Ragam Finder

A Prototype to Identify South Indian Classical Music Scales

By: Arun Shriram
Advisor: John Clements

June 11, 2020
Computer Science Department
College of Engineering
California Polytechnic State University
San Luis Obispo, CA

Table of Contents

Abstract

There are millions of listeners and students of South Indian classical music, and many are fascinated with, and sometimes struggle with identifying scales, called ragams. The Ragam Finder is a prototype program that strives to identify these ragams directly from audio input. It does so by using Fast Fourier Transforms to convert audio samples to pitches, identifying fundamental frequencies, despecklling and analyzing audio data, and determining the notes that were played. However, due to using methods that are useful but sometimes prone to error, the eventual output is not perfect, but is still promising for future research and implementation.

Introduction

Music has been in my family for generations. As professional performers of South Indian classical music, or Carnatic music, my family has been known to push the boundaries of the craft in different and creative directions. Over the last six months, I have set out to do the same, for my Senior Project. My initial goal for this project was to create an app that will listen to ragams, or South Indian classical music scales, in real time, and identify the name of the ragam. There are hundreds and thousands of people across the U.S. and millions in India who regularly listen to Carnatic music. As a performer and listener of Carnatic music, I would like to provide a convenient tool for listeners to identify any song's ragam and to hopefully have them improve this skill. There are thousands of possible ragams, which could make this app a unique and useful solution for listeners and musicians alike to ascertain the ragam of music being played.

However, there were several complications that made progression of the project difficult. In fact, this project was unable to reach the app development stage, as the primary algorithm has been under continuous development in Python. Given that this is a topic that is not well-researched, to my knowledge, there have been no attempts to create a program like this before. As a result, even with the help of my advisor John Clements, it has been difficult implementing a consistently successful ragam identification program that does not rely on clear and pure data, which is something that is typically not present in Carnatic music concert. This means that most of the test data I have used were of people singing Carnatic scales slowly and carefully, without excessive musical ornamentation, which is characteristic of Carnatic music and ragams. Although the project is still in a prototype phase, I hope to continue working on this after I graduate, performing research on the most optimal methods to identify ragams, and eventually releasing a highly functional version to the public.

Background

Carnatic Music

Carnatic music has been an important part of South Indian culture for centuries. This style of music is performed through songs, most of which were composed by great sages and musicians centuries ago. Concerts are typically performed centered around one musical pitch, called the sruthi, and songs are set to certain ragams. Ragams are sets of notes with certain rules, and have been compared to modes in Western music. However, there is no true equivalent of ragams in Western music, as ragams tend to have many more nuances. Most Carnatic songs are composed for one ragam, but some can span multiple ragams within the song.

The most common way to express Carnatic music is vocally. Typically, a vocalist is accompanied by a stringed instrument, such as a violin, and a percussion instrument, usually a mridangam. In a concert, the vocalist will choose a sruthi (a base pitch) that is most suitable for their voice (males usually choose C3 or D3, while females choose a range from F3 to A3). In the olden days, and sometimes in modern concerts, a tambura was plucked to continuously play the sruthi notes, which are the tonic, the dominant, (sometimes the subdominant, depending on the song), and an octave below the tonic. This allows performers to stay on pitch throughout the concert. Nowadays, performers use electronic tambura sounds. The tambura is played as a continuous drone that does not end until the concert finishes, and is a prominent sound in Carnatic performances.

The most important elements of Carnatic music are Sruthi, Swara, Ragam, and Taalam. As previously described, sruthi commonly refers to musical pitch. It is the approximate equivalent of a tonic, or less precisely, a musical key; it is the note from which all others are derived. A swara is a single note that defines a relative position of a note, rather than a defined frequency. Swaras also refer to the solfège of Carnatic music, which are: “sa ri ga ma pa da ni sa.” Every member of the solfège (called a swara) has two variants. The exceptions are the drone notes, which are sa and pa (the tonic and the dominant), which only have one variant. Lastly, a ragam in Carnatic music prescribes a set of rules for building a melody. It specifies separate rules for note movement up (called the aarohanam) and down (called the avarohanam), the scale to determine which notes should be used frequently, which notes should be used sparingly, which notes may be sung with ornamentation, which phrases should be used or avoided, and so on. If any of these rules are betrayed, the ragam may immediately sound like a different one. Ragams may have all seven notes, may have a small pattern in the ascending/descending scales, or may not have certain notes altogether. Every swara in the ragam may be one of its varieties. Taalam refers to a fixed time cycle or meter. Although taalam is itself a fascinating topic, it is not pertinent to this project.

Pitch Detection

Pitch detection begins with understanding the sample rate of audio. The sample rate is the number of audio samples taken from an audio file per second. The sample rate I used is the standard one: 44,100 samples per second, or 44.1 kHz. Samples can be grouped into windows containing a certain number of samples per window. This lets us see audio trends and patterns within each window and between windows. Figure 1 depicts a certain window of 4096 samples.

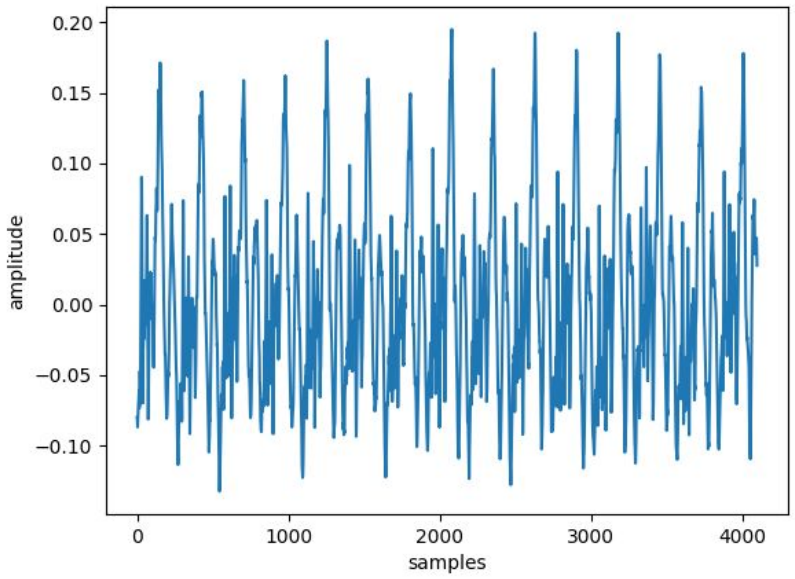

Figure 1. A graph that represents a window containing 4096 samples.

Using Fast Fourier Transforms, we can take the data from each window and convert it into a graph showing the intensities (amplitudes) of each frequency. If a frequency is found, it is usually denoted by a spike at a certain frequency, as shown in Figure 2.

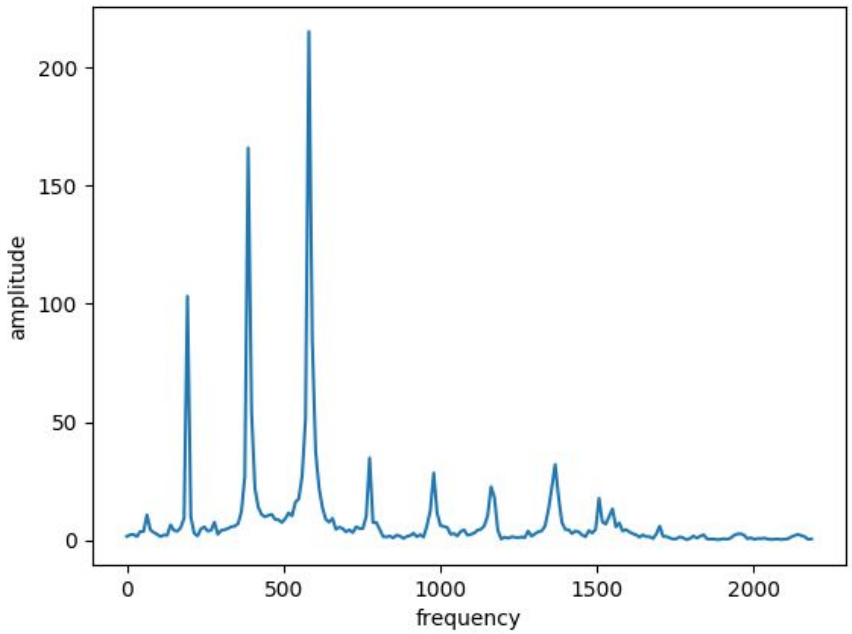

Figure 2. A graph that represents the frequencies given by FFT analysis from a window. Here, the x-axis represents the frequency of sound, and the y-axis represents amplitude.

A musical note is determined using the frequency at which a spike occurs. In Figure 2, there are multiple frequencies detected, at 196 Hz, 392 Hz, and at 587 Hz. Each frequency corresponds to a note; 196 Hz is the frequency for G3, 392 Hz represents G4, and 587 Hz represents D5. At this point, we can begin identifying fundamental frequencies. What is a fundamental frequency?

Sound, and music in particular, is usually carried through multiple frequencies at once, not just one at a time. When the pitch C3 is played on a violin, that is not the only pitch that is being physically carried through the air; acoustically, the violin strings are also producing the notes C4, G4, C5, E5, G5, and more. The pattern between these frequencies is that these pitches are all products of the base frequency (C3) and natural numbers: C3 is 1* the fundamental frequency, C4 is 2* the frequency, G4 is 3*, C5 is 4*, and so on. Thus, we can use these additional frequencies, called overtones, to identify the fundamental frequency, or multiple fundamental frequencies, for each window. After all the fundamental frequencies from the audio file are identified, we can convert each frequency to its corresponding pitch.

Methods

At first, my preliminary goal was to identify the sruthi of a given recording. However, I soon recognized that there was no accurate and consistent method of

identifying the sruthi. Given pitches alone, and without a tambura playing the sruthi notes, even a seasoned Carnatic musician may sometimes not be able to correctly identify which note is the tonic. Given that the sruthi could not be determined prior to identifying the ragam, I focused on correctly identifying notes instead, assuming that the sruthi is given as a manual input. One eventual goal of the project is to run the ragam identification algorithm using each sruthi (from C to B) and output the list of all the possible ragams for each sruthi. Although this is not the result I had originally envisioned, moving in this direction was necessary to create the core functionality of the program, which is identifying ragams, not sruthis.

The ragam identifying algorithm can be split into five parts: FFT analysis, fundamental frequency calculation, despeckling the acquired fundamental frequencies, forming transitions from source notes, and then comparing these transitions to those from a database of ragams.

FFT Analysis

The Python library "aubio" was used to obtain the samples from the given source file. As previously described, Fast Fourier Transform (FFT) analysis transforms raw sample data into frequencies divided into windows. In a file of 10 seconds, there should be a total of about 107 windows to analyze, given a sample rate of 44100 and a window size of 4096 samples per second. These values were chosen because 44.1 kHz is the standard sample rate for audio files, and using a window size of 4096 maintains a resolution of 10.8 Hz, which was the average distance between the frequencies that are typically played by Carnatic musicians. The mathematic Python libraries NumPy and PeakUtils were used to perform the FFTs on the samples provided by aubio. The default peak threshold of 0.3 provided by PeakUtils produced satisfactory peak detection, although there was no statistical analysis performed to ensure that this threshold is optimal. This threshold determined what percentage of the maximum amplitude value should be used to identify the other pitches for a certain subset of data; for example, if the highest peak had a y-value of 100, then all peaks identified would have y-values of 30 and above.

Fundamental Frequency Calculation

Most of the frequencies detected by aubio ranged from 32 Hz (C1) to 1570 Hz (G6). However, there were occasional outliers at double this frequency, in the 4000 Hz range. These outliers were processed for fundamental frequency calculation, along with all the other frequencies detected for each window. The algorithm that detects the fundamental frequencies for each window is given the list of frequencies that were detected in a particular window. The algorithm then attempts to determine an overtone relationship (where a higher frequency is a product of a natural number and the lower frequency) between every frequency in the list, checking in descending order. Since the maximum

sample resolution is about 10 Hz, the fundamental frequency calculation is most accurate when handling higher frequencies first. If there is only one music source in the given recording, there should be only one fundamental frequency for each window. However, if the singer sings with a tambura, or another instrument, there may be multiple fundamental frequencies detected for each window. This inconsistency makes it difficult to determine which fundamental frequency refers to a singer and which refers to another instrument or singer.

Despeckling

"Despeckling" refers to removing extraneous fundamental frequencies that may be incorrect or misplaced; the raw list of fundamental frequencies cannot be used to extract the notes. For example, a section of a given recording may contain frequencies that correspond to the pitches "G, G, G, F#, G, G, G#…" and so on. Given that these frequencies refer to just a few windows, less than a second of real time, it is immediately apparent to humans that the pitch being played is G, but the outliers F# and G# may be translated to notes that are not really being played. An example of the notes detected from raw fundamental frequencies is shown in Figure 3.

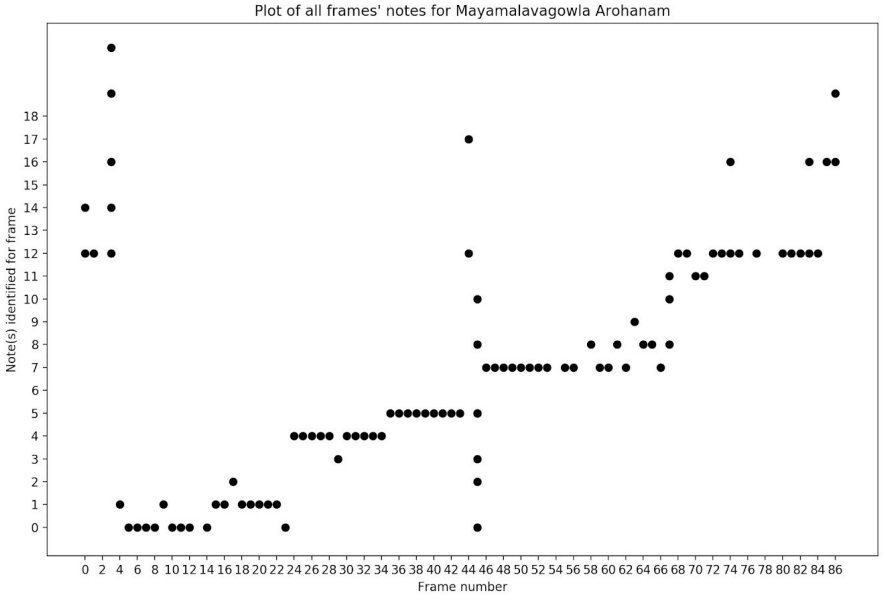

Figure 3. This graph shows the raw fundamental frequencies found for an ascending scale of ragam *Mayamalava gowla*. The y-axis represents notes, indexed 0-11, and the x-axis shows the frames (windows) from beginning to end.

Despeckling solves this issue by grouping these windows into so-called "superwindows," which contain an arbitrary number of windows. For the purposes of testing with simple files that contained the ascending and descending notes of ragams, with

singers stopping on each note for about a second, a superwindow size of five windows proved to be the most accurate size. The despeckling algorithm works by determining the most common frequency within the superwindow, and replacing the center note of the superwindow.

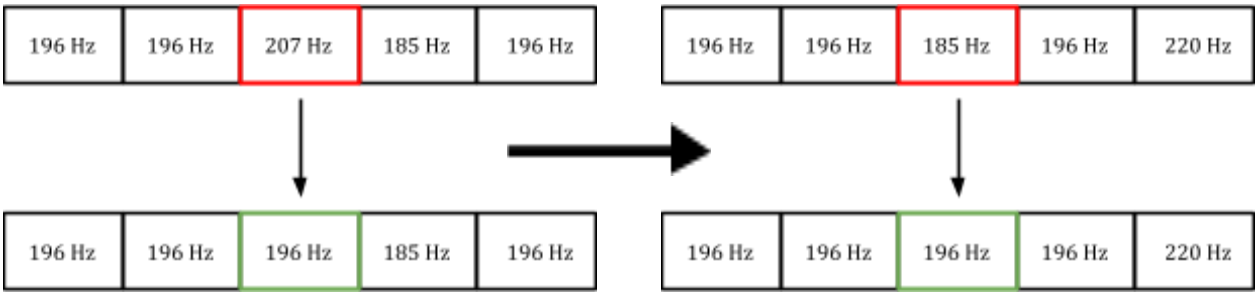

Figure 4. Demonstrates the despeckle replacement algorithm. The "207 Hz" is replaced with the most common frequency in the superwindow, "196 Hz". Then, the superwindow is shifted over by one window, and replaces the "185 Hz" with "196 Hz" in the same fashion.

The superwindow then shifts over in the list of fundamental frequencies by one, and reevaluates the most common note from the superwindow. This process repeats for all the fundamental frequencies from a sound file. This replacement action is shown in Figure 4. The results of despeckling the file from Figure 3 is shown in Figure 5. The despeckled file is much "cleaner" as a result, removing extraneous fundamental frequencies.

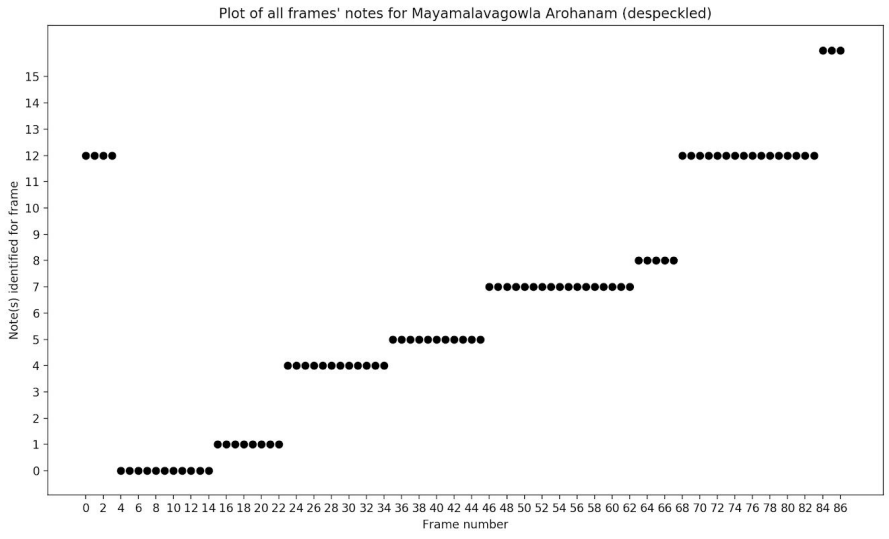

Figure 5. Resulting graph after despeckling ascending scale from the *Mayamalava gowla* ragam.

Note Transitions

In the Carnatic system of music, there are actually 16 notes mapped onto the 12 pitches; though the solfège of Carnatic music consists of "sa re ga ma pa da ni sa," (denoted as S R G M P D N S), there are multiple positions for most of the notes. These positions are S, R1, R2/G1, R3/G2, G3, M1, M2, P, D1, D2/N1, D3/N2, and N3. There are four pitches that have enharmonically equivalent notes, meaning that certain pairs share the same pitch, although the specific notes still matter when determining ragams. To avoid future complications, the 16 notes were simplified to the following 12 notes: S, R1, RG1, RG2, G3, M1, M2, P, D1, DN1, DN2, and N3. Using the despeckled fundamental frequencies, frequencies that were present at least twice in a row were counted as a "note," and the total list of frequencies was simplified to a list of notes. The transitions between notes were then identified and eventually compared with all the possible transitions for each ragam.

Transition Comparison

Although there are thousands of possible ragams, there is no public, centralized, or official documentation of each and every ragam. However, there are a few websites that list a large number of ragams. I picked the smallest list of ragams from these websites to use for ragam identification. I then created a parser that decodes this list of ragams and saves the ascending and descending scales of each ragam. While the parser decodes ragams, it maps the 16 notes to the 12 generalized notes mentioned above and creates a transition matrix that stores all the possible ascending and descending transitions possible for that ragam. After the ragam identifying algorithm discovers all the transitions present in the audio file, these transitions are used to narrow down the list of all ragams, comparing each transition with those present within each ragam, until a small subset of ragams is remaining.

Testing Files

Choosing the files to use for testing the ragam identifying algorithm was tricky. At first, I used audio snippets of real concerts, with a vocalist and other instrumentalists. However, I quickly discovered that isolating fundamental frequencies amidst multiple frequencies for each window was particularly difficult. I then began to test the algorithm on files that had only a singer and a tambura. However, the natural but observable vibrato and ornamentation of the singer made some notes indiscernable. This is particularly visible in Figure 3, where the note N3 (note index 11) is wavering between D1 (index 8) and N3. After despeckling (as seen in Figure 5), this note is erased entirely. Thus, to produce the most clean input to test the algorithm, I recorded myself singing the ascending and descending scales of ragams in a plain and flat voice, with a tambura playing in the

background so that I can be sure that despeckling accomplishes its purpose of removing occasional extraneous background noises.

Results

Most of my testing revolved around the ragam *Mayamalava gowla*, since this is usually the first ragam taught to novice singers. The ascending notes are as follows: S, R1, G3, M1, P, D1, N3, S. The descending notes are the same, in reverse order: S, N3, D1, P, M1, G3, R1, S. Using a recording of myself singing the ascending and descending scale of the ragam, the ragam identifying algorithm discovered the transitions listed in Figure 6.

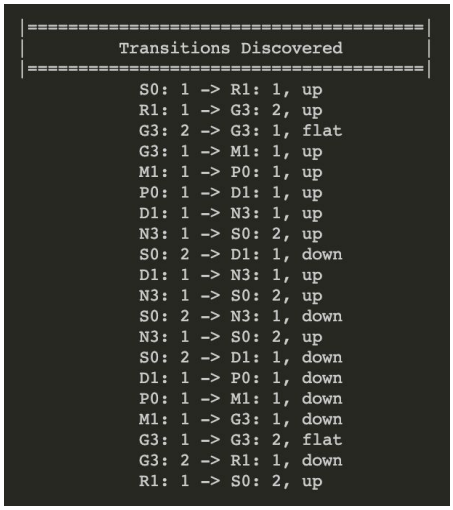

Figure 6. A list of transitions discovered for the ragam *Mayamalava gowla*.

The final result, produced after cross-analyzing the discovered transitions with those determined from the list of ragams, is shown in Figure 7.

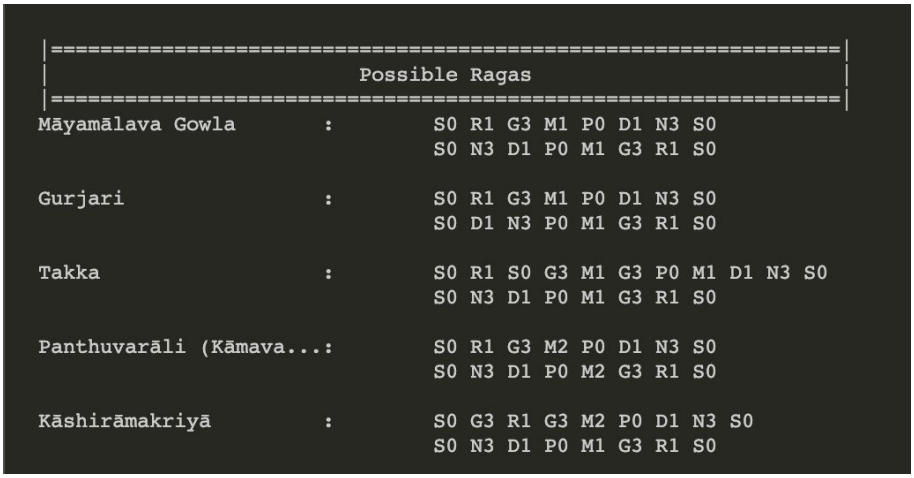

Figure 7. Final output of the program, attempting to identify the ragam *Mayamalava gowla*.

Assuming the sruthi is C3 (as a given input), and out of a total of 942 ragams, the ragam identifier narrowed the list down to five ragams: *Mayamalava gowla*, *Gurjari*, *Takka*, *Panthuvarali*, and *Kashiramakriya*. Clearly, each of the transitions shown in Figure 6 are present in each of the ragams shown in Figure 7. Although the final product of the ragam identifying algorithm produced five results instead of just one, it is important to note that there were several circumstances along the process of identifying a ragam where it was necessary to allow for error to maintain functionality. In addition, these results were determined based on the assumption that the sruthi is already known. If the sruthi were not known, the transition-identifying process would have to be repeated for every pitch from C to B, thus increasing the number of possible results that the program would output.

Discussion

There were several difficult decisions that had to be made along the process of designing this program that gradually increased the amount of error. These decisions included identifying the threshold to measure frequency peaks, deciding how many overtones should be present to identify a frequency as fundamental (which would affect the necessary threshold for peak detection), handling wavering notes with vibrato, isolating the right fundamental frequency from a window where more than one was discovered, determining whether a note is noise or a note actually played, and many more. Given that this process of audio analysis is delicate and relies deeply on the accuracy and purity of the input audio quality, working with 7-10 second audio clips proved to be tedious, but necessary to test the finer aspects of my pitch detection algorithm.

Future work might include tweaking the algorithm to work for specifically longer audio files. Longer audio files might help emphasize which pitches are notes, since the data set would be larger, and would make it more apparent which pitches are background noise, or part of the tambura. In addition, changing the superwindow size for despeckling depends on the speed at which notes are played. The reason a superwindow size of 5 was chosen was because the test audio files had notes sung at the rate of 1-1.5 seconds per note, meaning that a given note would last about five windows per note, allowing superwindows to smooth out the unavoidable pitch outliers between notes. Future work would include a more efficient and specific way to manage the superwindow size to fit notes that are sung faster. In addition, notes sung at different speeds may require appropriate window sizes, as a constant size of 4096 samples may not apply to every instance of Carnatic music performance.

Despite the nature of the various errors resulting from the decision-making process, *Mayamalava gowla* seems to have, from the ragams I have tested, the most accurate and acceptable ragam identification results. Further testing needs to be done to verify how accurate the ragam identification really is. Further investigation also needs to occur regarding the various debatable decisions that had to be made, and how to improve on

those decisions given the experience I now have. This investigation may need to be formed on some kind of statistical basis, so that decisions can be made with high levels of confidence and on a case-by-case basis, rather than generalizing rules that apply to all recordings and to all ragams.

My code for this project can be found here. Please keep in mind that upon submitting this report, this project has been a prototype, with functions, classes, and other code constantly in flux. As a result, there is not much overarching structure over all the code. However, it should be fairly straightforward, as it is thoroughly commented.

Conclusion

Though I began this project with high hopes and aspirations, there was much to learn and much to implement. The further I progressed with the program, the more challenges arose, albeit amidst smaller successes. I believe that if I continue working on this program in the future, beyond Cal Poly, it could potentially become a revolutionary success among the Carnatic population, since a program to identify ragams has never been attempted before and represents a strong need in the community.

In the distant future, I hope that the program can be expanded in the ways I had initially envisioned. Someday, the program should be able to distinguish voices from other instruments, identifying audio sources by timbre (denoted by distinct patterns of overtones), so that the notes that multiple singers and instruments perform together can still be analyzed in parallel, without notes getting mixed up with each other. In addition, I think that some level of machine learning will be useful in determining the sruthi from the recording itself, reducing the amount of error produced by ragam identification. Machine learning may also prove helpful in identifying ragams as well, as many ragams tend to have characteristic phrases that are nearly always used to express and discern them.

In order for machine learning to be useful, however, a vast quantity of training data should be acquired for each ragam. The machine learning model should be trained appropriately (i.e. categorize notes being sung rather than instruments being played), and the test data should be pure, without extraneous noise or other factors that may influence the training of the model.

Although there is quite a large amount of complexity with regards to the way humans identify ragams, experienced musicians can do so almost immediately. I firmly believe that it is possible for machines to do so as well, and in a way that Carnatic music students and lovers alike can use to enjoy and learn more about this beautiful art. I would like to thank John Clements and my family for providing me much guidance and assistance with this project. Working on this project was an incredible yet formidable experience, and a fantastic opportunity to learn how to analyze sound and music at a detailed level.