Query-by-Humming System to Illustrate Signal Processing Field for Prospective Students

Benjamin Peter Leonard
Department of Electrical Engineering
Northern Illinois University
Dekalb, USA
Z1865180@students.niu.edu

David Goldberg
Department of Electrical Engineering
Northern Illinois University
Dekalb, USA
Z1634095@students.niu.edu

Nathan McDonald
Department of Electrical Engineering
Northern Illinois University
Dekalb, USA
Z1788059@students.niu.edu

***Abstract*—Query-by-humming is a content-based music information retrieval system where a user hums part of a tune into a microphone, the audio is sampled and processed by a computer and compared against a database of stored songs. The tune hummed by the user is identified, a matching score is provided, and the actual song is replayed back to the user. This system has software framework that allows for real time use, and a graphical user interface which displays the hummed signal, matching information, and has buttons to control the application.**

Keywords-Query-by-Humming, Graphical User Interface, Software Framework

I. INTRODUCTION

High school students often have a limited view of all the fields of electrical engineering. This limited view may cause students to overlook electrical engineering as a career. This project highlights the field of signal processing by developing a real time Query-by-Humming system paired with an informative interactive GUI (graphical user interface). This system is divided into 4 sub-components: Audio processing framework and real time usage, feature extraction, matching & decision making, and the GUI. Python was used to code the feature extraction, matching & decision making, and the GUI. While C++ was utilized as a framework allowing real time audio analysis.

II. METHOD AND CODING

A. *Audio processing Framework*

The audio processing framework is built using C++ for the main portion of the code. The Application programming interface, and the graphical user interface control the main program and are implemented in python. The main program has three persistent sub processes that handles the extracting, matching, and decision steps. Each of these processes pulls code from the user creating an instance of this program. It can take code in the form of either a C++ executable or a Python script. This framework model allows for iterative design, as well as being a basis for a real time application. The API/GUI communicate with the main program using named pipes, a method for inter process communication. The Main program communicates with the subprocesses using a similar method. In the case of a real time application, Audio data comes into the API/GUI, and is piped directly to the extraction process. The features are extracted using the user supplied code. The resulting extracted features are piped directly to the matching process, where they are compared against the existing database. The output of this comparison is piped to the Decision process, where the correct match will be decided. This result is then communicated to the main program, and then subsequently communicated to the API/GUI, which will then display the graphics informing the user of the match.

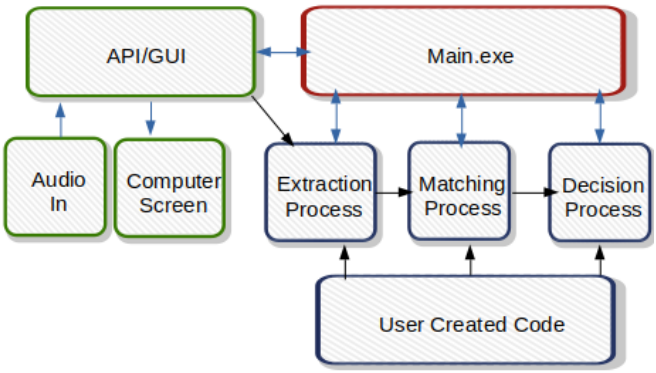

Figure 1: Block Diagram of Audio Processing Framework

B. *Feature Extraction Algorithm*

The feature extraction process begins by analyzing the energy of each frame from the time series audio data. If the energy is greater than a minimum value (chosen from experimentation), the frame of data has its pitch extracted and the value is stored into a list. If the value of energy is below the min energy, the frame is discarded. An average over the extracted pitches is computed and the average value is stored into a list. A frame with energy less than the min value corresponds to when the user ends humming a particular tone. Taking the average allows us to account for fluctuations in raw pitch values and obtain an approximate value for the intended tone.

Pitch extraction is done by using the python library 'Librosa' [1]. The function used is an implementation of the

YIN autocorrelation method which estimates the fundamental frequency of a signal, which corresponds to pitch [2].

Once there are two values in the average pitch list, the last entry is compared with the second to last entry each time the list is updated to generate another list referred to as the up-Down List. If the average pitch value increase, a "1" is placed into the list. If the value decreases a "-1" is placed, and if the values are within a certain threshold, they are considered the same value and a "0" is placed.

In summary the feature extraction algorithm converts raw time series audio data into a series of '1s', '-1s', and '0s'. These values directly represent the relative pitch of the hummed input query.

C. *Matching & Decision Making*

As the up-Down list is being generated from the feature extraction algorithm it immediately begins the matching process by being compared to a database of preprocessed songs with a function called fast dynamic time warping [3].

Fast dynamic time warping (DTW). This technique makes an informed low-resolution estimate of the DTW path, then increases the resolution over the initial path and calculates the Euclidian distance between two time series data sets by generating a lowest cost matrix. This method is much faster than a true full DTW cost matrix and allows us to calculate the distance between the hummed input and each of the stored songs in the database in real time.

The name of each song and the corresponding distance is stored inside a tuple. The tuple is sorted in descending order and saved. The shortest distance calculated directly corresponds to the most likely match of the input hum and is stored in the first entry of the tuple. Simple string-matching logic is performed that checks the first entry of the tuple for the name of the song and it prints the name of the song and a matching score as output to the user.

D. *Graphical User Interface*

The GUI (graphical user interface) is how the user will interact and learn about signal processing. Only relevant information regarding the entire process is shown. Real time audio input (Figure 2), graph of the extracted pitch & the averages pitches (Figure 3), graph of the relative pitch of the hummed input (Figure 4), graph of the relative pitch of the corresponding song in the database (Figure 5), and a text box which displays the name of the matched song.

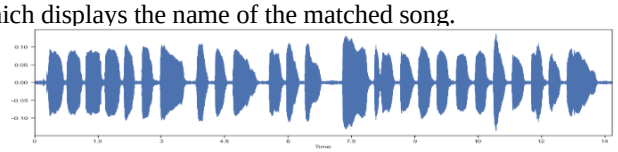

Fig 2:Real time Audio Input

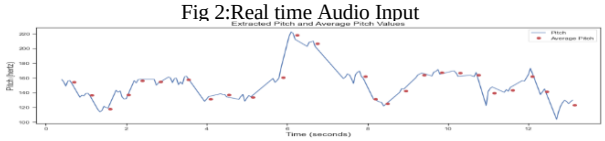

Fig 3: Extracted Pitch and Average Pitch

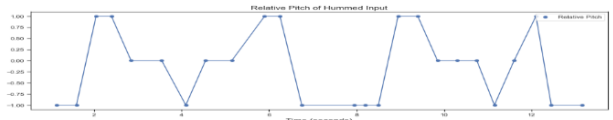

Fig 4: Relative Pitch of Hummed Input

Fig 5: Relative Pitch of corresponding Song

III. RESULTS

The goal for the performance of the query-by-humming system was to have an overall probability of error less than 10%, and to be able to function for male, female, and children users. All performance goals were met, however due to covid-19 we were not able to obtain a proper diverse sample size. The testing was conducted with n=45 samples from 3 adult males, 1 adult female, and 1 female child.

Table 1: Results Breakdown

Song Name		**Prob. Of Error**
Happy Birthday		9.09%
Jingle Bells		9.99%
Itsy Bitsy Spider		12.5%
Twinkle Twinkle Little Star		0.0%
Mary Had a Little Lamb		9.09%
Total Prob of Error		*8.88%*

The 5 songs were selected with the intent of picking songs with the most likelihood to be known by the most amount of people. Virtually everyone born and raised in the United States hears each of these songs as a child and knows them by heart.

IV. CONCLUSION

In this paper we present a real time Query-by-Humming system to illustrate signal processing for prospective students.

ACKNOWLEDGMENT

The authors would like to thank Dr. Benedito Fonseca and Dr. Mansoor Alam, from the Department of Electrical Engineering at Northern Illinois University, and Sandhya Chapagain for their constant and continuous technical support throughout the development of the Query-by-Humming system.

REFERENCES

[1] Brian McFee, Vincent Lostanlen, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thomé, … Taewoon Kim. (2020, July 22). librosa/librosa: 0.8.0 (Version 0.8.0). Zenodo. http://doi.org/10.5281/zenodo.395522

[2] De Cheveigné, Alain, and Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music." The Journal of the Acoustical Society of America 111.4 (2002): 1917-1930.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] Stan Salvador, and Philip Chan. "FastDTW: Toward accurate dynamic time warping in linear time and space." Intelligent Data Analysis 11.5 (2007): 561-580.