2015

Exploring Algorithmic Musical Key Recognition

Nathan J. Levine
Claremont McKenna College

CLAREMONT MCKENNA COLLEGE

EXPLORING ALGORITHMIC MUSICAL KEY RECOGNITION

SUBMITTED TO

PROFESSOR EVERETT BULL

BY

NATHAN LEVINE

FOR SENIOR THESIS

SPRING 2015

APRIL 27, 2015

Table of Contents

Acknowledgments

I would like to acknowledge my thesis reader Professor Bull, as his guidance and unwavering patience are what ultimately made this possible. My family and my loving friends deserve acknowledgement for helping me get to this point, and for their much-needed support through the thesis writing process.

Abstract

The following thesis outlines the goal and process of algorithmic musical key detection as well as the underlying music theory. This includes a discussion of signal-processing techniques intended to most accurately detect musical pitch, as well as a detailed description of the Krumhansl-Shmuckler (KS) key-finding algorithm. It also describes the Java based implementation and testing process of a musical key-finding program based on the KS algorithm. This thesis provides an analysis of the results and a comparison with the original algorithm, ending with a discussion of the recommended direction of further development.

I. Introduction

Music that we hear, composed fundamentally of physical vibrations, or sound waves, has many underlying mathematical concepts. For instance, any note coming from a musical instrument is a wave composed of many different frequencies that are often multiples of each other (Müller et al. 2011). Furthermore, sound waves stacked in this way that are exact integer multiples of each other, or harmonic frequencies, tend to sound most pleasing to the human ear (Parker, 2009). Because it is possible to mathematically characterize sound waves and derivatively musical notes, it is possible to computationally analyze music in terms of concepts in music theory. This can take many forms, but the particular concepts in Western music theory that we are interested in for the scope of this thesis are those concerning musical key. Since music theory already relates keys to each other in logical and algorithmic ways, the prospect of using computers to analyze music for its key is appealing. To this end, much investigation has been made into methods of pitch detection and various bases for key detection (Krumhansl 1990, Temperley 1999, Shmulevich et. al 2000, Gerhard 2003, Sapp 2011).

The theoretical concepts underneath music's tonal surface inform the mathematical formulas that allow for pitch recognition, which in turn enables the key selection process. The rest of this thesis is laid out following this process, starting with the relevant music theory, moving to a discussion of pitch detection methods, followed by a detailed account of the Krumhansl-Shmuckler key-finding algorithm, and finishing with the description and analysis of my own creation of a key-finding program.

II. Understanding Musical Key

Most people, if not musically trained, would have a difficult time identifying the key of a given piece of music upon listening to it (Pauws 2004). Despite this, musical key is a fundamental concept in Western music theory that applies to most, if not all pieces of music. It is such a core feature of a piece of music that it is written on the first line of the musical staff, though the same key may not hold for the entirety of the piece (Sapp 2011).

Represented by a base musical note called the tonic, a key provides a reference by which different notes are perceived to relate throughout a piece of music. Each key is associated with a scale starting with the tonic, or note for which that key is named (Harrison 2009). Each position in the scale for a given key is perceived to have a different effect in terms of harmony or melodic motion. For instance, the fifth note in a given scale is called the dominant note, which when played, gives a sense of harmony with the tonic note. However, if a chord progression ends on the dominant note, it sounds incomplete and leads the ear to expect a continuation of the melody (Temperley, 1999). Each such scale position has its own relationship and thus tonal effects on the other notes in the key. The note that sits in each place in the given scale for a key is determined by that key's tonic note. Thus, the dominant (or any other position) is a different note in each key. Certain keys have more sharps or more flats, which gives a certain tonal quality to a piece of music as well. It is the key signature that denotes which notes are played as sharp (a half step higher) or flat (a half step lower) in that key (Harrison 2009).

In Western music theory, there are twenty-four keys in total- twelve major and twelve minor. Since there are only twelve distinct pitches, each pitch has one associated

major and one associated minor key. Two major and minor keys sharing the same tonic, or first scale note, are differentiated by the intervals between each following note in the scale. These intervals simply represent how many distinct pitches, if any, are skipped to find the next note in the scale. The scale intervals for all major keys (though not the specific notes themselves) are the same. Additionally, the same relationship holds for all minor keys as well, but with a different set of intervals than the majors. This is what gives major and minor keys their distinct tonal qualities (Krumhansl 1990). These underlying conceptual relationships- both among the scale positions of a key and the keys themselves - are what many key-finding algorithms use to accurately compute key correlation (Krumhansl 1990, Rowe 2001, Zhu et al. 2006).

III. Applications of Musical Key Analysis

Though a musical key just denotes a certain set progression of notes, including sharps and flats, musical key analysis is the basis for many other concepts in music theory by which music can be understood. As briefly mentioned earlier, the key of a piece of music is often associated with its tone, or the mood the piece evokes. A piece of music, especially in the classical genre, may change keys many times throughout its duration, each time evoking a new atmosphere and possibly tension, or bringing back an old context for completeness. Furthermore, the context set by a certain key can be intentionally broken by notes outside of the key, for the purpose of creating tension, providing novelty, or hinting at an upcoming key change (Rowe 2001). On a macro scale, analyzing these key changes provides essential insight into the flow of a piece of music

and helps the categorization of certain types of music into different styles (Krumhansl, 1999). This doesn't only apply to classical music. A DJ (disc-jockey) who mixes different songs together for performance purposes pays close attention to the key of each track in order to smoothly transition from song to song- since certain keys are highly incompatible and sound jarring if played in unison. Since almost all modern DJs use digital music files and software packages to accomplish this mixing, the problem of key determination has become a computational problem. Multiple programs exist with the sole purpose of musical key determination, and they perform this task with varying degrees of accuracy (White, 2014). In addition to DJ specific software, I believe there is potential application of computational key determination in music recommendation software such as Spotify or Pandora. Since key is so fundamental to the overall feel of a musical track, when considering the next song to recommend, the same or a compatible key should be considered as a major factor.

IV. Overview of Algorithmic Key Detection

To computationally determine musical key from an audio sample, there are two general problems to be solved. First is the process of pitch detection. Since the type of audio input we are concerned with is in some encoded form composed of bits digitally representing a sound wave, there are no implicit pitch characteristics easily recognizable by a computer, only data points (Zhu et al. 2006). This is in contrast with an input format such as MIDI, which transmits discrete pitch data as part of its encoding (Rowe 2001). The problem of pitch detection, then, is to somehow analyze this digital representation of

a sound wave to determine as accurately as possible which individual notes or chords it represents.

Assuming this is satisfactorily accomplished, the second part of the key detection process is to formulate a method by which a key is chosen given the individual tones or chords identified in the pitch detection step (Zhu et al. 2006). This is also a complex endeavor since, as mentioned earlier, the key of a piece of music can shift rapidly, or be contradicted for dramatic effect. Furthermore, certain keys share a similar tone and consequently contain many of the same notes and chords in sequence. This is most likely to happen with relative keys, i.e. a major and a minor key pair that share the same key signature but each has a different tonic note. These keys are most prone to be transitioned between in compositions, and therefore also likely to be conflated by key-finding algorithms (Krumhansl, 1990). Because of this, many algorithms provide a ranking of the likely keys by how well correlated they are to the input (Temperley 1999, Sapp 2011). The relative success of this key determination step is also highly dependent on the accuracy of the pitch detection method used.

V. Pitch Detection

Because pitch is a purely a human perceptual feature of music, i.e. it represents the way that music sounds once it has been converted by our sensory organs into neuro-electrical signals and processed by a section of our brain, it is a very difficult problem to try and pinpoint pitch using a computer (Müller et al. 2011). Putting it another way, the goal of pitch detection is to process a signal in whatever way most accurately determines

what it would sound like if that signal were to be physically produced and then experienced by a human.

Though each note is composed of many different frequencies, the perceived pitch of the note is commonly equated with the fundamental frequency of the note, since this is usually the basis from which the other harmonic frequencies are derived. Given this, many pitch detection methods focus on identifying the fundamental frequency or frequencies of a sample (Salamon 2014). This is not an entirely accurate model of pitch, since real instruments produce complex sound waves with many overlapping frequencies, especially in the upper ranges. However, in most cases this is a necessary simplification that makes the problem of pitch detection much more manageable. Given this, the goal of a large number of pitch detection algorithms is to make this simplification from a rich waveform with many harmonic components to one fundamental sinusoid with a set frequency as clean and accurate as possible (Müller et al. 2011). Obviously, the more harmonics contained in the waveform, the more difficult this becomes.

A common general approach to this problem of fundamental frequency detection is to simply use the sound wave in the time-domain as the input. In particular, looking for periodically repeating events, or shapes in the sound wave, which can then be related inversely to the frequency of the sample. One extremely basic method used for this purpose is to measure the zero-crossing rate, i.e. the number of times that the wave function crosses zero on the y-axis. If the sound wave is clean and very sinusoidal, this may give a useful correlation with frequency since a sine wave will cross the y-axis twice each period. However, this is quite simplistic since if the waveform contains many other partial components, it may vibrate around the y-axis and cross many times, yielding an

inaccurate measure. Another similar time-domain based approach for finding fundamental frequency is to measure the periodicity of either absolute maximum or absolute minimum values in the wave instead of zero-crossing points. Yet a third method uses the same reasoning, but looks at the function's slope instead, which should be periodic if the function itself is periodic. All three of these share the same weaknesses. They are prone to inaccuracy given a complex input, since there is no guarantee of a smooth, constant wave around the zero-point, the peaks, or with regard to slope. Despite this, they are useful methods due to their relative simplicity, which translates to ease of implementation and efficiency with respect to memory and processing power. Furthermore, they can be improved using simple filtering techniques aimed at cleaning up a waveform, such as filtering out the confounding higher frequencies (low-pass filtering) (Gerhard 2003). These time-domain based techniques are the most simplistic approaches to the problem of pitch detection and seem insufficient in any case with input samples of high musical complexity. They could potentially be used in conjunction with some of the other methods or as a preliminary test, but in my own application I chose not to implement any of them.

Another time-domain based method of detecting the fundamental frequency, dissimilar from the previous three, is that of auto-correlation. This is the mathematical process of computing the correlation of a waveform with itself, as a function of an increasing time-lag between the start of one copy of the function and the other. If a waveform is periodic, this auto-correlation function will also be periodic, since as the time-lag of the function copy approaches half the period of the original function, the two will be maximally out of sync, and therefore least correlated. Then as it approaches the

full period, the two will again be in sync and therefore most correlated- repeating so on and so forth. The peak in this auto-correlation function would therefore represent the period of the original function, from which the frequency could be derived (McLeod 2014). This seems to be useful in many cases, but may be inaccurate if the function has the appearance of periodicity early on due to its overlapping harmonics, but is not truly repeating until much later. Similar to the time-domain methods, the auto-correlation function appears to be usably accurate only in simple cases, and would need to be further refined or combined in some way with another technique.

Other methods of pitch detection take place in what's called phase space. Rather than the time-domain, this is the domain created by plotting the value of a waveform at a certain time with its slope at the same time. If the waveform is periodic, the phase space representation will have a periodic cycle as well. Using higher order derivatives of a function's value as the y-coordinate creates higher order phase space representations of the function. If a waveform plotted in phase space has a closed cycle, the problem is to find the point at which the cycle starts to retrace itself, which gives the period of the function. This is difficult since a phase space function may cross itself without completing a cycle, or follow itself inexactly (Gerhard 2003). These problems are similar to those faced by the zero-crossing rate method of determining period, and there is no one clear-cut solution.

The major category of pitch detection algorithms that have been the subject of the most recent development are those that seek to analyze waveforms in the frequency domain. These generally follow the structure of first filtering the input in some way, breaking it into many discrete parts, and then transforming it into frequency data through

some mathematical process (Salamon 2014). A common way of performing this transformation is a fast Fourier transform – a process using matrix multiplication that takes in an array of data representing the sound wave and returns an array in which each bin represents a certain frequency range, where the value contained in it represents the magnitude of those frequencies in the original sample (Roche, 2012). This encapsulates the basic idea, but there are many algorithms building off of this idea with refinements and tuning either aimed at cleaning the signal before processing it, or transforming it more accurately into frequency values (Pauws 2004, Zhu 2006, Salamon 2014). One specific such algorithm is called a Cepstrum analysis. Besides some preliminary filtering, the Cepstrum analysis also takes the log of the magnitudes before transforming it, so as to more accurately follow natural harmonics (Gerhard 2003). These frequency-based methods of pitch detection seem the most effective in general and the most rich for further development, which is the reason I chose one for my own program.

VI. The Krumhansl-Shmuckler Key-Finding Algorithm

This key-finding algorithm, developed by Carol Krumhansl and Mark Shmuckler is heavily referenced in its field (Temperley 1999, Shmulevich 2000, Zhu 2006, Sapp 2011). It is relatively simple to implement in terms of code, yet it takes into account many complex concepts of music theory. The algorithm takes its inspiration largely from ideas in cognitive psychological analyses of musical perception. It features twenty-four key-profiles of twelve values each, where each value represents the degree to which each pitch class in Western music theory is perceived to relate to that key. The basis of these

twelve values is a set of psychological studies conducted by Krumhansl along with several other researchers, to attempt to put a numerical value on this idea of how well a tone fits into a certain key. The methodology for these studies is based on the concepts outlined earlier regarding the way certain scale degrees within a key sound with respect to each other (Shmulevich et al. 2000). For instance, sounding seven tones of an A major scale, from A to G# leads the ear to expect the completion of the scale on the tonic note, A. This is the most pleasing progression to hear, even with an untrained ear. However, sounding the dominant note in A major, an E, would certainly sound somewhat more fitting than a Bb for instance. Thus, we see that there are certainly degrees to which different tones might be perceived to fit into a key just based on the perceptual qualities they evoke, even in those without formal musical training (Temperley 1999, Pauws 2004).

Using this exact principle, Krumhansl and her associates carried out their experiments on groups of volunteers from diverse musical backgrounds. They played an incomplete major scale as described above, and completed it with each of thirteen different “probe tones,” one for each pitch class, and one of the tonic but in a different octave. This was done with different instruments, some digital and some analog, to control for the instrument type. They then asked each participant to numerically rate each of these for how well they perceived them to complete the scale. They found that as an overall trend, the tonic note was most preferred, followed by the other tones that appear in the scale, followed by the tones that do not occur in the scale. This result was the most clear cut in the group they tested with the most musical experience, and started to break down more and more in the groups with less and less experience. However, overall, tones

closer in pitch to the musical context being presented were preferred to those further away (Krumhansl 1990, Shmulevich et al. 2000).

To extend and confirm these findings, as well as to finalize the key-profile values used in the final key-finding algorithm, Krumhansl carried out a second set of studies. In these trials, different contexts were used instead of simply an incomplete major key scale. They included longer scales and different types of chord progressions in both major and minor keys. Instead of asking participants how well they thought the final tone completed the scale, they asked how well they thought that it fit into the overall musical context (the key) that was established. In this second run of trials they also specifically selected participants with at least a moderate level of musical experience, so they could easily distinguish tonal differences, but not too advanced, so the results would be influenced most by perceptual qualities rather than higher theoretical concepts. The results of these newer studies served to confirm the tonal hierarchies established initially, and helped to remove any confounding factors. Ultimately, the final twelve key-profile values were taken as averages of the participants' numerical ratings for each pitch class's perceived fit (Krumhansl 1990). Since the scale degrees in a given major key and the following major key are simply transposed by one degree, with the second note becoming the first, and the first wrapping around to become the last, the same relationship can be followed with the twelve key-profile values. Therefore, with just one set of values for a key-profile, the rest of the profiles can easily be inferred. The same relationship holds for all twelve minor keys.

The idea behind the key correlation part of the algorithm was based on the notion that these key-profile values could give a sense of interkey distance. In other

words, that the values could numerically quantify how different keys relate to one another, how similar each key is to the others, and how each pitch fits into these interrelated hierarchies (Krumhansl 1990). Along the same lines, if two keys can be related in this way, a given key-profile could also be correlated with an input of similar form representing the tones in a musical sample. The resulting correlation value would then quantify the degree to which the musical input is similar to or fits into the context of the key being used. A high correlation value with the values from a certain key would imply high likelihood that the musical piece was written in that key. This correlation process can be done for the pairing of the input values to each of twenty-four key-profiles, resulting in a ranking of how well each key matches the sample. These twenty-four ranked correlation values is the output of the algorithm (Temperley 1999). To properly compute correlation, the input for this process also must be a vector with twelve values, each corresponding to a pitch class. In Krumhansl and Shmuckler's original formulation of the algorithm, these twelve values are meant to be a measure of the exact duration of each of the twelve pitches in the musical piece. These could be manually entered based on reading the actual sheet music, taken from MIDI data (as with an electronic keyboard), or based on processing of the audio signal. The twelve values represent the duration of each pitch irrespective of the octave they are played in, e.g. any C note in any octave will increase the value representing C by the duration of the note. This input vector can be taken from any musical segment of any length, although the number of notes included in the sample used to formulate the input may affect the accuracy of the algorithm. Krumhansl and Shmuckler suggest that in-depth analysis of longer, more complex pieces of music can be accomplished by running the algorithm on

multiple input vectors taken from different sections of the piece. In this way, macro level music theory concepts such as shifts in key and changes in tonal hierarchies throughout the piece can be detected and analyzed (Krumhansl 1990).

Krumhansl and Shmuckler tested their completed key-finding algorithm first on the fourty-eight preludes of J.S. Bach's *Well-Tempered Clavier*. These pieces all begin in the key of the key signature they are written in, and they cover all twenty-four major and minor keys, making them an ideal choice for initial testing. There was a study done by another psychologist, Annabel Cohen, where participants were played only the first four notes of twelve of the preludes and asked to sing the scale of the key that they thought it was in. They were accurate a majority of the time, implying that accurate key identifying information could be parsed from very short segments of music (Cohen, 1977). For this reason, Krumhansl and Shmuckler chose to test their algorithm using just the first four notes of the preludes as well. They ran the algorithm and analyzed the resulting correlation coefficients (r-values) based on whether they were statistically significant or not, whether the key of the piece being analyzed came up first in the ranking, and if not how far off it was from the top. In this first trial, they found that for each of the forty-eight preludes, the r-value for the key it was written in was statistically significant. Additionally, in all but four cases, the key of the piece was correctly identified as having the highest r-value- a 91.7% success rate. In the four cases where the key was misplaced, it was only off by an average of 1.40 places. Shmuckler also compared the algorithm she helped create with Cohen's earlier study that helped inspire it. In the twelve cases used in the psychological study, participants were only on average 75% accurate in identifying

the key in which the piece was written, making the algorithm over 15% more accurate than human perception in that case (Krumhansl 1990).

Krumhansl and Shmuckler tested their algorithm further with other collections of classical music written by Shostokavich and Chopin. With input vectors created from the first four notes of twenty-four Shostokavich preludes, the algorithm chose the correct key all but seven times, a 71% success rate. In the majority of those seven cases, however, the algorithm chose either a parallel or tonally related key. The Shostokavich pieces, like the Bach preludes, were chosen because of their clear tonal quality and relative adherence to the keys they are written in. Though not quite as accurate as the results from the Bach preludes, the results in both cases were most likely influenced by these qualities. In contrast, the twenty-four Chopin preludes used in testing are far more tonally ambiguous, often straying from the written key, as well as using dissonance and more chromatic tones (sharps and flats). This was a purposeful decision, to see the algorithm's effectiveness given a different, more complex style of music. Overall, on the Chopin preludes, the algorithm identified the correct key with a 45.8% success rate, a marked decrease from the earlier results. Additionally, in the cases where the correct key was not ranked first by the algorithm, it was on average further away from the top than in the incorrect cases of the Bach and Shostokavich preludes. Krumhansl and Shmuckler attempt to explain this drop in performance in several different ways, dividing the Chopin preludes into three categories based on the tonal qualities of the first four notes (Krumhansl 1990). Their reasoning for the algorithm's misidentifications in each case, as well as my own is discussed in the results section below.

VII. My Implementation

For the implementation of my own program based on the original Krumhansl-Shmuckler key-finding algorithm, I decided to code in Java since it is the high-level language that I am most comfortable with. It also has built in libraries for reading in and processing audio data, and third party math libraries for fast Fourier transforms. This made it an ideal choice for my purposes, though probably not the most efficient language I could have used. To start off, I decided to focus on reading in WAV files as inputs, since these are uncompressed audio and are easily converted to byte data for further processing. The code I wrote uses an AudioInputStream object from Java's sound libraries to read in the file, the name of which is taken as an argument when the program is run. Next I create and initialize a byte array with a default size of 1024 frames worth of bytes from the input file. This is then used as a buffer to store the audio data as it is read from the AudioInputStream. Next, I convert the byte array iteratively into an array of doubles, simply because the fast Fourier transform library I used utilizes doubles as inputs rather than bytes.

For the fast Fourier transform step, I downloaded and imported the Apache Commons Mathematics Library. This provided me with a relatively easy to use FastFourierTransformer object as well as a Complex data type to store the complex numbers coming out of the transform. The output of the fast Fourier transform run with the double array as an input is an array of complex numbers. I take the magnitude of each value with a basic absolute value formula for a complex number, which is the square root of the sum of the squares of the real and the imaginary parts respectively. The result of

this is an array with magnitude values stored in order by frequency. Each bin of the array represents a frequency range, which can be computed by multiplying the bin number by the Nyquist frequency (half the sampling rate of the file), divided by the number of bins (Burk et al. 2011). In this case, the number of relevant bins is only half the size of the array, so we can ignore the second half. This is because calculating the magnitude yields a complex conjugate symmetric array, but since the input was entirely real numbers, the complex parts can be ignored (Roche 2012).

Based on this, my first parsing of the output was to duplicate the array of magnitudes and sort it in descending order. Then I took the top ten highest magnitudes and searched the original, unsorted array with these to find their original bin number. Next I converted these ten magnitudes to their frequencies using the above method involving the Nyquist frequency, and listed the ten frequencies in descending order. This gave a rough idea of which frequencies were prominent in the music file, and by comparing these to the actual frequencies of musical notes; I was able to see which ten notes were most prominent in the sample, based on my analysis. This does not yield a musical key recommendation, however, but was merely an intermediary step to test if my array transformations were working as intended. Since they were, I then moved on to the next step, which was the key correlation step using the KS key-profiles.

My first move was to create an array holding the frequency values of eighty-eight piano notes spanning slightly more than eight octaves – from a double pedal A on the low end, up to an eighth octave C on the top. The eight octaves covered by this span is enough of a range to cover most if not all the notes used in any given musical piece (Krumhansl 1990). The creation of this array was necessary for the purpose of sorting the

frequency values taken from the FFT into a meaningful form with respect to actual musical notes. I accomplished this by creating set intervals within which a certain frequency would be considered to be a specific note. I used the midpoint between one note's frequency and the next note's as the breaking point for this sorting process- essentially rounding in whichever direction the value in question fell closest to. For example, an A1 (first octave A) has a frequency of 55Hz, and the next note, a Bb has a frequency of 58.2705. So I would find the midpoint between the two, and a given frequency value from the audio input that falls between 55Hz and 58.2705Hz would count for whichever side of the midpoint it falls on. I decided that an exact match with the midpoint would go to the lower note since notes at lower frequencies are less spaced out, but I doubt this consideration made any significant difference in the outcomes. Since there are only twelve distinct notes, the array of eighty-eight values represents eight octaves and four extra notes, one each of A, Bb, B, and C. Though this has potential to skew the results since those notes are slightly more represented in the sorting algorithm- eight distinct versions of a note are enough representation, and this should not make a significant difference. Once the sorting is complete, the result is an array of twelve values, each representing the prominence of each of the twelve pitches in the input file. This array is the ideal input vector for the KS algorithm.

To run Krumhansl and Shmuckler's algorithm, in addition to the input vector, I needed the key-profiles to be hard-coded into objects as well so that they could each be used for comparison. To this end, in a separate file I created a data structure, called a KSVector, which holds the name of the musical key it represents, an array of the twelve pitch values associated with that profile, and a correlation value with the song's input

vector. I initialized and created twenty-four of these objects, one for each musical key, using an iterative process to set the twelve pitch values, and initially setting the correlation value to zero. As described in the previous section on the KS algorithm, all the major key-profiles are simply different orderings of the same twelve pitch values, and the same is true for the minor key-profiles. Using this property, I was able to automate the creation of the twenty-four key-profiles by inputting the pitch values just one time each for the major and minor keys (starting with A) and then moving them by one place within the array (wrapping around), to get the ordering for the next consecutive key.

Once these twenty-four key-profiles are created, I put them into an array, and iterate through, setting the correlation value of each to the correlation with the song's input vector. This is found using the basic statistical correlation (or distance) formula. The array is then sorted in descending order by correlation value, and the results are printed with each key's name beside its value. I used a NumberFormat object to truncate the run-on digits past four for aesthetic value. The top key displayed being the algorithm's most preferred recommendation, the second being the second most preferred, and so on down the list.

VIII. Results

As test input to my algorithm, I tried to find high-quality samples from various genres of music- focusing on classical since the notion of key is most prominent there, and also because that is what Krumhansl and Shmuckler used to test their original algorithm. Since the inputs must be high-sampling rate and bit rate WAV files, clipped to

include only the relevant parts of the song, my selection pool was limited. Also, the process of removing a channel (since stereo has twice the relevant information), clipping the track to the proper size, and removing metadata proved to be time-consuming. For these reasons, I settled on using twenty, ten second clips of music ranging from rock, to funk, to electronica. I tried to select samples that well exemplified the key the music was written in, and also were in sections with low note density since I knew that overlapping notes would make the pitch detection far less accurate. Ten of the samples were the beginnings of selected preludes and fugues from the *Well Tempered Clavier*, the same collection of pieces that Krumhansl and Shmuckler used. The other ten were a mixture of clips taken from different parts of songs from my own music library. The results were by and large not statistically significant whatsoever, and in most cases the key was not correctly identified by my code. With the ten Bach pieces as input, the algorithm identified the correct key one out of ten times, and in three cases, the correct key was found in the top five suggested keys. However, in all of these cases, the R-value was so low that it would be considered a not statistically significant correlation. Table 1 shows the results by musical piece, R-value, and the placing of the actual key in the ranked list of twenty-four keys returned by the program.

Table 1: Results of ten samples from Bach's *Well Tempered Clavier*

Fugue (f) or Prelude (p)	**R-Value**	**Correct Key Placement**
Bb Major (f)	0.0156	5
C Major (f)	-0.0095	16
C Minor (f)	-0.0037	13

D Major (f)	-0.0913	22
D Minor (f)	0.0184	7
Bb Major (p)	0.0237	10
C Major (p)	0.0384	2
C Minor (p)	0.0398	1
D Major (p)	-0.0444	23
D Minor (p)	0.0167	8

With the song clips taken from other genres as input, the algorithm performed similarly. Out of the ten songs, none of them had the key perfectly identified; however two out of the ten had the correct key in the second spot on the list. Four out of the ten had their correct keys placed in the top five most correlated keys. Even so, the R-values were again outside any statistically significant range. It is worth noting that in one case where the correct key was off by one spot (Sick Muse), the correlation was quite high at 0.1329 and hovering around what could be considered a statistically significant positive correlation (Pearson's r Correlation 1999). Table 2 presents the results from those ten songs along with the song name and artist.

Table 2: Results of ten samples from various musical genres

Song Name (Artist)	**Key**	**R-Value**	**Correct Key Placement**
Sick Muse (Metric)	A Major	0.1329	2
Intro (Alt-J)	B Minor	0.0031	10

Breaking Ties (OceanLab)	G Minor	-0.0006	15
Annie Waits (Ben Folds)	C Major	0.0690	3
King of Carrot Flowers Pt. 1 (Neutral Milk Hotel)	C Major	0.0375	2
Maggot Brain (Funkadelic)	E Minor	-0.0004	11
Filmic (Above and Beyond)	C# Minor	0.0104	5
Piano Sonata No. 3 in B Minor (Chopin)	B Minor	0.0160	12
Guitar Flute & String (Moby)	G Major	-0.0123	13
Strobe (Deadmau5)	B Major	-0.1557	23

Despite the one positive and borderline significant R-value in this data set, the overall trend of inaccuracy is the same as with the classical pieces.

XI. Discussion

It is clear that in its current form, my key-finding algorithm falls rather short of its goal in terms of pure accuracy. It is nowhere near the accuracy that Krumhansl and Shmuckler's original tests demonstrated for their algorithm, even with the tonally ambiguous Chopin preludes as input. I believe the reasons for this lie mainly in the pitch recognition stage, i.e. computing the input vector before the algorithm is applied. On top of that my algorithm is also subject to the same pitfalls that the original is subject to, and the combination of these factors result in a largely inaccurate output.

In terms of pitch-recognition, I opted for a fairly simplistic method because anything more complex would involve highly math-based manipulations of audio data of which I am not well versed. One problem with the method I used is that simply running a

fast Fourier transform gives frequency values in the linear domain, but human pitch perception depends on pitches related logarithmically (Pauws 2004). Furthermore, the transform is not intelligent in terms of which frequencies are musical- it processes every sound equally. This is significant because there is a lot of very high frequency, inaudible spectral content in music that should not be part of the key-identification process. Additionally, the input vector to Krumhansl and Shmuckler's original algorithm is meant to be twelve values representing the exact note duration of each pitch class in the sample. In my case, I am computing the magnitude of certain frequencies over the sample and first correlating these frequencies to the frequencies of musical notes and then performing the key correlation. Since the magnitude values computed by my code are not equivalent to note duration, there is certainly an impact in the degree of correlation I found with the KS key profiles. I also used piano notes as a reasonable standard by which to associate frequencies with musical notes, but not all the samples I used were piano music. This is significant because different instruments produce sound waves with different timbre, or tonal coloring, and spectral quality (Müller et al. 2011). Finally, while I tried to choose music that was relatively simple and monophonic as input, this wasn't always possible or desirable since I wanted to have an idea of the algorithm's performance on a wide range of genres. Pitch detection on anything other than monophonic input becomes quite complicated since overlapping notes means frequencies that overlap and clash with each other. This, along with the other factors mentioned, certainly contributed to the inaccuracy of the pitch-detection stage, and consequently the formulation of the input vector to the key-correlation algorithm.

Outside the pitch-detection portion, even the original algorithm had its weaknesses, especially highlighted with complex inputs such as Chopin's preludes. For one, the algorithm is based on tonal hierarchies within a given key, and many keys are similar in this way. Therefore it is likely that the algorithm could misidentify the key by recognizing a tonally similar one, such as the parallel or relative major or minor. This was the case in many of the misidentifications Krumhansl and Shmuckler found with the Chopin pieces. In other cases, their input, the first four notes of the song, did not contain the tonic note of that key's scale, and therefore would not have indicated that key very strongly even to human perception. And in the two cases of the least accurately identified Chopin preludes, the first four notes contained notes not in the key at all (Krumhansl 1990). Though this is largely an issue with the selection of the input and could most likely be avoided by using more notes of the song, it still highlights a weakness of the algorithm. It is strongly dependent on one set context for a key, and does not cope well with pretention to other keys or modes, something that is fairly common in some styles of classical music.

There has also been suggestion by critics of the KS algorithm that the key profile values themselves are flawed and don't reflect the significance of each pitch within a key in a satisfactory way. Most of this is based on music theory concepts such as the prevalence of certain scale degrees in the major vs. the minor keys. An alternative set of key profile values, tweaked based on scale degrees has been suggested by David Temperley. He has tested it and found noticeably improvement over the old values (1999). Other suggestions for alternative key profile values have been made based mainly on the analysis of the tonal content in representative collections of music. For instance,

the prominence of chords in 100 works of classical music, or note counts from a collection of folk music (Sapp 2011). This may result in highly effective values for identifying key when given an input similar in style to the body of music used to create the key profiles in the first place. However, the original KS algorithm values or some variation thereof based on music theory, such as Temperley's suggestions, seem to be the best option for generalized key identification across multiple genres.

X. Further Development and Conclusion

As mentioned in the discussion section, I attribute most of the inaccuracy of my algorithm to my implementation of pitch recognition. Consequently, most of the developments I have the option of making are to this section of the code. In particular, I would implement a filtering process even before the data is transformed. This would help immensely with the issue of spectral noise in the form of high-level harmonics interfering with the Fourier transform. A low pass filter that simultaneously removes the unwanted upper frequencies and marginally downsamples the signal to improve computation time would be ideal (Pauws 2004). Along similar lines, there are methods of controlling for the different timbres produced by different instruments. This involves normalizing the input by essentially boosting up quieter frequencies, allowing the analysis to focus more on pitch and not the spectral signature created by the instrument (Müller et al. 2011). Additionally, I could operate on the data set prior to the Fourier transform to transform it into a logarithmic scale, similar to the process used by the Cepstrum analysis mentioned earlier. Since a large part of the accuracy of the algorithm, both in my experience, and in

Krumhansl's analysis, depends on the length and form of input, I could certainly improve my own results by changing the audio samples used for testing. Primarily, I would search for monophonic samples that stay solidly within a certain key, with useful musical passages as long as possible. The more data points given, the more accurate the algorithm seems to be. Additionally, I would explore the idea of using electronic music for testing since it utilizes sound generated directly by a computer, and therefore is not subject to the concerns involved with analog instrument's timbre or with recording noise. For simple testing on the lowest level of the pitch detection stage, inputting a pure sine wave generated by a synthesizer would be ideal. Improvements to the key correlation stage of the algorithm are also possible. For one, the implementation of alternate key-profile values as suggested by other researchers is a relatively simple tweak that I could explore. With some more music theory background I could formulate my own key-profiles for testing on different genres, which could have the concurrent effect of identifying patterns of key usage within a certain genre.

Overall, my foray into the field of algorithmic musical key recognition illuminated many of the complexities of audio signal processing, as well as the difficulty of computing something that is largely defined in terms of human perceptions. It allowed me to understand the complications involved with processing audio to detect musical pitch, and some of the solutions to these complications as well - though in terms of mathematical analysis, I have only scratched the surface. I was able to survey these precise methods of pitch detection and the specific music theory concepts behind different key-finding algorithms even though I couldn't implement them all. In terms of implementation, I created a complete musical key-finding program that takes an audio

sample as input and returns a ranked list of key suggestions. Even though the results from the limited tests that I was able to perform are highly inaccurate, I now have a full platform from which to explore further. My key-finding program has all the necessary components, input, signal processing for pitches, key correlation, and output, and is now open to be improved by specific changes to each part.

XII. References

Burk, Phil, Larry Polanski, Douglas Repetto, Mary Roberts, and Dan Rockmore. "Music and Computers: A Theoretical and Historical Approach." 1 Jan. 2011. Web.

Cohen, Annabel J.. "Tonality and Perception: Musical Scales Prompted by Excerpts From Das Wohl Temperierte Clavier of J. S. Bach." Second Workshop on Physical and Neuropsychological Foundations of Music. , Ossiach, Austria. Lecture.

Gerhard, David. "Pitch Extraction and Fundamental Frequency: History and Current Techniques." (2003). University of Regina. Web. <http://www.cs.uregina.ca/Research/Techreports/2003-06.pdf.>.

Harrison, Mark. *All About Music Theory: A Fun and Simple Guide to Understanding Music*. Milwaukee: Hal Leonard Corporation, 2009. Print.

Krumhansl, Carol L. *Cognitive Foundations of Musical Pitch*. New York: Oxford UP, 1990. Print.

McLeod, Philip, and Geoff Wyvill. "A Smarter Way to Find Pitch." (2005). University of Otago. Web. <http://www.cs.otago.ac.nz/tartini/papers/A_Smarter_Way_to_Find_Pitch.pdf>.

Müller, Meinard, Daniel P.W. Lewis, Anssi Klapuri, and Gael Richard. "Signal Processing for Music Analysis." *IEEE Journal of Selected Topics in Signal Processing* (2011): 1-23. Web. <http://www.ee.columbia.edu/~dpwe/pubs/MuEKR11-spmus.pdf>.

Parker, Barry R. *Good Vibrations: The Physics of Music*. Baltimore: Johns Hopkins UP, 2009. Print.

Pauws, Steffen. "Musical Key Extraction from Audio." (2004). Philips Research Laboratories Eindhoven. Web. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.9184&rep=rep1&type=pdf>.

"Pearson's R Correlation." Quinnipiac University. Web. <http://faculty.quinnipiac.edu/libarts/polsci/Statistics.html>.

Roche, Bjorn. "Frequency Detection Using the FFT (aka Pitch Tracking) With Source Code." 22 July 2012. Web.

Rowe, Robert. *Machine Musicianship*. Cambridge: MIT, 2001. Print.

Salamon, J., and E. Gomez. "Melody Extraction from Polyphonic Music Signals Using Pitch Contour Characteristics." *IEEE Transactions on Audio, Speech, and Language Processing* 20.6 (2012): 1759-1770. Web. <http://www.mtg.upf.edu/system/files/publications/SalamonGomezMelodyTASLP2012.pdf>.

Sapp, Craig Stuart. “Computational Methods For The Analysis Of Musical Structure.” (2011). Stanford University. Web. <http://purl.stanford.edu/br237mp4161>.

Shmulevich, Ilya, and Olli Yli-Harja. "Localized Key Finding: Algorithms and Applications." *Music Perception: An Interdisciplinary Journal* 17.4 (2000): 531-44. Web. <http://www.jstor.org/stable/40285832>.

Temperley, David. "What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered." *Music Perception: An Interdisciplinary Journal* 17.1 (1999): 65-100. Web. <http://theory.esm.rochester.edu/temperley/papers/temperley-mp99.pdf>.

White, Dan. "Key Detection Software Comparison: 2014 Edition." *DJ Techtools*. 14 Jan. 2014. Web.

Zhu, Yongwei, and M.s. Kankanhalli. "Precise Pitch Profile Feature Extraction From Musical Audio for Key Detection." *IEEE Transactions on Multimedia* 8.3 (2006): 575-84. Web. <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1632042>.