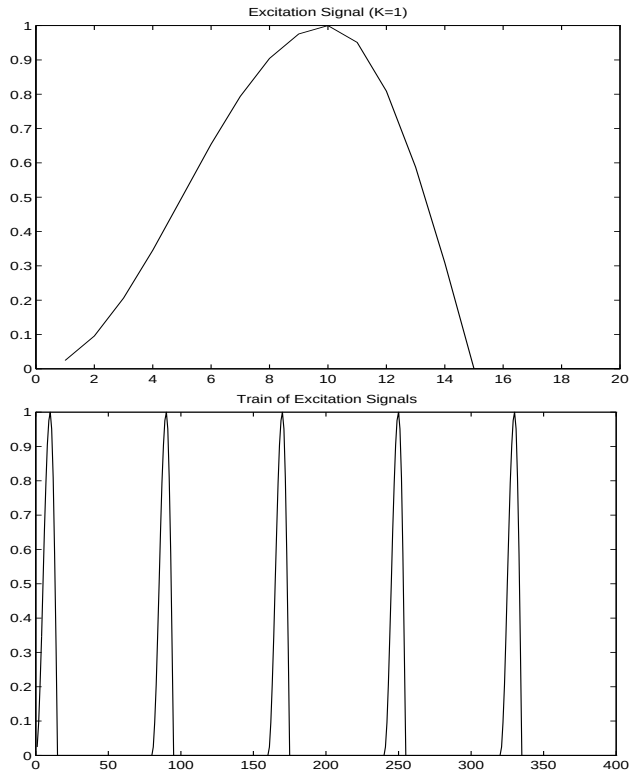


Figure 2: Excitation signal used to create the synthesized pitch. The period in the train of excitations is $T_0 = 0.01$s making the pitch 100Hz.

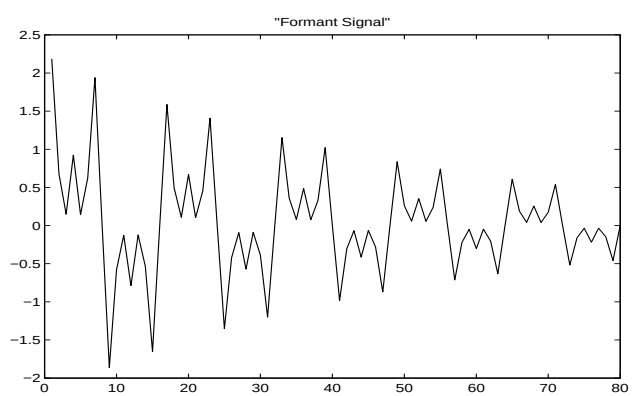


Figure 3: Formant structure created using 500Hz, 1500Hz and 2500Hz as the formant frequencies

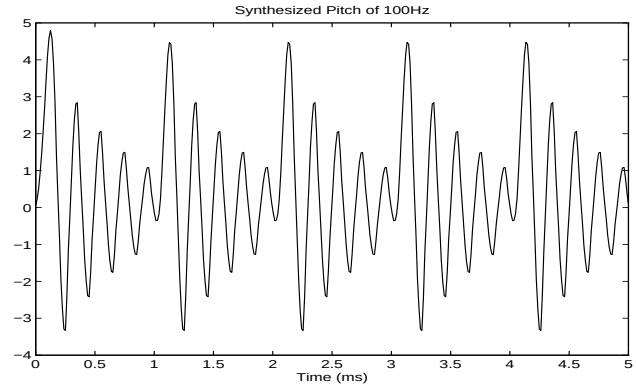


Figure 4: Synthesized pitch of 100Hz created by convolving the train of excitation pulses (spaced by 0.01s) and the formant structure

Now that we have a model of the human voice, how can it be converted to pitch? The most prevalent view on pitch is that what we hear as pitch is actually the frequency at which the bursts of air occur. So if we can track those bursts of air we can find the pitch of the segment.

### Tracking pitch

We tried three methods for tracking pitch: Autocorrelation, Maximum Likelihood, Cepstrum Analysis.

- Autocorrelation

Autocorrelation is one of the oldest of the classical pitch trackers[7]. Autocorrelation isolates and tracks the peak energy levels of the signal which is a measure of the pitch. Referring back to figure 3, we see that the signal $s(n)$ peaks where the impulses occur. Therefore, tracking the frequency of this peaks should give us the pitch of the signal.
In order to get the frequency of these peaks we can employ autocorrelation as defined by:

$$R(l) = \sum_{k=-\infty}^{\infty} h(k)h(l+k) \tag{3}$$

Unfortunately autocorrelation is subject to aliasing (picking an integer multiple of the actual pitch) and is computationally complex. We found our implementation of autocorrelation to require approximately 45 seconds for 10 seconds of 44KHz, 16-bit audio on a 90MHz pentium workstation.

- Maximum Likelihood

Maximum Likelihood[14] is a modification of Autocorrelation that increases the accuracy of the pitch and decreases the chances of aliasing.

Unfortunately, the computational complexity of this method makes autocorrelation look blindingly fast. A straight-forward implementation in Matlab takes approximately one hour to evaluate 10 seconds of audio on a 90MHz Pentium workstation. With some optimizations, we improved the performance to approximately 15 minutes per 10 seconds of audio, but this is still far too slow for our purposes. Therefore, we discarded this method. For a detailed explanation of this method, the reader may refer to [14].

- Cepstrum Analysis

Cepstrum analysis is the definitive classical method of pitch extraction. For an explanation, the reader is directed to Oppenheim and Schafer's original work in [10] or in a more compact form in [11]. We found that this method did not give very accurate results for humming.

The output of these methods can be construed as a sequence of frequency estimations for successive pitches in the input. We convert these estimates into a three-step contour representation by comparing each estimated pitch with the previous one. In our system adjacent pitches are considered the same if they are within a quarter-step of each other (on an equal-tempered musical scale), but this parameter is adjustable.

After analyzing the costs and benefits of these methods, we decided to use a modified form of autocorrelation for our implementation.

### Searching the database

Having described how the user input (a hummed tune) is converted into a string in a 3 letter alphabet, we now discuss our method for searching an audio database. Our method of searching the database is simple. Songs in the database are preprocessed to convert the melody into a stream of U,D,S characters, and the converted user input (the *key*) is compared with all the songs. The pattern-matching uses a 'fuzzy' search to allow for errors within the matches. These errors reflect the inaccuracies in the way people hum as well as errors in the representation of the songs themselves.

For performing the key-search within the database we need an efficient approximate pattern matching algorithm. By "approximate" we mean that the algorithm should be able to take into account various forms of errors.

Figure summarizes the various forms of errors anticipated in a typical pattern matching scheme. The algorithm that

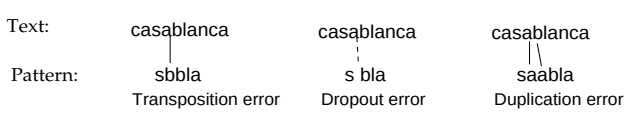


Figure 5: Three forms of anticipated errors with one mismatch

we adopted for this purpose is described by Baesa-Yates and Perleberg [1]. This algorithm addresses the problem of string matching with k mismatches. The problem consists of finding all instances of a pattern string $P = p_1p_2p_3...p_m$ in a text string $T = t_1t_2t_3..t_n$ such that there are at most $k$ mismatches (characters that are not the same) for each instance of $P$ in $T$. When $k = 0$ (no mismatches) we have the simple string matching problem, solvable in $O(n)$ time. When $k = m$, every substring of $T$ of length $m$ qualifies as a match, since every character of $P$ can be mismatched. Each of the errors in the figure above corresponds to k=1.

It is worth mentioning that several algorithms have been developed that address the problem of approximate string matching. Running times have ranged from $O(mn)$ for the brute force algorithm to $O(kn)$ [9] or $O(n \log(m))$ [2]. The algorithm that we adopted offers better performance for average cases than most other algorithms.

The worst case for this algorithm occurs when $P$ (the key) consists of $m$ occurrences of a single distinct character, and $T$ (contour representation of song) consists of $n$ instances of that character. In this case the running time is $O(mn)$. However this is neither a common nor useful situation for our purposes. In the average case of an alphabet in which each character is equally likely to occur, the running time is $O(n(1 + \frac{m}{|\Gamma|})$ where $|\Gamma|$ is the size of the alphabet.

The database incorporates the key-searching scheme (using pattern matching techniques explained above). We envisioned the following design goals for the database. For a given query, the database returns a list of songs ranked by how well they matched the query, not just one best match. The number of matches that the database should retrieve depends upon the error-tolerance used during the key-search. This error-tolerance could be set in one of two possible ways: either it can be a user-definable parameter or the database can itself determine this parameter based on, for example by heuristics that depends on the length of the key. This design gives the user an opportunity to perform queries even if the user is not sure of some notes within the tune.

From the results of the query the user can identify the song of interest. If the list is too large, the user can perform a new query on a restricted search list consisting of songs just retrieved. A consequence of this scheme is that the user can identify sets of songs that contain similar melodies.

### Evaluation

This section describes the results of an experimental evaluation of the system. Our evaluation tested the tolerance of the system with respect to input errors, whether from mistakes in the user's humming or from problems with the pitch-tracking.

### Robustness

The effectiveness of this method is directly related to the accuracy with which pitches that are hummed can be tracked and the accuracy of the melodic information within the database. Under ideal circumstances, we can achieve close to 100% accuracy tracking humming, where ideal circumstances mean the user places a small amount of space between each note and hits each note strongly. For this purpose, humming short notes is encouraged. Even more ideal is for the user to aspirate the notes as much as possible, perhaps going so far as to voice a vowel, as in "haaa haaa haaa". We have currently only experimented with male voices.

The evaluation database currently contains a total of 183

songs. Each song was converted from public domain General MIDI sources. Melodies from different musical genres were included, including both classical and popular music. A few simple heuristics were used to cut down on the amount of irrelevant information from the data, e.g. MIDI channel 10 was ignored as this is reserved for percussion in the General MIDI standard. However the database still contains a great deal of information unrelated to the main theme of the melody. Even with this limitation, we discovered that sequences of 10-12 pitch transitions were sufficient to discriminate 90% of the songs.

As a consequence of using a fast approximate string matching algorithm, search keys can be matched with any portion of the melody, rather than just the beginning. As the size of the database grows larger, however, this may not prove to be an advantage.

### Performance

The version of the pitch-tracker using a modified form of autocorrelation[2] takes between 20 and 45 seconds on a Sparc10 workstation to process typical sequences of hummed notes. A brute-force search of the database unsurprisingly shows linear growth with the size of the database, but remains below 4 seconds for 100 songs on a Sparc2. Therefore the search time is currently effectively limited by the efficiency of the pitch-tracker.

Contour representations for each song are currently stored in separate files, so opening and closing files becomes a significant overhead. Performance could be improved by packing all the songs into one file, or by using a database manager. We plan to modularize our code to make it independent of any particular database schema.

### Future directions and Related Work

We plan to improve the performance and speed and robustness of the pitch-tracking algorithm by using a cubic-spline wavelet. The cubic spline wavelet peaks at discontinuities in the signal (i.e. the air impulses). One of the most significant features of the wavelet analysis is that it can be computed in $O(n)$ time. Currently, the pitch tracker is the slowest link in our system, so using wavelets for this purpose has obvious advantages.

The pattern matching algorithm in its present form does not discriminate the various forms of pattern matching errors discussed earlier, but only accounts for them collectively. Some forms of errors may be more common than others depending upon the way people casually hum different tunes. For example drop-out errors reflected as dropped notes in tunes are more common than transposition or duplication errors. Tuning the key-search so that it is more tolerant to drop-out errors, for example, may yield better results.

The melodic contours of the source songs are currently generated automatically from MIDI data, which is convenient but not optimal. More accuracy and less redundant information could be obtained by entering the melodic themes for particular songs by hand. From a research standpoint, an interesting question is how to extract melodies from complex audio signals[4].

Finally, we would like to characterize the improvement gained by increasing the resolution of the relative pitch differences by considering query alphabets of three, five and more possible relationships between adjacent pitches. Early experiments using an alphabet of five relative pitch differences (same, higher, much higher, lower, much lower) verified that changes of this sort are promising. One drawback of introducing more resolution is that the user must be somewhat more accurate in the intervals they actually hum. We will explore the various tradeoffs involved. An important issue is precisely where to draw the line between notes that are a little higher from the previous note and those that are much higher.

Previous work on efficiently searching a database of melodies by humming seems to be limited. Mike Hawley [4] briefly discusses a method of querying a collection of melodic themes by searching for exact matches of sequences of relative pitches input by a MIDI keyboard. We have incorporated approximate pattern matching, implementing an actual audio database (of MIDI songs) and most significantly by allowing queries to be hummed. Kageyama and Takashima [8] published a paper on retrieving melodies using a hummed melody in a Japanese journal, but we were unable to locate a translated version.

[2]The modifications include low-pass filtering and center-clipping (as described in Sondhi's paper [13]) which help eliminate the formant structure that generally causes difficulty for autocorrelation based pitch detectors.

### REFERENCES


1. Ricardo A. Baesa-Yates and Chris H. Perleberg. Fast and practical approximate string matching. *Combinatorial Pattern Matching, Third Annual Symposium*, pages 185–192, 1992.

2. Ricardo Baeza-Yates and G.H. Gonnet. Fast string matching with mismatches. *Information and Computation*, 1992.

3. Stephen Handel. *Listening: An Introduction to the Perception of Auditory Events*. The MIT Press, 1989.

4. Michael Jerome Hawley. *Structure out of Sound*. PhD thesis, MIT, September 1993.

5. Wolfgang Hess. *Pitch Determination of Speech Signals*. Springer-Verlag, Berlin Heidelberg, 1983.

6. M. Hirano. Structure and vibratory behavior of the vocal folds. In M. Sawashima and F.S. Cooper, editors, *Dynamic aspects of speech production*, pages 13–27. University of Tokyo Press, 1976.

7. L.R. Rabiner J.J. Dubnowski and R.W. Schafer. Real-time digital hardware pitch detector. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-24(1):2–8, Feb 1976.

8. T. Kageyama and Y. Takashima. A melody retrieval method with hummed melody (language: Japanese). *Transactions of the Institute of Electronics, Information and Communication Engineers D-II*, J77D-II(8):1543–1551, August 1994.

9. G. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986.

10. A. V. Oppenheim. A speech analysis-synthesis system based on homomorphic filtering. *J. Acoustical Society of America*, 45:458–465, February 1969.

11. Alan V. Oppenheim and Ronald W. Schafer. *Discrete-time Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, 1989.

12. R. Plomp. *Aspects of tone sensation*. Academic Press, London, 1976.

13. M. M. Sondhi. New methods of pitch extraction. *IEEE Trans. Audio Electroacoust. (Special Issue on Speech Communication and Processing—PartII*, AU-16:262–266, June 1968.

14. James D. Wise, James R. Caprio, and Thomas W. Parks. Maximum likelihood pitch estimation. *IEEE Trans. Acoust., Speech, Signal Processing*, 24(5):418–423, October 1976.