Earworm

Authors: Anja Kalaba, Wenting Chang, Nolan Hiehle

Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— A system capable of identifying a song based on a simple section of melody sung into a microphone by a user. This will be implemented with a combination of matching against preformatted MIDI files using melodic contour analysis where available, and using machine learning methods on a cross-similarity matrix of a song's chroma feature representation for songs without available MIDIs.**

***Index Terms*— Chroma feature, convolutional neural network (CNN), cross-similarity, dynamic time warping, machine learning, melodic contour, MUSART, query by humming, Roger Dannenberg, signals, sound, theme extraction**

I. Introduction

THIS project will be making an application that will match a human's sung or hummed input to the existing song that they were trying to identify. Often times a person will have a tune stuck in their head and will not have any way to identify it unless they can remember some lyrics to search, so the application we are trying to build will try to solve this.

There are currently very few applications that will recognize a song through human singing or humming due to the complexity of the problem. The two most common apps to achieve something to this effect are Shazam and SoundHound. However, Shazam can only match to actual songs rather than human input, and SoundHound has poor performance and vague technical details. The approach we will be taking in our project borrows concepts from the query by humming project and chroma feature analysis with convolutional neural networks, which has been done for cover song identification. These two approaches will take parallel paths to match the input to an existing song. The system goals and metrics are to achieve 75% accuracy on matching singing or humming to a song and to match in under one minute.

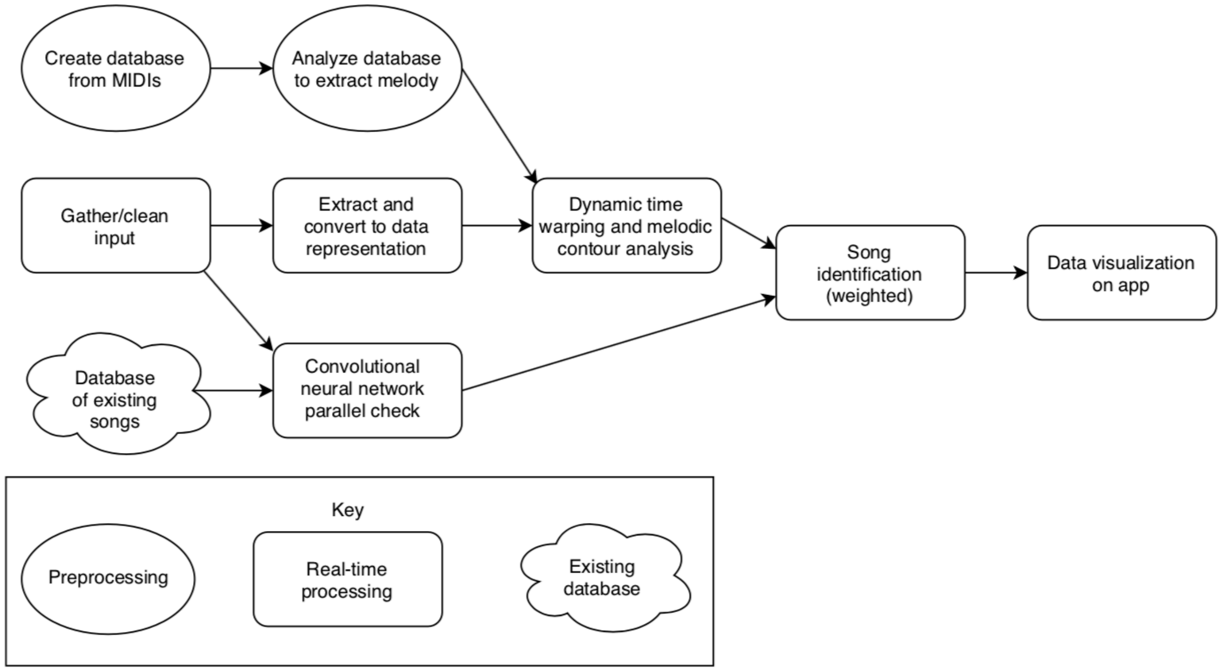
II. Design Requirements

The project will be packaged as an app. This adds some constraints to what the user's experience should be like. From start to end, opening the app should lead to a straightforward interface, from which the user sings into the microphone. Qualitatively, the user should not have to extensively worry about the sound quality or noisiness of their environment, but it should be reasonable. For example, if there are multiple other voices adding to the input captured by the microphone, the accuracy of the app is guaranteed to decrease. We will test our ability to filter out audio input noise by trying out multiple queries which are all consistent along the dimensions of singer, song, time frame, and instead vary the environment of recording.

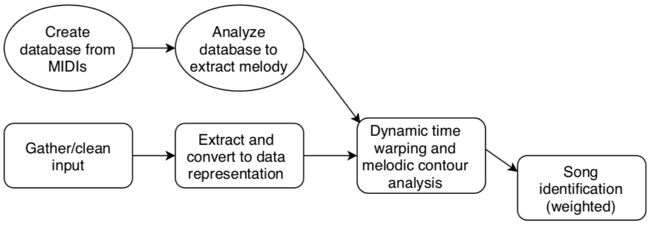
Next, as part of the experience, past a seamless vocal input, the user should not expect to wait more than 1 minute for the app's response. From our technical point of view, we hope to constrain the system to taking only 1 minute for completion and decision. All algorithmic components to this pipeline include taking in and bucketing the input to sampled frames that are empirically optimized, in parallel taking in the input and running it against chroma feature extraction, running the melodic contour and DTW algorithm against all possible pairings of (sung_input, library_song_midi), in parallel creating a cross-similarity matrix for all possible pairings of (sung_input_chroma, library_song_raw_chroma), sorting the outputs of each stream by closest match, averaging them and selecting a final song output list.

Finally, the user should see an ordered list of the top 3 songs that their sung input potentially matches against. The last critical constraint we plan for is obtaining 75% accuracy, with a success defined as the truly intended song, let's call this the *target*, appearing in the top 3 list. We plan to test this across multiple singers, songs, and song time frames, and begin with a small database of about 10 songs. Qualitatively, the user should see a simple to understand list, and on top of this, a data visualization image that indicates why the song matches that were given were made. The visual will explore indicating which parts of the user's query lined up with the matched songs, and to what degree the music lined up.

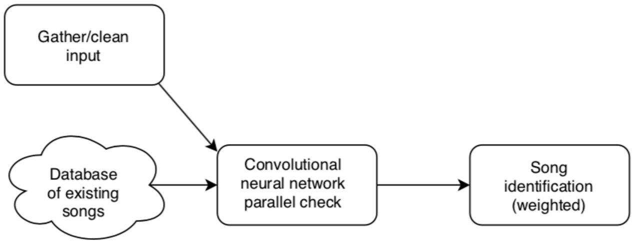
III. Architecture and/or Principle of Operation


Create database from MIDIs
Analyze database to extract melody
Gather/clean input
Extract and convert to data representation
Dynamic time warping and melodic contour analysis
Song identification (weighted)
Data visualization on app
Database of existing songs
Convolutional neural network parallel check
Key
Preprocessing
Real-time processing
Existing database

(a)


Create database from MIDIs
Analyze database to extract melody
Gather/clean input
Extract and convert to data representation
Dynamic time warping and melodic contour analysis
Song identification (weighted)

(b)


Gather/clean input
Database of existing songs
Convolutional neural network parallel check
Song identification (weighted)

(c)

Fig. 1. System picture. (a) overall system. (b) zoom-in of dynamic time warping/melodic contour analysis. (c) zoom-in of chroma feature analysis and CNNs

The flow of data begins with a user singing into a microphone. That input will be put through a filter to reduce background noise, then put through two different branches that use different matching methods to maximize the probability of getting a match.

The section in Fig. 1(b) shows the dynamic time warping and melodic contour analysis branch. The pitches and rhythm will be extracted from the filtered input and will be stored as a list. The existing songs will be pre-processed from MIDI files into the same representation, and then the two will be compared to find a match through modified dynamic time warping.

The section in Fig. 1(c) shows the chroma feature analysis branch. The input will be converted in a chroma representation to compare against that of the existing song. A convolutional neural network will check for patterns in the cross-similarity matrices of the sung input and songs in the library.

Once the song is identified or the system has maxed out on time and was unable to identify the song, the app will show a visualization of what was matched and how it was matched. An example of how this might be shown is displayed in Fig. 2.

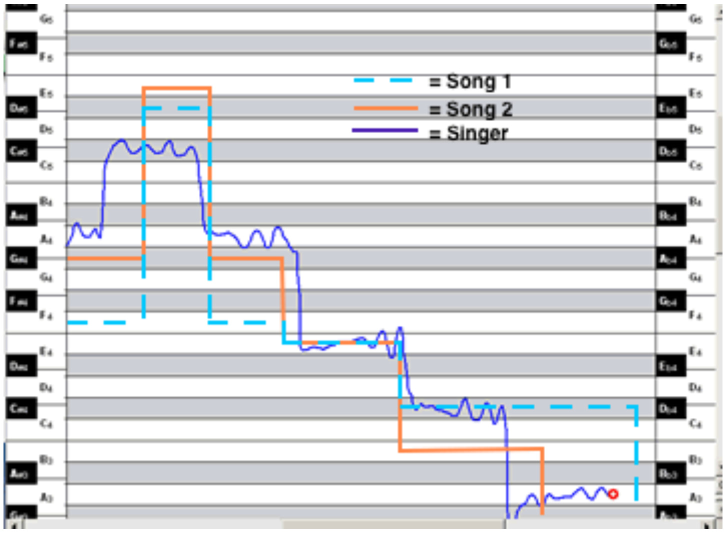


Fig. 2. Example of data visualization

IV. Design Trade Studies

There are many parts to our design, including a parallel sub-track to compute the decision in 2 ways. Design decisions like these were made on the basis of prioritizing robustness and putting most emphasis on the accuracy metric. In general, our trade offs were between accuracy and time of implementation or algorithmic time complexity, and helpfulness vs. time of implementation, and amount learned vs. time of implementation. Key categories that were logistically debated were the following:

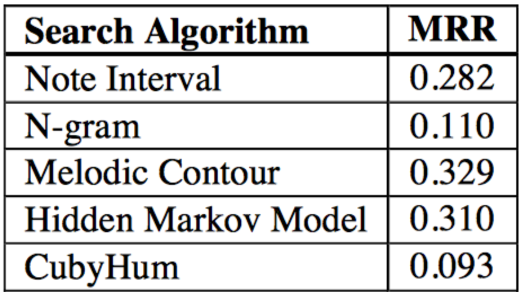
- deciding on the algorithm to use for musical processing and comparison
- deciding how large to make the database
- deciding which parts of the project to borrow and which parts to implement ourselves
- deciding which test dimensions were the most important to analyze.

The design decision progression is outlined for the above compartments of the overall system in the following sections, along the debated lines of importance as mentioned before the list of key categories.

A. Algorithm and Database Size

We consulted the *Query By Humming* paper which did a proper testbench to compare multiple algorithms. As shown by their results in Table 1, the most successful algorithm was the Melodic Contour matching one, which we have selected to use in our project. The test suite for these results was fairly randomized and uniform, so it more closely resembled our expected range of input and usage. We found in another paper for it to have a mixture of 100% accuracy, also measured with the "contained-in-top-3" success definition, and 80% accuracy, for varying test suites of rock songs and jazz songs, respectively [4]. Since both of these values are above our target value of 75% for accuracy, this algorithm seemed sufficient.

TABLE I. Table of MRR values for various algorithms from MUSART test bench



Search Algorithm	MRR
Note Interval	0.282
N-gram	0.110
Melodic Contour	0.329
Hidden Markov Model	0.310
CubyHum	0.093

A heavy point of consideration was the tradeoff in algorithmic time complexity and accuracy. While the Melodic Contour algorithm is accurate to the degree we wish to see, it is also significantly slow, as it is a linear search through the database per length of the input string. To hone in on a region of higher performance, we consult Fig. 3, wherein we notice that there is a plateau in algorithmic performance as a function of songs in the database. We also logically noted that having a smaller database would decrease the general search time for a linear algorithm, helping us achieve our goal of under a minute potentially, so we were set on starting with a small database of roughly 10 songs. We also found in the paper an estimate of about 23ms on a 1.8GHz Pentium 4 Processor per song match for an algorithm that is a *O(mn)* where m and n are string input lengths, as melodic contour will be since it tries every window frame for overlap. Considering the added runtime for the DTW on top of this, since it will not be constant but also a linear search of these lengths, we get $O(mn^2)$ so 529ms. Thus, to obtain our under a minute goal, our database would need to meet an optimized version of (1), with T_max set to 1 minute measured in milliseconds.

$$\text{T\_max} \geq 529*\text{database\_size} \tag{1}$$

A third consideration was to include the CNN parallel component in the algorithm pipeline. Fortunately, training the network is the biggest bottleneck, and we can make choices in our network architecture to ensure that, once trained, the network can emit classifications in a reasonably short amount of time. Our database of waveform-format songs can be preprocessed into chromagrams, so that the only calculation necessary will be:

1. Turning a sung query into a chromagram
2. Creating a cross-similarity matrix
3. Running the classifier (CNN).

Since the maximum length of a song is about 3-4 minutes, and the maximum length of a sung query should be about 30 seconds, our cross-similarity matrix has some reasonable bounds on its size: we can expect that somewhere under 10,000 different distances need to be computed to create the matrix. However, as our database size increases, the model will need to be updated to account for new songs—there is no way to add a song to the classification output of the neural network without retraining the entire thing.

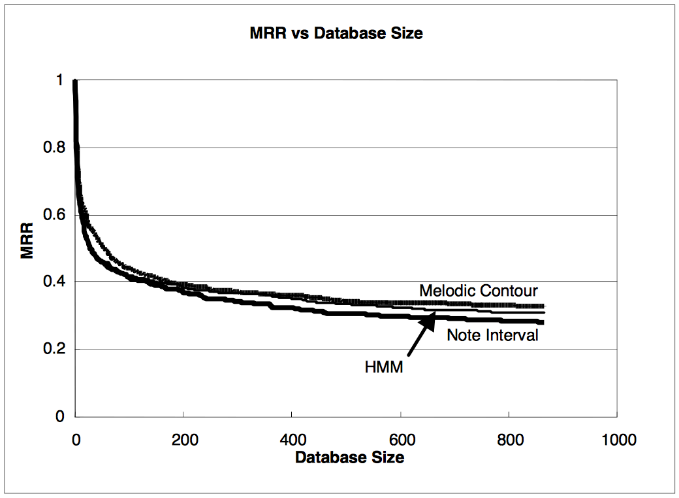


Fig. 3. Plot of MRR, a measure of accuracy which when closer to 1 indicates greater accuracy, vs. database size.

A final note about the algorithm chosen is that the decision to keep the 2 pipelines in parallel, as opposed to sequential, might be modified later. We do not have personal values for what our implemented versions of the algorithms will look like with respect to accuracy and time complexity. So, in creating both pipelines, we might find a glaring performance gap which might dictate that having a 2-stage search for database refinement would be better to meet our accuracy and complexity constraints. That is, putting the 2 pipelines in sequential order could look something like having a fast but less accurate decision be made to filter out obviously incorrect songs, which is then fed to a slower but more accurate part of the pipeline which can be sped up on the smaller database.

B. Self-implemented vs. Borrowed Subsystems

Another consideration was deciding which parts would be borrowed and which parts would be implemented by us. We decided after looking at the extensive algorithm for MIDI processing in the Meek paper that the tradeoff for time of implementation was not worth doing that portion of the pipeline by ourselves [5]. Since we also were not able to interface with their code, we decided in the end to do melody extraction by hand, with the help of Professor Roger Dannenberg's graduate student.

The Chroma feature extraction portion was also not seen to be worth self-implementing, since successful MATLAB toolboxes existed already [2].

We do not have any equations to describe our decision-making process for this category. It was instead a qualitative evaluation based on thinking about which portions of the pipeline were most vital to the integrity of the algorithms, which parts we were most interested in learning about, and which parts would take unnecessarily long amounts of time to complete. With this in mind, we pruned MIDI file pre-processing and chroma feature extraction from our list of duties.

C. Test Dimensions

Here we discuss the test dimensions that were of most concern for our evaluating project success. The dimensions we could enumerate were

- different singers
- different songs
- different time frame within a song
- duration of query
- noise-levels of environment of query

and we plan to create a test suite to vary each in an isolated manner to see its performance. Since we have at least three test users, it should be easy to compare a sung sample from each of us and ensure that it can generate accurate results a reasonable amount of the time. Testing different songs will also be relatively straightforward once we have entered these songs into our database or trained our model on them. Once these tasks are complete, we will simply be able to confirm that sung samples of melodies from each of the songs in the database can return a match. We can also test queries for the "main melody" of a song as well as other, less important musical sections, test with shorter queries that contain only a fragment of a melody, and test with queries in different environments--ideally, this would work even in a noisy, outdoor, public space.

An automated regression testing suite can easily be constructed once we have recorded a handful of queries from each of us for several songs each. These canonical queries could span the range of these potential limiting factors and ensure that any changes to our algorithm do not affect our correctness guarantees.

V. System Description

A. Branch 1: Dynamic Time Warping and Melodic Contour Analysis

The first component of our pipeline will be discretizing the input and applying a melodic contour / dynamic time warping (DTW) algorithm to the input. The first objective is to create a sequence of notes/pitches from the input. The input is to be sampled, with some experimentally determined sampling rate but we will most likely begin with 100ms [4]. For each frame that the sample audio is broken down into, a representative mode frequency is to be assigned to that frame, and thus a sense of bucketing the input is applied; we can call this creating a melodic contour on the input. The representative frequency for each sample frame is to simply be assigned the pitch of the note that occupies the most space in the frame. From this we obtain our sequence of n pitches for the query.

This same processing input, of creating a sequence we call the melodic contour, is to be applied to each song in the database as well. This procedure is to have been done ahead of time, so that with each song in the fixed database, a sequence of the melodic contour already exists as data to sift through.

Next, upon the input, the same form of segmentation is to be done to multiple versions of the original input. These versions will all differ by a scaling factor done in a sense to recreate a slower or faster version of the song. This is to account for the possibility that the user did not sing the song at the same tempo as the original input target that is potentially located in the database. The scaling factors we consider are within the range of 0.5 to 2.0, with the actual total number of versions we would want to check for being experimentally determined in order to not delay the algorithm's time complexity. In fact, this scaling will be done towards the second half of refining the implementation to increase the accuracy.

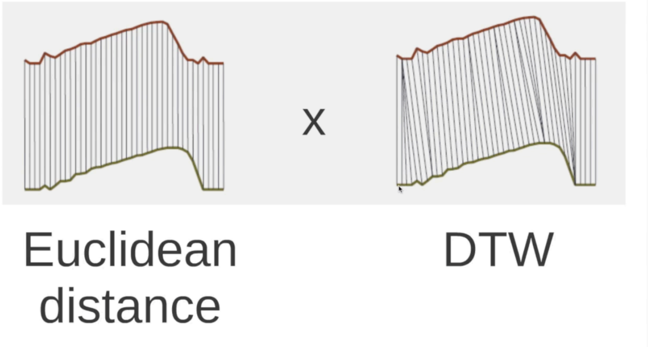


Fig. 4. A visual of how the data across 2 time series are lined up differently between Euclidean distance and DTW comparison methods.

Next, we describe the concept behind dynamic time warping. The use of this comparison technique is to find a more intelligent measure of distance between 2 time series data sets. Instead of a simple naive Euclidean distance approach, DTW uses a matrix with each song along each axis to instead find which parts of one series more closely resemble other parts of the other series. This can be visualized in Fig. 4, which clearly shows that DTW lines up parts of the data series which are more similar. Overall, using this approach is more helpful in finding melodic similarities considering that factors such as temporal alignment or scaling (which preserves shape but not length, but with DTW would be noticed), which should not impact the accuracy, can now be naively dropped as influencing the search. Thus, overall the match is more based on melodic integrity.

Within our framework, the DTW algorithm, which is computed using a distance matrix and from this distance matrix finding the optimal/cheapest path to the final point, will be applied between the pitch vectors from the input and each song in the database. Moreover, it will be applied to each pair of input and database song for each n offsets of the data points, in a windowing effort. Then, with all these comparisons, an overall distance score will be computed, and the scores will be sorted in by smallest distance for all the possible combinations of matches between audio input and database song for all possible offsets. The top 3 smallest distances for distinct songs will represent the closest matches of the query to an element in the database.

B. Branch 2: Chroma Feature Analysis

Chroma Feature analysis will be the second component of our song recognition pipeline. For this, we will convert both songs and sung samples into a popular audio processing data format known as a Chroma Feature (or chromagram). This is more or less a variant of a normal spectrogram: data is converted from the time domain into the frequency domain, and then is bucketed amongst the 12 notes of the Western chromatic scale. Since note is usually more informative than octave, different octaves are combined together so that the output is a single, 12-element vector, where each element corresponds to the total intensity of a note, independent of octave.

There are several interesting post-processing techniques that can be applied to a chromagram [2], such as normalization along a single 12-dimensional chroma vector, normalization across different chroma vectors, changing note intensity to be represented with a logarithmic versus a linear scale (closer to how sound intensity is perceived by the human ear), and performing windowing functions across neighboring chroma samples to smooth notes out (this has the effect of removing much of the timbre/instrument information). These, plus the problem of how many samples to create in the first place, leave a lot of tuning even in the chroma feature step of this matching algorithm.

These Chromagrams will then be subjected to cross-similarity analysis. There are many potential ways to measure which song's chromagram corresponds to a user-created sung chromagram--we could attempt to compute a distance function, or use some pattern matching techniques. Several researchers have had success creating cross-similarity matrices and training a Convolutional Neural Network to identify patterns within these matrices [6]. The cross-similarity matrix will essentially show the distance between EVERY point on the sung sample and EVERY point on the reference song. If the song is a match, it should be more likely to produce interesting patterns, such as diagonal lines of similarity over time, and blocks where many adjacent samples are all similar to one another. Training a CNN is the current state-of-the-art for analysis on problems similar to these, but since we are relatively inexperienced with machine learning and since, to our knowledge, nobody has ever attempted to use a CNN for this problem before, we are not guaranteed success. While a CNN has the highest potential to produce the best possible results, there may exist other chromagram matching strategies that we could use, such as those in [3]. We may also encounter problems training a neural net since we are not actually planning on working with a huge data set for this project--our end goal is to have a merely demo-size library of songs that can be matched against, and training a CNN might require much more than this.

C. Post-processing: Data Visualization

There will be a data visualization portion to see what the matching algorithm did regardless of whether or not it is able to generate a match. This will also serve as a form of visual debugging for the system. An example of what might be displayed can be seen in Fig. 2 in Section III. This example plots the determined melody of the input on top of the melody of the songs in the library, which shows the user how their input compared to existing potential matched. To visualize the other branch of matching, the app will display the cross-similarity matrix generated between the closest match(es) to the input song. Then, we would like to display the work of the CNN or another pattern-matching scheme we develop to match songs from the chroma feature analysis.

VI. Project Management

A. Schedule

The schedule can be viewed in Fig. 5 at the end of the document. This project began with researching similar existing projects and contacting professors for their advice and suggestions for our process. Using this information and guidance, we decided on the approach to take and began implementation.

The dynamic time warping and melodic contour analysis portion will begin with the pre-processing of MIDI files and filtering background noise from human input, and then proceed to matching the input against the existing songs. The chroma feature analysis will begin with converting songs into the chroma format and examining the similarity matrices, and then training convolutional neural nets to recognize the similarities. Testing for both branches of the matching process will be done separately for each portion to see their individual performance and then integrated together for the complete system.

The data visualization and application portion will begin with research and testing of existing data visualization methods to see what can be used and what concepts can be borrowed to create our own visualization technique. The design and creation of the app will take place later in the process.

B. Team Member Responsibilities

Anja is responsible for the majority of the dynamic time warping and melodic contour analysis portion of the system. Additionally, she will be working on creating the database of songs that the system will match to.

Nolan will be working on the chroma feature analysis and training the CNNs or other form of pattern-matching. This will involve generating the cross-similarity matrices for chroma features.

Wenting will be working on the data visualization aspect of the project and the design and creation of the application. She will also be working on filtering the human input to minimize background noise.

C. Budget

QCII One set of Bose headphones for audio verification testing - $350

D. Risk Management

The major risk of this project is its performance. Since there are no hardware dependencies, the points of failure are all in the algorithm not working or being very slow. We were aware of the risks from the beginning after looking at the performance of existing projects. For example, the original plan was to do polyphonic pitch tracking and break down sound files by ourselves, but that was found to be too difficult, so we switched to the back-up plan of just using existing MIDI files for analysis.

The existing projects that we looked at, which are detailed in Section VII, had varying levels of performance. The query by humming project only achieved around 30% accuracy, while the chroma feature analysis with CNNs performed well but in cover song identification, correctly identifying 8.04 out of 10 songs on average [6]. We do not know yet how it will perform for our task, though we will bias as necessary using our test findings. Doing the matching twice through different methods requires more computation power but will hopefully result in more accurate results. It may also take longer than our target of one minute, but we will try to reduce the time by refining results and biasing the model.

VII. Related Work

Since the problem we are setting out to solve has no well-accepted solution, we are drawing most of our inspiration from related work in the computer music community. Our query by humming path featuring dynamic time warping and melodic contour analysis draws from a query by humming paper by Roger Dannenberg, a CMU professor [1], [4]. The area of extracting melody from music is a topic of much current research. Most melody recognition on waveforms is not advanced enough to handle a popular song, but there are several methods for extracting a melody from a MIDI file. One of these was used by Prof. Dannenberg in his query by humming paper [5]. While this program is no longer under development, we are hoping to find a program with similar features that is currently available. There are many such programs being developed by computer music research groups around the world. Most extract melodies by identifying repeated sections of music to return a list of potential important themes.

Chroma feature analysis is a thoroughly-researched topic, and many variations of it and post-processing techniques have been proposed for different applications by different research groups [2]. Also, chromagrams have been used to monster mash songs together in many different contexts: work has been done on matching a chromagram of a song fragment to the corresponding location in the song [3]. This team also considered matching songs to their covers, but they worked within a pretty specific library of different recordings of classical pieces--each "cover song" had the same orchestration and music, but different performers and conductors. Because of this, we should expect a match on our system (which will be guaranteed to have different music and performers, and will most likely also have some slight tempo variations) to look very different than a match on their system. Work has also been done using cross-similarity matrices and neural networks to identify cover songs [6]. This work also differs from ours because the matching is on two complete songs, with (presumably) complete instrumentation and also with all melodies and sections of the song intact, while our plan is to match a full studio-recorded song with a solo vocal artist who will only sing one important melody, or perhaps only a fragment of one

VIII. Summary

We hope that our system will be able to meet the design specifications and will make modifications along the way as necessary. While we have solidified the approach we will be taking, we acknowledge that it will require some tweaking to figure out the fastest and most effective way to achieve our goals for matching. Our performance could be improved with more computing power and more detailed refinement of the neural net, which will be trained on a limited amount of data.

We anticipate that there will be many lessons learned along the way as we experiment with different methods of matching and get varying test results for the things that we try. So far we have found that polyphonic pitch tracking is a very difficult problem that has not yet been solved in a robust manner. We encourage people to make further attempts in the future, though for the scope of our project it turned out to be too difficult.

References

[1] R. Dannenberg et. al, "A Comparative Evaluation of Search Techniques for Query-by-Humming Using the MUSART Testbed" http://www.cs.cmu.edu/~rbd/papers/musart-testbed-JASIST-2007.pdf

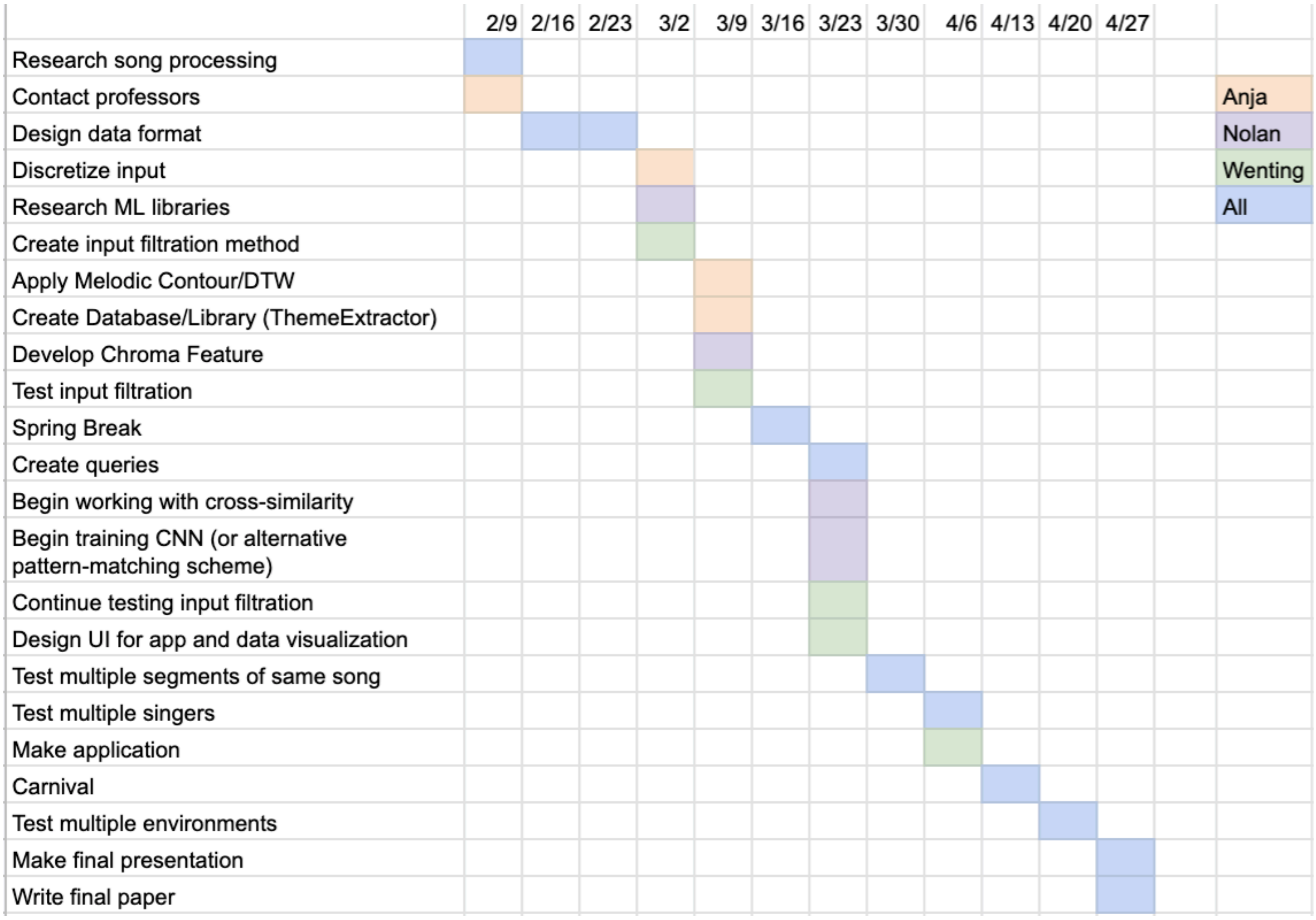
[2] M. Müller and S. Ewert, "Chroma Toolbox: MATLAB Implementations for Extracting Variants of Chroma-Based Audio Features" https://www.audiolabs-erlangen.de/content/05-fau/professor/00-mueller/03-publications/2011_MuellerEwert_ChromaToolbox_ISMIR.pdf

[3] M. Müller, F. Kurth, and M. Clausen, "Audio Matching Via Chroma-Based Statistical Features" http://resources.mpi-inf.mpg.de/MIR/chromatoolbox/2005_MuellerKurthClausen_AudioMatching_ISMIR.pdf

[4] D. Mazzoni and R. Dannberg, "Melody Matching Directly From Audio" https://www.cs.cmu.edu/~rbd/papers/melodymatching-ismir01.pdf

[5] C. Meek and W. Birmingham, "Thematic Extractor" http://ismir2001.ismir.net/pdf/meek.pdf

[6] S. Chang, J. Lee, S. Choe, and K. Lee, "Audio Cover Song Identification using Convolutional Neural Network" https://arxiv.org/pdf/1712.00166.pdf



	2/9	2/16	2/23	3/2	3/9	3/16	3/23	3/30	4/6	4/13	4/20	4/27		
Research song processing	🟦													
Contact professors	🟧													🟧 Anja
Design data format		🟦	🟦											🟪 Nolan
Discretize input				🟧										🟩 Wenting
Research ML libraries				🟪										🟦 All
Create input filtration method				🟩										
Apply Melodic Contour/DTW					🟧									
Create Database/Library (ThemeExtractor)					🟧									
Develop Chroma Feature					🟪									
Test input filtration					🟩									
Spring Break						🟦								
Create queries							🟦							
Begin working with cross-similarity							🟪							
Begin training CNN (or alternative pattern-matching scheme)							🟪							
Continue testing input filtration							🟩							
Design UI for app and data visualization							🟩							
Test multiple segments of same song								🟦						
Test multiple singers									🟦					
Make application									🟩					
Carnival										🟦				
Test multiple environments											🟦			
Make final presentation												🟦		
Write final paper												🟦		

Fig. 5. Gantt chart of the schedule and division of tasks