QUERY BY SINGING/HUMMING (QbSH) SYSTEM

Himanshu Gupta
University of Texas at Dallas
hxg151930@utdallas.edu

Himanshi Aggarwal
University of Texas at Dallas
hxa152730@utdallas.edu

ABSTRACT

A Query by Singing/Humming (QbSH) system tries to pull out information of a song given a melody hummed or even sung by the user into it. No musical training is required. The idea is to hum the melody in the microphone, the computer records the hum and extracts features of the melody and rhythm characteristics and then compares features to the features of songs in the database. Then we get ranked list of songs from the database. The goal here is to build an efficient system that collects the songs and responds in seconds.

1. INTRODUCTION

Query by Humming has played a major role as an area of research in audio-based search engines, and building efficient Human Computer Interfaces [20]. The purpose of this project is to experiment with different tools and technologies in the field of the Music Information Retrieval (MIR) and to build a complete functional prototype of a Query by Singing/Humming system [23].

However, building such a system, presents significantly greater challenges because unlike text content, the way to represent and store melodic content in a database does not work here. Moreover, several issues unique to QBH systems are:

1. Users may not make perfect requests. User may start at the wrong key, or he may hum a few notes off-pitch throughout the course of the tune.

2. Capturing pitches and notes from user hums accurately is difficult, even if the user manages to submit a perfect query.

3. Also, it is difficult to capture accurately the melodic information from a pre-recorded music file [14]. Such QbSH systems would be useful in any multimedia database containing musical data by providing an alternative, simpler and natural way of querying. Such a system can be used widely in commercial music industry, music radio and TV stations, music stores and even for one's personal use.

With the help of this project, we address the issue of specifying a hummed query and report on an efficient query execution implementation using approximate pattern matching. Our approach has been built upon the observation that the melodic pitch contour, defined as the sequence of relative differences in pitch between successive notes, can be used to discriminate between the melodies. Handel [11] 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 five possible relationships between pitches ('u', 'd', 'U', 'D', and 'S'), representing the situations where a note is slightly above, significantly above, slightly below, significantly 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 800 notes. Our database currently comprises a collection of all kinds of melodies like monophonics, polyphonics, hummed with 'La' syllable etc. from 25 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 [9].

This paper is organized as follows. The first section describes the problem specification of the current system. The next section describes the Information extraction and analysis. The third section describes the architecture of the current system. Next we discuss 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. Then 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 specify some future extensions that we are considering incorporating in the existing system.

2. RELATED WORK

There are some QBH systems developed by researchers in the past. In 2003 research paper [19], a query by humming based musical retrieval system was developed called TANSEN, it explains in detail about pitch extraction and the algorithm to be used for matching songs. Also in [18], an end to end music search system was developed that we can use to support query by humming. We also came across a MUSART Testbed for Query-By-Humming Evaluation which helps us to organize experiments by providing explicit representations and standard formats for queries, targets, collections (subsets of queries or targets), preprocessing stages, search algorithms, and result reporting [8]. Moreover, a Dynamic Time Warping (DTW) framework explains how to adapt existing time series indexing schemes for the Euclidean distance measure to the DTW distance measure and such an indexing scheme guarantees no false negatives [13, 16, 25]. In [15] paper, a new triplet melody representation and new hierarchical matching algorithm have been presented. In order to extract features from a se-

quence one can also use the Discrete Fourier Transform (DFT) described in detail in [17]. The attractive property of the DFT is that the Euclidean distance in the time domain is preserved in the frequency domain, because of Parseval's theorem. Thus, DFT fulfills the completeness of feature extraction criterion.

In addition, DFT is fast with a complexity of (O(n log n)) [6].In [12] article, a comprehensive survey of recent work on time series data mining has been done to show that because of several kinds of experimental flaws, many of the results claimed in the literature have very little generalizability to real world problems. The basic steps that most PEAs perform to track the pitch of a signal are: First, the signal is split into windows. Then, for each window the following steps are performed: (i) the spectrum is estimated using a short-time Fourier transform (STFT), (ii) a score is computed for each pitch candidate within a predefined range by computing an integral transform (IT) over the spectrum, and (iii) the candidate with the highest score is selected as the estimated pitch. The pitfalls which most of the pitch extraction algorithms have are related to missing harmonics, subharmonic errors, not recognizing the pitch of inharmonic signals etc. A sawtooth waveform inspired pitch estimator (SWIPE) has been developed for speech and music. This type of signal is the one that motivated the name of the algorithm: sawtooth waveform inspired pitch estimator: SWIPE [7]. It is assumed in SWIPE that what matters to estimate pitch and its strength is the amplitude of the spectral components of the sound, and not their phase, which in fact is ignored here. In SWIPE, the square-root warping of the spectrum is done rather than using logarithms as it is more convenient for three reasons. First, it matches better the response of the auditory system to amplitude, which is close to a power function with an exponent in the range 0.4-0.6; second, it allows for a weighting of the harmonics proportional to their amplitude; and third, it produces better pitch estimates. Most of the mistakes that pitch estimators make, including SWIPE, are not random: they consist of estimations of the pitch as multiples or submultiples of the pitch. Therefore, a good source of error to attack is the score (pitch strength) of these candidates. A good feature to reduce supraharmonic errors is to use negative weights between harmonics. When analyzing a pitch candidate, if there is energy between any pair of consecutive harmonics of the candidate, this suggests that the pitch, if any, is a lower candidate. This idea is implemented by the negative weights, which reduce the score of the candidate if there is any energy between its harmonics. This feature is used by algorithms like SHR, AC, CEP, and SWIPE.

The frame-based systems can achieve a good performance, but it is time-consuming. In [24], they have proposed an efficient note-based system, which is mainly comprised of noted-based linear scaling (NLS) and noted-based recursive align (NRA). The system after post-processing can achieve 96.1and 0.211s in time. Regarding string matching, the algorithm used in [10] offers better performance for average cases of string matching than most other algorithms like brute force algorithm from O(mn) to O(kn) or O(n log(m)).

3. PROBLEM SPECIFICATION

The situation of not being able to find a song to which we have a melody running on our minds is known to everyone. Therefore, to overcome these situations, we are dealing with the wide problem of building a complete Query by Singing/Humming (QbSH) system in this project.

A Query by Singing/Humming system is able to output a ranked list of songs which matches the user hummed query [4].

Given recording of hummed tune, it will be able to identify the song being hummed.

This system needs to be robust to:

1. Poor humming
   - Wrong pitch
   - Wrong note duration
   - Wrong key
2. Noise and distortion

4. PITCH EXTRACTION ALGORITHMS (PEA)

4.1 SWIPE Pitch Extraction

A sawtooth waveform inspired pitch estimator (SWIPE) has been developed for speech and music. SWIPE estimates the pitch as the fundamental frequency of the sawtooth waveform whose spectrum best matches the spectrum of the input signal. The comparison of the spectra is done by computing a normalized inner product between the spectrum of the signal and a modified cosine. The size of the analysis window is chosen appropriately to make the width of the main lobes of the spectrum match the width of the positive lobes of the cosine. SWIPE, a variation of SWIPE, utilizes only the first and prime harmonics of the signal, which significantly reduces subharmonic errors commonly found in other pitch estimation algorithms. Tests indicate that SWIPE and SWIPE performed better on two spoken speech and one disordered voice database and one musical instrument database consisting of single notes performed at a variety of pitches. It will be shown that the types of signals for which the algorithm is optimized are periodic signals whose spectral envelope decays inversely proportional to frequency. An example of such a signal is a sawtooth waveform. This type of signal is the one that motivated the name of the algorithm: sawtooth waveform inspired pitch estimator: SWIPE. [7]

5. PATTERN MATCHING METHODS

5.1 Dynamic Programming Method

We illustrate dynamic programming using the edit distance problem. We assume a finite set of characters or letters, , which we refer to as the alphabet, and we consider strings or words formed by concatenating finitely many characters

from the alphabet. The edit distance between two words is the minimum number of letter insertions, letter deletions, and letter substitutions required to transform one word to the other. The edit distance of two strings, str1 and str2, is defined as the minimum number of point mutation required to change str1 into str2, where a point mutation is one of:

1. Change a letter
2. Insert a letter or
3. Delete a letter.

This algorithm averages pitch values within the 50 to 80 percentage of the duration of the note [2].

$$d\,ij = \min \begin{bmatrix} d\,i-1, j + w(ai,0)\ \text{(deletion)} \\ d\,i-1, j-1 + w(ai, b\,j)\ \text{(match/change)} \\ d\,i, j-1 + w(0, b\,j)\ \text{(insertion)} \end{bmatrix}$$

Figure 1. Edit Distance Formula

The initial conditions are given in Figure .

$$\begin{aligned} &d\,0,0 = 0 \\ &d\,i,0 = d\,i-1,0 + w(ai,0), i \geq 1 \\ &d\,0,j = d\,0,j-1 + w(0,b\,j), j \geq 1 \end{aligned}$$

Figure 2. Edit Distance Initial Conditions

Where w (ai, 0) is the weight associated with the deletion of ai, w (0, bj) is the weight for insertion of bj, and w (ai, bj) is the weight for replacement of element i of sequence A by element j of sequence B. The operation titled "match/change" sets w (ai, bj) = 0 if ai = bj and a value greater than 0 if ai bj. The weights used here are 1 for insertion, deletion and substitution(change) and 0 for match. As an example, if two pitch contour strings *UDDSSUD and *UDDSUD are compared, the edit distance is 1.

5.2 Dynamic Time Warping Method

The other efficient algorithm for pattern matching is Dynamic Time Warping, it allows a comparison between two signals with time variations (Figure 3), which is the case of a query sung by two different users, they will probably not sing at the same exact tempo, and that will also happen, as seen before, with the key. The basic idea of the algorithm is computing the path relating the samples of the two signals being compared which has minimum cost. This cost is computed by adding the differences between the values of the two samples being compared in a certain point of the path. Therefore, what is being made is, in the end, mapping between the two signals being compared. Depending on the application, different path patterns can be allowed. Interesting point to remark is the fact that the basic DTW algorithms usually try to match the two signals from start to end. In this system, that idea is useless, since trying to match a 10 seconds length of query with a complete song, which can last for several minutes and that makes no sense. For example, if a query contains the chorus of a song, the algorithm should allow the matching process to start from the chorus of the song, avoiding the initial part of the song. This can be improved by us in the future to allow the start and end samples of the query to be matched from any starting point in the song and can be implemented using the formula in (Figure 4).

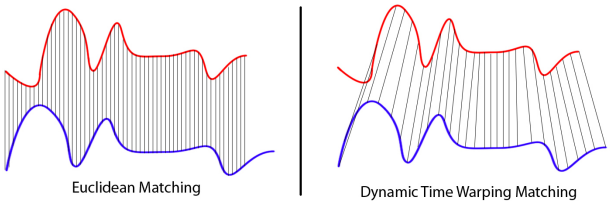

Figure 3. Euclidean Matching vs DTW Matching

$$\text{DTW} \begin{cases} \textit{Initialization:} \begin{cases} D(i,j) = 0 \ \ if \ \ i \in \{0.1\} \\ D(i,j) = -1 \ \ otherwise \end{cases} \\ \textit{Iteration:}\ D(i,j) = d(x_i, y_j) + \min(S_{path}) \end{cases}$$

Figure 4. DTW Formula

6. INFORMATION EXTRACTION AND ANALYSIS

Our database is essentially an indexed set of sound-tracks which are in .wav format. Currently, the songs in the database have been recorded and are both monophonic and some polyphonic audios for evaluation. These songs as well as the user query are presently restricted to be of only 10 seconds. The user-hummed query is hummed or sung by the user. In order to find the ranked list of the songs that matches this user input, this query is processed to detect its melody line. This output list of matching melodies can be used to retrieve the desired soundtrack.

The pitch extraction algorithm is used to extract the information from query and from the songs which are the input to the database. Then the comparison of this information (pitch contours) to find the distance between them is being done.

Dynamic Programming is used as a way to find the distance between the query and the database songs [21, 22]. SWIPE' pitch estimator is implemented for pitch extraction to test it's performance and efficiency. Moreover, it results in a better contour than FFT and Autocorrelation PEAs. Therefore, it results into a time series of pitch with varying pitch and duration.

6.1 Pitch Extraction

6.1.1 SWIPE' Pitch Extraction

We have implemented SWIPE' algorithm, variance of SWIPE (Sawtooth waveform inspired pitch estimator) in Python

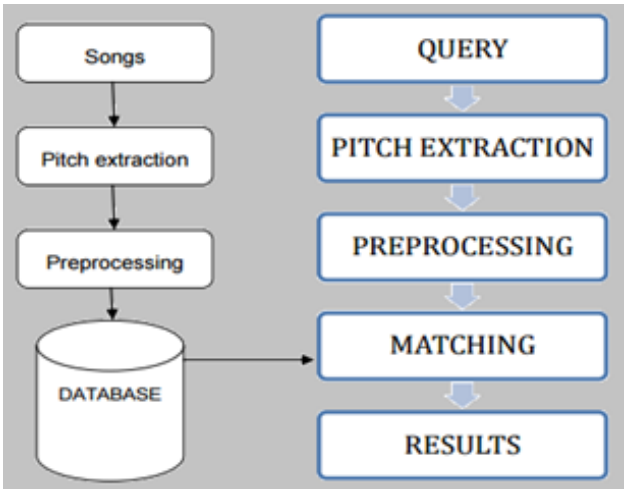

Figure 5. Basic Block Diagram

for estimating the pitch as the fundamental frequency of the waveform whose spectrum best matches the spectrum of the input signal. The comparison of the spectra is done by computing a normalized inner product between the spectrum of the signal and a modified cosine. The size of the analysis window is chosen appropriately to make the width of the main lobes of the spectrum match the width of the positive lobes of the cosine. We are able to fetch the frequencies, duration, pitch strength and get the pitch contours of the waveform [7]. The SWIPEP(X,Fs,[PMIN PMAX],DT,DLOG2P,DERBS,STHR) estimates the pitch of the vector signal X with sampling frequency Fs (in Hertz) every DT seconds. The pitch is estimated by sampling the spectrum in the ERB scale using a step of size DERBS ERBs. The pitch is searched within the range [PMIN PMAX] (in Hertz) sampled every DLOG2P units in a base-2 logarithmic scale of Hertz. The pitch is fine tuned by using parabolic interpolation with a resolution of 1/64 of semitone (approx. 1.6 cents). Pitches with a strength lower than STHR are treated as undefined.

SWIPEP function returns the times T at which the pitch was estimated and their corresponding pitch strength. [7] The resultant example pitch contour for an audio "qbhexamples" has been shown in the Figure 6.

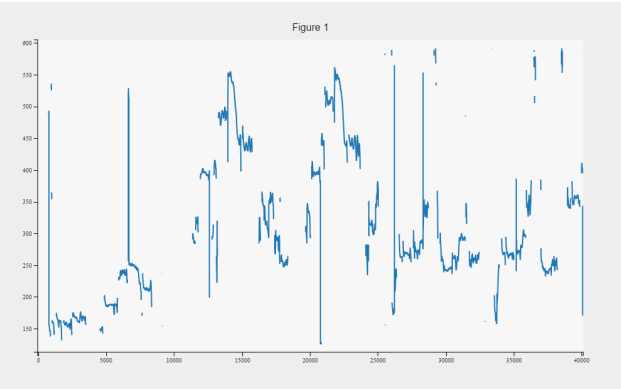

Figure 6. Pitch Contour of qbhexamples using SWIPE'

Our Database Schema consists of melodies wave files name, their location, frequencies, MIDI notes, pitch contour images and string patterns of each song. Thus, these notes of the database songs can be evaluated against the notes found in the query using an appropriate Pattern Matching Algorithm (Edit distance algorithm). The system can then return the best matched melody to the requested query.

6.2 Pattern Matching and Note Segmentation

After getting the output as the list of fundamental frequencies from the PEA, we have converted them into the midi notes by using the following formula:

$$p = 69 + 12 * log2(f/440) \quad (1)$$

These notes are rounded off and converted to whole numbers by eradicating the decimal part so as to segment the notes which is called quantization.

For these notes to be converted into a string or a pattern, after quantization, we are further converting them into the U (up), S (same) and D (down) string as per the Parson's code. Rather than using just these 3 levels to represent the notes of a melody. In this project, we are taking 5 levels of contour i.e., small u, big U, S, small d and big D in order to match the patterns with extra refinement. The idea is that the input is converted to the small u if the successive note is not significantly higher than the previous one and likewise for small d. So in our case, if the difference between the notes is less than or equal to 2 then we will assign small u's or small d's for ups and downs respectively. Otherwise, they will be assigned big u's and big d's respectively. For instance, if we have notes as: 60, 61, 61, 61, 64, 62, 58 then we will get the pattern as: uSSUdD The database songs are also converted to the u, U, S, d, D strings. The comparison of input to the list of songs is done using a pattern matching algorithm. Here, for pattern matching we are taking the edit distance algorithm of Dynamic Programming. These strings are then compared using the edit-distance algorithm. The distances for each song in the database is computed with respect to the recorded/hummed query.

7. IMPLEMENTATION AND INTERACTION

After getting the pitch contour of the hummed query from the SWIPE' function, we converted the frequencies to MIDI notes using the formula (1). These MIDI notes are then converted to the sequence of relative pitch transitions. The input is converted to the string with the representation of five letter alphabet which is u, U, S, d, D. A note is classified in five ways: If a note is same as the previous note then it is denoted by 'S', if it is slightly lower than the previous note then it is 'd' else for significantly lower it is 'D' and if it is significantly higher than the previous note then it is 'U' else if it slightly higher then it is 'u'. Now in order to compare the query with the database songs, Dynamic programming edit distance algorithm is used to match the pattern represented by five level contours, i.e., uUSdD of query and the database songs. As a result, it will return a ranked list of songs based on the distances after the pattern matching.

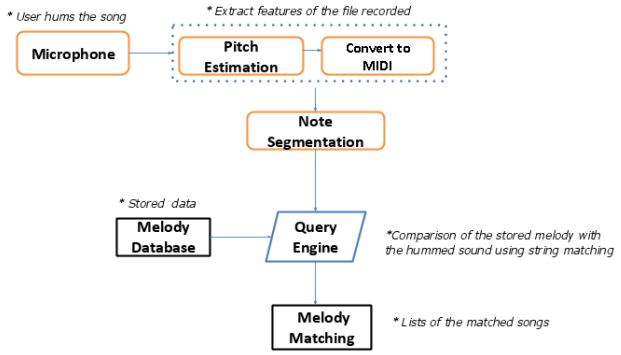

Figure 7. System Architecture

Song Type	Song Count
Hummed	4
Hummed with 1 syllable	11
Sung with words	8
Polyphonic Actual song (device)	2

Table 1. Dataset Categorization

8. SQLITE: DATABASE

SQLite is an in-process library that implements a self contained, serverless, zero-configuration, transactional SQL database engine. It is an embedded SQL database engine. Unlike most other SQL databases, it does not have a separate server process. It also reads and writes directly to ordinary disk files. A complete SQL database is having 3 tables, one is for audio features of database songs, the other is for audio features of the hummed query dn the third one is the result table of matched songs. They have attributes such as Name of the song, frequencies, durations, distances, MIDI notes, pattern, path of wave file and the generated pitch contour image of the song. [1] The SQLite3 can be integrated with Python using sqlite3 module. To use sqlite3 module, we first created a connection object that represented the database and then we created cursor object which helped in executing all the SQL statements. [5] Database schema for this project has been shown in the Figure 8.

8.1 Dataset

In our dataset, we have taken four song types in our dataset as hummed, hummed with one syllable, sung with words and polyphonic actual song. For each song type we have taken some set of melodies in the dataset which are listed in Table 2.

First table is 'audioFeature' that store the dataset of songs (hummed and sung), it has been created using the SWIPE feature extraction method and then converting the frequencies to MIDI notes and creating their patterns and then storing these attributes of a song in the database along wth the pitch contour image as BLOB and respective edit distances of dataset songs from the hummed query.

Second table is 'hummedFeature' that stores the attributes for a hummed query icluding pitch contour image as BLOB that is recorded by the user from UI.

'matchedSongs' is another table that stores the ranked list of songs on the basis of distances of hummed and dataset songs that are generated based on the matching of hummed query and dataset of songs.This table is used to fetch the songs stored in this table and display them on the UI.

We have taken 25 melodies in the database yet because the pattern matching of the melodies is computationally slow. And this computation time is proportional to the number of songs in the database because the pattern of the query is matched with every song in the database. The acoustic query, which is typically a few notes whistled, hummed or sung by the user, is processed to detect its melody line. The database is searched to find those songs that best match the query. The system returns a ranked set of matching melodies, which can be used to retrieve the desired melody. The dataset also contains all the melodies hummed or sung but with a time duration of 10 seconds, equal to the duration of the hummed query.

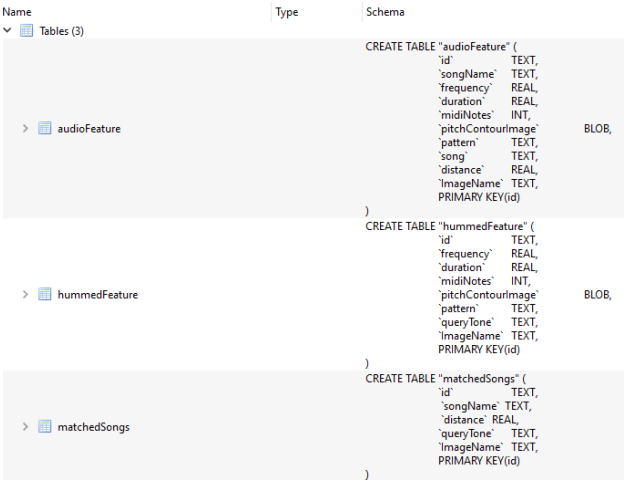

Figure 8. Database Schema

8.2 PyQT: Graphical User Interface

The interface is created using the standard Pyqt4 library available in Python. For designing, we have used Pyqt GPL Designer version 5.5.1. This designer interface creates a corresponding XML file after we put/use various GUI components. Then in order to use this XML file into our python code, we have performed the conversion using the pyuic4 command.

Following is the little description of every component of the GUI created. On the top it has two buttons to record and play the recorded hummed query. Below that, there is another button for feature extraction that is done using SWIPE, once it is done, we have annother button besides it to plot Query, we have kept the space on the right side for showing the pitch contour of the recorded sound. Next this recorded query is matched with the songs stored in the database and a list of closely matched songs are then displayed in the left bottom half of the GUI. In the right bottom half, this GUI depicts the pitch contour of the matched

song if user wishes to see it. There are also buttons to clear their plots. The images for the pitch contour are shown based on their paths where they are stored on the system and the database also stores this information for every song. Interestingly, the UI initially does not contain the ranked list of songs but once user click on 'Find matched Songs' then a list appears on the screen that are ranked on the basis of best to worst matching with hummed query.

1. The Figure 9 represents the UI of QbSH project which shows up on initial loading.

2. Once user starts interacting with it , following steps are performed:

- User records and play the song/hummed query and its respective pitch contour is displayed on the screen on click of Plot Pitch Contour. Figure 10 shows this view.

- Now the user click on Find matched songs button to view the results that shows ranked list of matched songs with database. Figure 11 shows the results.

- If user wishes to see the pitch contour of a selected song, then he can select the radiobutton of a song and click on 'Plot Result Pitch' that display its pitch contour on the right small window as shown in Figure 12.

- Now user can play the selected song too.

- User can also clear the plots for hummed and selected song as shown in Figure 13 and 14 [3]

Figure 9. QbH: Graphical User Interface

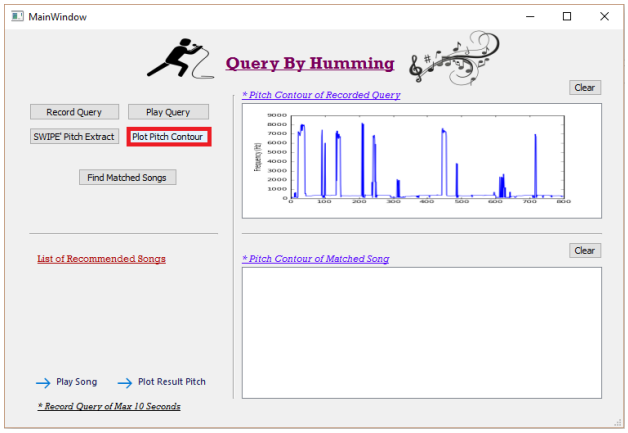

Figure 10. GUI: Plot hummed query pitch contour

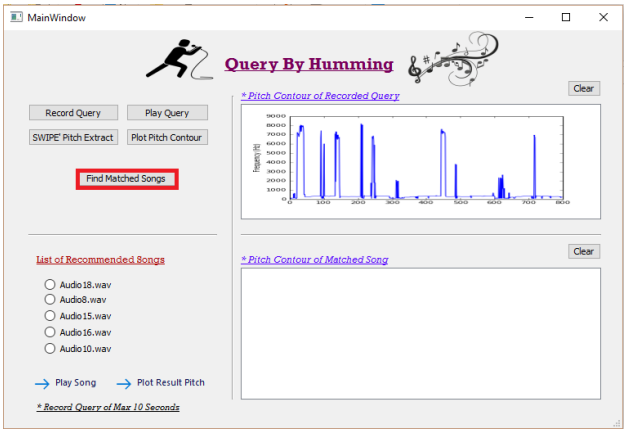

Figure 11. GUI: Matched Results

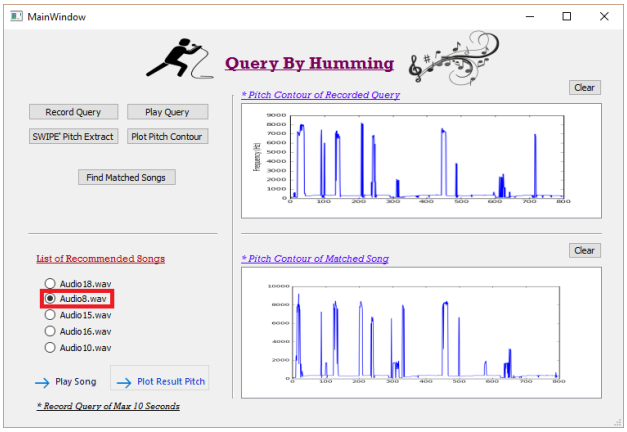

Figure 12. GUI: Plot selected song pitch contour

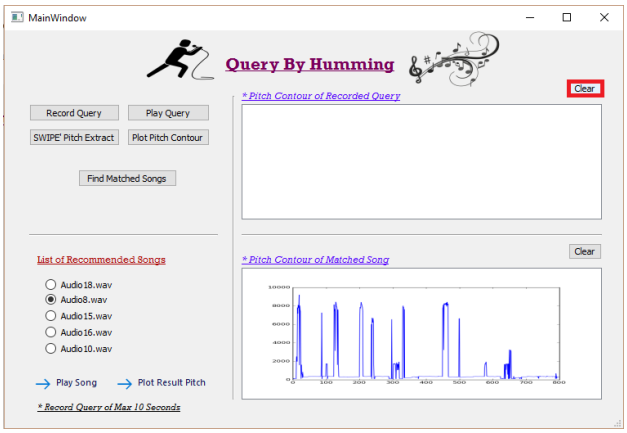

Figure 13. GUI: Clear hummed query plot

9. SYSTEM SPECIFICATIONS

In our project Query by Singing/Humming, for the front end we are using Python (with PyQt binding module) for GUI, implemented SWIPE in python to analyze frequencies, duration, pitch strength and get the resulting pitch contour. In the backend, SQLite is used as our database for storing audio files along with their features. For the report generation and documentation, we are using Latex, PDF and Microsoft Word.

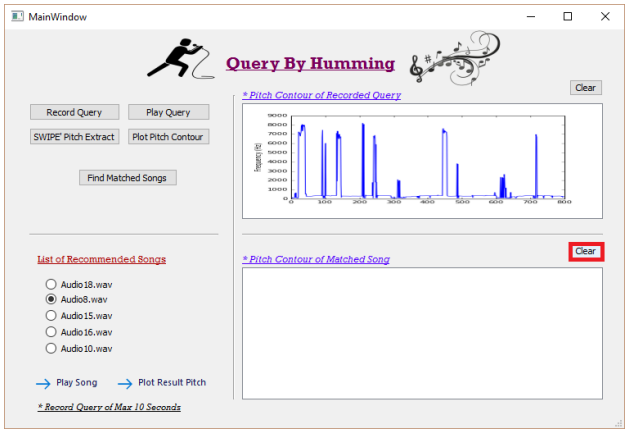

Figure 14. GUI: Clear selected song plot

Query Type	SWIPE'(in Perc)
Hummed	75
Hummed with 1 syllable	85
Sung with words	80
Polyphonic song (device)	70

Table 2. Accuracy (in Percentage) of pitch extraction method implemented

10. EVALUATION

This section provides a detailed and quantitative analysis by comparing the performance of various melodies by the SWIPE' Pitch Extraction algorithm. The evaluations are performed on the this QbSH system with our database. In this database, there are 25 recorded melodies hummed or sung by 3 people. In the evaluation, each query is retrieved from our song database containing the pattern of those 25 melodies or wav files. All the algorithms run on the computer with a 2.20 GHz core i5 CPU and 8 GB memory. In the experiment, the computation time contains the time of tracking pitch as well as the time consumed in the pattern matching algorithm. The time consumed in pattern matching is high because the pattern matching is dependent on the size of output from the PEA. We compared the result on the basis of humming query, humming with one syllable, query sung in words.

11. RESULTS

We have taken various song types and tested song 10 times with each type. These are hummed song, other is hummed with a single syllable, also song sung in words and actual song played from a music device. This test with each of the type is performed with SWIPE pitch extraction algorithm. The comparison of accuracy of results we got after matching query with the songs is listed in Table 3.

12. FUTURE DIRECTIONS AND RELATED WORK

1. 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 errors due to dropped notes in tunes or variation in tempo are more common errors. Tuning the key-search so that it is more tolerant to drop-out errors, for example, may yield better results.

2. The melodic contours of the source songs are currently generated automatically from the pitch extraction 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.

3. We will be using Dynamic Time Warping for pitch extraction which is more efficient than dynamic programming Edit distance algorithm since it allows the comparison between two signals with time variation which is the case of a query sung by two different users, they will probably not sing at the same exact tempo, and that will also happen, as seen before, with the key. The only problem is that it start matching the song from start to end so for that purpose, it can be modified to allow the start and end samples of the query to be matched from any starting point in the song.

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 trade-offs involved. An important issue resides precisely where to draw the line between notes that are a little higher from the previous note and those that are much higher.

5. The Graphical user interface of QbSH has been made in PyQT with many features included to record/ play query, perform feature extraction and match songs using Edit distance algorithm that renders the result of ranked list of matched songs. Also we can view/ clear pitch contour plot of the hummed query and matched selected song. We can also enhance our UI by adding the feature of progress bar in future to notice the progress of ongoing operation on the UI.

6. We are in the process of developing this as a web application as well so that it can be accessed on server publically.

7. Previous work on efficiently searching a database of melodies by humming seems to be limited. Mike Hawley 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 published a paper on retrieving melodies using a hummed melody in a Japanese journal, but we were unable to locate a translated version [10] .

13. REFERENCES

[1] About SQLite. https://www.sqlite.org/about.html.

[2] Dynamic Programming Algorithm (DPA) for Edit-Distance. http://www.csse.monash.edu.au.

[3] PyQt Tutorials. https://wiki.python.org/moin/PyQt/Tutorials.

[4] Query By Humming wikipedia. https://en.wikipedia.org/wiki/Querybyhumming.

[5] SQLite Python Tutorial. from tutorialspoint.com.

[6] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. *Efficient similarity search in sequence databases*. Springer, 1993.

[7] Arturo Camacho and John G Harris. A sawtooth waveform inspired pitch estimator for speech and music. *The Journal of the Acoustical Society of America*, 124(3):1638–1652, 2008.

[8] Roger B Dannenberg, William P Birmingham, George Tzanetakis, Colin Meek, Ning Hu, and Bryan Pardo. The musart testbed for query-by-humming evaluation. *Computer Music Journal*, 28(2):34–48, 2004.

[9] Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C Smith. Query by humming: musical information retrieval in an audio database. In *Proceedings of the third ACM international conference on Multimedia*, pages 231–236. ACM, 1995.

[10] Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C Smith. Query by humming: musical information retrieval in an audio database. In *Proceedings of the third ACM international conference on Multimedia*, pages 231–236. ACM, 1995.

[11] Stephen Handel. *Listening: An introduction to the perception of auditory events.* The MIT Press, 1993.

[12] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371, 2003.

[13] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[14] Edmond Lau, Annie Ding, and J Calvin. Musicdb: A query by humming system. *Final Project Report, Massachusetts Institute of Technology*, 2005.

[15] Lie Lu, Hong You, HongJiang Zhang, et al. A newapproach to query by humming in music retrieval. In *ICME*, pages 22–25, 2001.

[16] T Merrett. Query by humming. *McGill University, Montreal*, 2008.

[17] Alan V Oppenheim, Ronald W Schafer, John R Buck, et al. *Discrete-time signal processing*, volume 2. Prentice hall Englewood Cliffs, NJ, 1989.

[18] Steffen Pauws. Cubyhum: a fully operational" query by humming" system. In *ISMIR*, 2002.

[19] M Anand Raju, Bharat Sundaram, and Preeti Rao. Tansen: a query-by-humming based music retrieval system. In *Proceedings of the national conference on communications (NCC)*, 2003.

[20] S Dutta Roy, Preeti Rao, and Ameya S Galinde. Contour-based melody representation: An analytical study. In *Proc. National Conference on Communications (NCC)*, pages 536–540, 2004.

[21] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, Feb 1978.

[22] Amiya Kumar Tripathy, Neha Chhatre, Namrata Surendranath, and Manpreet Kalsi. Query by humming system. *Int. Journal of Recent Trends in Engg*, 2009.

[23] Pau Tur Vallés. Query by humming. 2014.

[24] Jingzhou Yang, Jia Liu, and Weiqiang Zhang. A fast query by humming system based on notes. Citeseer, 2010.

[25] Yunyue Zhu and Dennis Shasha. Query by humming: a time series database approach. In *Proc. of SIGMOD*, page 675, 2003.