Audible Frequency Detection and Classification

David E. Gray, W. Craig McKnight, Stephen R. Wood

Audible Frequency Detection and Classification Group
Department of Electrical and Computer Engineering
Mississippi State University
216 Simrall, Hardy Rd.
Mississippi State, Mississippi 39762
{gray, wcm1, srw1}@ece.msstate.edu

1. Abstract

Current music software relies on external input from MIDI capable devices. Because traditional musical instruments are inherently analog, the interaction of musicians and computers is rare.

The purpose of this project is to develop a software package for music education utilizing an acoustical instrument interface so that players of all instruments can begin to utilize the computing power of today's world. Musicians who play tones into a microphone will see those tones analyzed in the areas of relative and absolute pitch.

2. Overview

2.1 A Brief History of Computers in Music

There has been an interest in using computers to aid in the creation of music for more than thirty years. Bell Telephone Laboratories developed a program called *Music 4* as early as the 1960's. Through various updates and revisions, this program eventually developed into what is now *CSound*. *CSound* allows the user to write programs that represent arrangements of music for different instruments that will be simulated by the computer. One of the new features of the program allows the user to input information into the computer via a MIDI (Musical Instrument Digital Interface) equipped instrument [1].

One area where the use of computers in music holds great promise is in education. *Listen,* by Imaja, is an educational software package that teaches students relative pitch, harmony, intervals between notes, and chord structure in an interactive environment. For example, the computer would give an interval to a student studying intervals. The student would then play the interval on the keyboard or other MIDI instrument and the computer would tell the student if the interval played was correct or if one of the notes was too high or too low. This package relies on MIDI codes being sent to the computer from a MIDI capable device [2].

MIDI codes are a form of communication protocol decided upon by the music manufacturing industry. The codes are transmitted serially at a 31.25 K bits/s data rate and contain information about which key was played, when the event started, when the event stopped, what voice patch to use, and other aesthetics involved with playing a note [3] [4].

In the case of a piano style keyboard being connected to a computer using MIDI, the information traveling to the computer is generated digitally, transmitted digitally, and manipulated on a digital computer. Throughout the entire signal chain from the instrument to the computer, the signal is never analog. This is an obstacle for musical instruments in general since most are inherently analog. MIDI controllers for instruments such as the guitar have been developed and in some cases the entire instruments themselves have been designed as the controllers, but an interface to accommodate all instruments without changing hardware is not in wide use.

The goal of this project is to create a program that uses a computer to recognize musical notes originating from an analog source. Users of the program are able to play notes on an instrument into a microphone. They are then informed of what note was played and if the note was flat, sharp, or in tune. Another option allows the user to learn about the intervals between a group of notes. The notes can either be played simultaneously or sequentially.

This project could serve as foundational research for the creation of a complete music education and notation

software package for instruments that are not MIDI capable.

2.2 Overall System Algorithm

Figure 1 illustrates the process by which musical notes are analyzed with the software developed in this project. A musical, analog signal is played into a microphone or other analog transducer. The analog signal is then converted to a digital signal sampled at a given rate by the recording feature of *Network Audio*.

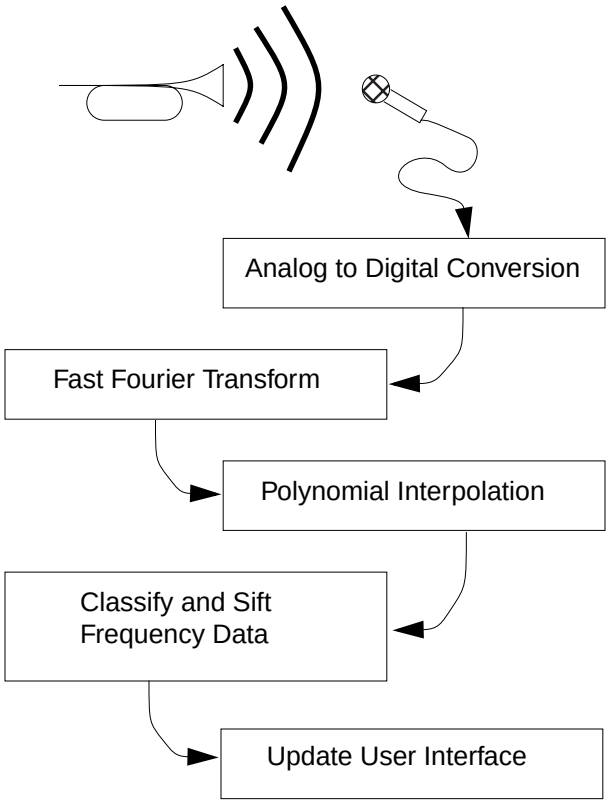

Figure 1: Block diagram of the implemented system showing the main steps in analyzing the musical signal.

After sampling, a Fast Fourier Transform is used to convert the time domain signal into the frequency domain. Once the signal is in the frequency domain, the peaks in the frequency spectrum are found and passed to a polynomial interpolation routine to increase the accuracy of the program. A list of possible notes is then built. The possible notes in the note list are classified as notes if they pass given criteria corresponding to magnitude and overtone patterns that imply musical signals.

Once the note list has been created, each note is given a note name corresponding to its frequency. Depending on the mode of operation, notes are found to be in tune or not, or the intervals between notes played either sequentially or simultaneously are determined.

3. Musical Signals

3.1 Musical Notes

When describing or measuring musical notes, only a single frequency is usually mentioned. For example, the musical note A4 is generally accepted to be the frequency corresponding to 440 Hz. However, when a single musical note is played on an instrument, more than one frequency is produced. The additional frequencies occur at integer multiples of the lowest frequency, which is the frequency used to name the note (See Figure 2).

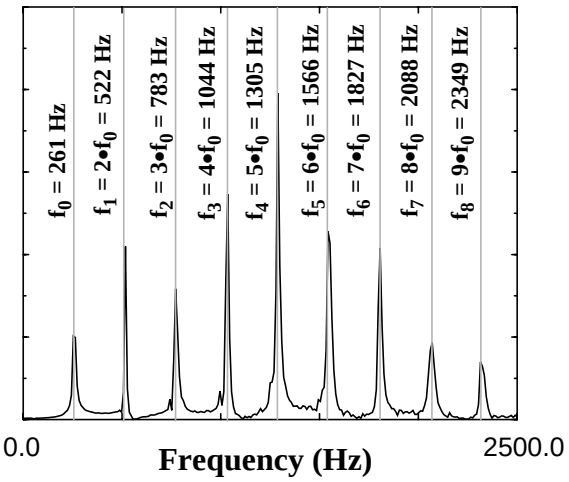

Figure 2: Frequency Spectrum of a C4 played on a Trumpet. Notice the integer relationship of each overtone frequency to the fundamental frequency at 261 Hz.

The lowest frequency is called the fundamental frequency and the frequencies occurring at integer multiples of the fundamental are called harmonics or overtones. Given the frequency, f_f, of the fundamental, the frequency of the n^{th} overtone can be calculated as follows:

$$f_n = (n+1) f_f \quad (1)$$

The relative amplitude of the different harmonics produced varies from instrument to instrument and this amplitude pattern defines the *timbre* of an instrument. Timbre is the quality, or color, of a sound based on a harmonic series.

Because musical notes produce a particular pattern of frequencies and because the energy of noise is distributed randomly throughout the frequency spectrum, it is possible to distinguish musical notes from noise [5]. Characteristic spectra for white noise, random noise and a musical note are shown in Figure 3. The distinguishing characteristic of the overtone pattern was used to identify and classify notes in this project.

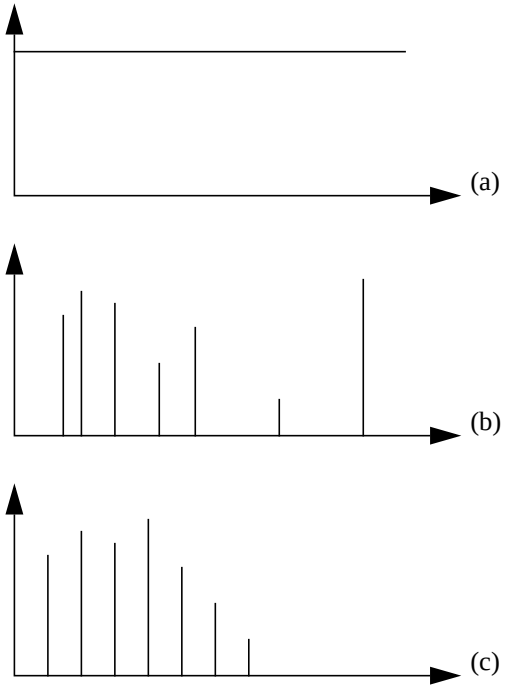

Figure 3: Frequency spectra of (a) white noise (b) noise with irregular energy concentration (c) musical note.

3.2 Even-Tempered Tuning

Perhaps the most important interval in music is the octave. The octave serves to define the musical scale. It is defined as the interval between two notes where the higher note is exactly twice the frequency of the lower note. In Western music, there are 12 notes, called semitones, which divide up the range between a note and its octave counter part. Patterns of these notes played sequentially make up a musical scale. The distance between two adjacent notes is called a semitone and the distance between two notes with one in between is called a tone or whole tone.

The major scale is defined by the following pattern of tones and semitones: T, T, S, T, T, T, S. If the eight frequencies representing the notes in the scale are picked to be musically pleasing, the notes are tuned to *ideal temperament*. The fourth column of Table 1 shows the ratios of the frequency of notes in the major scale in relation to the root or lowest note of the scale [6] [7].

Table 1: Comparison of Equal and Ideal Temperament for C Major Scale

Note Name	Equal Temperament		Ideal Temperament
	Ratio	Frequency	Ratio
C	1.0000	261.63	1.0000
C#	1.0595	277.18	
D	1.1225	293.66	1.1250
D#	1.1892	311.13	
E	1.2599	329.63	1.2500
F	1.3348	349.23	$1.333\overline{3}$
F#	1.4142	369.99	
G	1.4983	391.99	1.5000
G#	1.5874	415.31	
A	1.6818	440.00	$1.666\overline{6}$
A#	1.7818	466.16	
B	1.8877	493.88	1.8750
C	2.0000	523.25	2.0000

A problem is encountered when an instrument is tuned using ideal temperament ratios. If the instrument is tuned to the key of C and all the notes in between C and its corresponding octave obey the ratios of ideal temperament, the instrument will perform well for music written in the key of C. However, if a piece of music is to be played in a different key using the ideal temperament tuning for the key of C, the ratios between the notes for the new key do not correspond to ideal temperament for the key of C. For example, the difference in the ratio values for the notes C and D is

1.125 - 1.000 = 0.125. If a piece of music was in the key of G, the difference in the ratios between the notes of G and A is $1.66\overline{6}$ - 1.500 = $0.16\overline{6}$, which is clearly different. Therefore, using ideal temperament tuning, an instrument would have to be retuned anytime it played in a different key. This is quite undesirable and makes playing a piece of music that changes keys impossible.

To remedy the tuning problem, a tuning method called *even temperament* was derived. Even temperament tuning divides the distance between semitones such that each semitone is a factor K larger than the previous semitone. That is,

$$f_1 = K \bullet f_0 \qquad (2)$$

where $K = \sqrt[12]{2} \approx 1.05946$.

While no note is exactly in tune using this system, each note is very close to being in tune. This allows musicians to play in various keys without having to retune their instruments. The resulting note frequency combinations are given in Table 2. The even tempered scale was used for classifying notes in the project because it is the accepted tuning scheme for Western music.

The range of notes listed in Table 2 contains the full range of a piano tuned using even tempered tuning. The area of interest for this project was limited to the notes between 80 Hz, a flat E2, and 2534 Hz, a sharp D#7. This restriction was made in the interest of frequency resolution which is discussed in Section 4. The musical notes not included (shaded in light gray) are on the extremes of the frequency range of piano and thus, used less often. Only frequencies within the unshaded range (the range of interest for this project) were classified as notes.

Table 2: Frequency Values for Even-Tempered Piano Scale

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	--	--	--	--	--	--	--	--	--	27.5	29.13524	30.86771
1	32.7032	34.64783	36.7081	38.89087	41.20344	43.65353	46.2493	48.99943	51.91309	55.0	58.27047	61.73541
2	65.40639	69.29566	73.41619	77.78175	82.40689	87.30706	92.49861	97.99886	103.8262	110.0	116.5409	123.4708
3	130.8128	138.5913	146.8324	155.5635	164.8138	174.6141	184.9972	195.9977	207.6523	220.0	233.0819	246.9417
4	261.6256	277.1826	293.6648	311.127	329.6276	349.2282	369.9944	391.9954	415.3047	440.0	466.1638	493.8833
5	523.2511	554.3653	587.3295	622.254	659.2551	698.4565	739.9888	783.9909	830.6094	880.0	932.3275	987.7666
6	1046.502	1108.731	1174.659	1244.508	1318.51	1396.913	1479.978	1567.982	1661.219	1760.0	1864.655	1975.533
7	2093.005	2217.461	2349.318	2489.016	2637.02	2793.826	2959.955	3135.963	3322.438	3520.0	3729.31	3951.066
8	4186.009	--	--	--	--	--	--	--	--	--	--	--

4. Internal Algorithms

4.1 Sample Frequency

The first consideration of the project was to decide on the frequency at which to sample the analog input signal. If an analog signal containing a maximum frequency, f_{max}, is to be recovered without aliasing, it must be sampled at a rate greater than its Nyquist rate, f_N [8]. The Nyquist rate for a signal is defined by the following equation:

$$f_N = 2 \bullet f_{max} \tag{3}$$

As stated in Section 3.2, the highest note that will be recognized is a D#7. The frequency for this note is 2489.02 Hz. For best performance, the first overtone of this note should also be detectable. The first overtone of a D#7 can be calculated from Equation (1) to be 4978.04 Hz. Therefore, the Nyquist rate is determined to be 9956.08 Hz. This is the minimum sample frequency.

Once the analog input signal is sampled, it will be transformed to the frequency domain via a Fast Fourier Transform (FFT). The frequency resolution for an FFT is defined as

$$\Delta f = \frac{f_s}{N} \tag{4}$$

where f_s is the sample frequency and N is the number of points of the FFT [9].

The peaks in the frequency spectrum of a signal that is transformed with an FFT appear to occur at integer multiples of the resolution. Therefore, the desired frequency resolution must also be considered along with the Nyquist rate when determining the sample frequency.

There are many algorithms that implement an FFT [10]. The Decimation in Time and Frequency (DITF) algorithm was implemented first [11]. This algorithm proved to be extremely slow when performing a 1024 point transform. For the project to execute in real time, a faster algorithm was necessary. The algorithm known as the Quick Fourier Transform (QFT) was tested to see if it had better speed performance [12]. For our application, and for 1024 points, the QFT was more than 10 times faster than the DITF. This speed enhancement was sufficient for the project to perform in real time; therefore, the QFT was implemented in the project.

The sample frequency was first set at 20 kHz. This choice would have allowed more overtones of the D#7 to be detected. However, from Equation (4), it would only offer a frequency resolution of 19.53 Hz. For the low end of the frequency range defined in Table 2, this resolution would allow up to three semitones to have a peak in the frequency spectrum at the same frequency value. For example, the frequency spectrum of an F#2, G2, and G#2 would all indicate a maximum at 97.66 Hz. This would not be acceptable.

The first two notes in the frequency detection range differ by approximately 5 Hz. Therefore, a resolution of at least 5 Hz would be needed in order for each note to have a maximum at a unique frequency. From Equation (4), this would require a sample frequency of 5120 Hz. However, the Nyquist rate, as discussed previously, sets a lower limit for the sample frequency of 9956 Hz. Therefore, the sample frequency was chosen to be 10 kHz.

The frequency resolution obtained by using a sample frequency of 10 kHz and a 1024 point FFT is given by Equation (4) to be 9.765625 Hz. This resolution will not provide a unique maximum point in the frequency spectrum for all the notes in the desired detectable range. Polynomial interpolation schemes provide a solution to this deficiency.

4.2 Polynomial Interpolation

The data points $(x_1,y_1),(x_2,y_2),...,(x_n,y_n)$ for a function, f(x), are the necessary information needed to compute an interpolating Polynomial, p(x), of order n-1. This polynomial will be unique and will agree with f(x) for all data points. A polynomial interpolation scheme is said to have Lagrange form if it can be written as

$$P_L(x) = \sum_{k=1}^{n} y_k L_k(x) \tag{5}$$

L_k represents a family of polynomials of degree n-1 which satisfy

$$L_k(x_j) = \begin{Bmatrix} 0, k \neq j \\ 1, k = j \end{Bmatrix} \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{6}$$

This definition insures that $p_L(x_j) = f(x_j) = x_j$ for j = 1,...,n.

For some arbitrary value of x,

$$L_k(x) = \prod_{\substack{j=1 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j} \tag{7}$$

Evaluation of $p_L(x)$ will require n+1 evaluations of L(x). The number of multiplies required to compute $p_L(x)$ is $n^2 + 2n - 1$ and the number of additions is $n^2 + n - 1$ [13].

If the data points $(x_1,y_1,y_1'),(x_2,y_2,y_2'),\ldots,(x_n,y_n,y_n')$ are known for a function, f(x), then it is possible to compute $p_H(x)$ of order 2n-1. This interpolating polynomial has Hermite form [13]. The Hermite polynomial satisfies the conditions $p_H(x_i) = f(x_i)$ as well as $p_H'(x_i) = f'(x_i)$ for all i = 1,2,...n.

The Hermite polynomial is represented by the following equation:

$$p_H(x) = \sum_{k=1}^{n} H_k(x) f(x_k) + \sum_{k=1}^{n} \hat{H}_k(x) f'(x_k) \tag{8}$$

where

$$H_k(x_j) = \begin{Bmatrix} 0, k \neq j \\ 1, k = j \end{Bmatrix} \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{9}$$

$$H_k'(x_j) = 0 \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{10}$$

$$\hat{H}_k'(x_j) = \begin{Bmatrix} 0, k \neq j \\ 1, k = j \end{Bmatrix} \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{11}$$

$$\hat{H}_k(x_j) = 0 \quad \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \tag{12}$$

For arbitrary values of x, the value of L_k defined in Equation (7) is used to define the value of H_k and $\hat{H}_k$ in the following manner:

$$H_k(x) = [1 - 2(x - x_k)L_k'(x_k)]L_k^2(x) \tag{13}$$

$$\hat{H}_k(x) = (x - x_k)L_k^2(x) \tag{14}$$

A 5-point Lagrange interpolation and a 3-point Hermite interpolation were tested to determine the increased accuracy that could be obtained by interpolating the data from the frequency spectrum. The goal of the Hermite scheme was to interpolate using three consecutive points (x_0,y_0), (x_1,y_1), and (x_2,y_2), where (x_1,y_1) is the maximum point in the frequency spectrum. However, the Hermite scheme requires knowledge of the derivative of f(x), as seen by Equation (8). This data is not directly available in the frequency spectrum. To produce a Hermite polynomial, the derivative at the points of interest was estimated using the centered difference formula for numerical differentiation given below [13]:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} \tag{15}$$

where h is the distance between consecutive data points and is the resolution of the FFT in this context.

For the Lagrange interpolation scheme, five consecutive data points, (x_0,y_0), (x_1,y_1), (x_2,y_2), (x_3,y_3), and (x_4,y_4) were used where (x_2,y_2) is the maximum point in the frequency spectrum.

Since the resolution of the FFT is approximately 10 Hz, the true frequency of the input signal will be within 5 Hz of the maximum in the frequency spectrum. Both of the interpolating polynomials were evaluated at 0.1 Hz increments across a 10 Hz range centered about the maximum data point. From this data, the maximum of

the interpolation polynomial is determined with an accuracy of 0.1 Hz.

The two interpolation routines were tested with computer generated sine waves of known frequency. The results are listed in Table 3

Table 3: Comparison among FFT, Lagrange, and Hermite

Actual Peak	FFT	5-point Lagrange	3-point Hermite
82.0	78.1	80.1	79.3
154.0	156.3	155.9	156.0
155.0	156.3	156.1	156.2
156.0	156.3	156.2	156.3
157.0	156.3	156.3	156.3
158.0	156.3	156.4	156.3
438.0	439.5	439.3	439.4
440.0	439.5	439.5	439.5
442.0	439.5	440.0	439.7
582.0	585.9	584.0	584.8
584.0	585.9	585.7	585.8
586.0	585.9	585.9	585.9
588.0	585.9	586.2	586.1
590.0	585.9	588.1	587.3
650.0	654.3	651.5	652.5
652.0	654.3	653.9	654.1
654.0	654.3	654.3	654.3

This data indicates that the 3-point Hermite polynomial never performs better than the 5-point Lagrange polynomial. The Hermite is also computationally more expensive, since it computes L_k as well as L_k' for $k = 1,...,n$ and must estimate 3 derivatives. Therefore, a 5-point Lagrange interpolating polynomial is used to determine the maximum point in the frequency spectrum more accurately.

The Lagrange interpolating algorithm was also tested on a range of frequencies given in Table 4. This table indicates the increased accuracy gained by interpolating as well as the error that still occurs in interpolated values.

Also, Table 4 shows that the FFT produces data points at 976.56 Hz and 986.33 Hz. Interpolation is most beneficial when the frequency to be detected is approximately half way between consecutive FFT data points. These results are illustrated below in Figure 4.

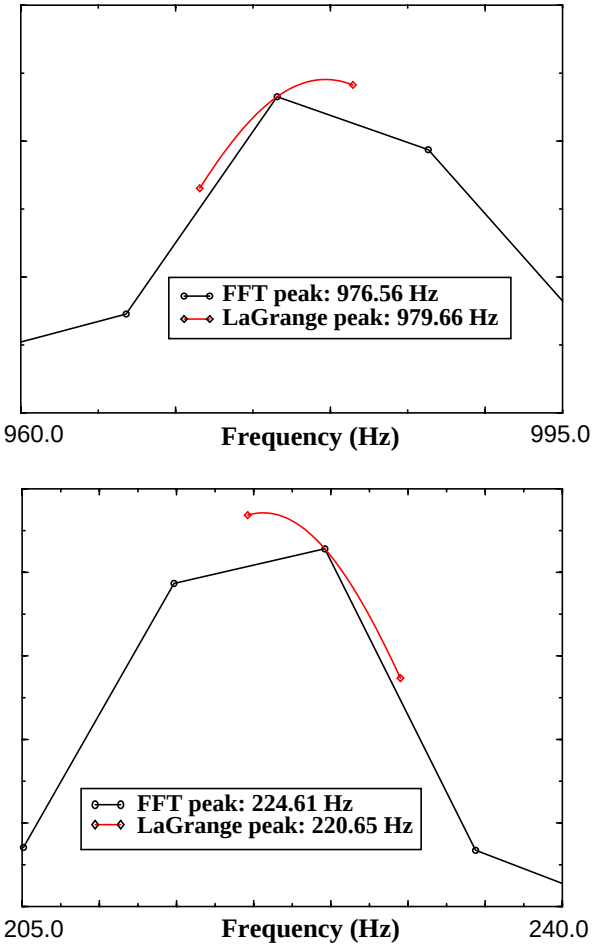

Figure 4: Two graphs comparing the accuracy of frequency data with and without Lagrange interpolation. (top) 981 Hz peak generated with a function generator. (bottom) A3, which is 220 Hz, played on a bass guitar.

A higher number of points could have been used in the calculation of the FFT in order to produce a better resolution as opposed to interpolation. However, even if this option is computationally feasible in real time, it is not as desirable as interpolation. If 2048 points were used, the resolution would be 4.88 Hz. A note that is one to two Hz sharp or flat would be likely to have the same maximum in the frequency spectrum as if it were played in tune. By interpolating the peak, however, it is

Table 4: Uninterpolated Frequency Values and Interpolated Frequency Values Using 5- point Lagrange Interpolation for Δf=9.766 Hz

Generated Frequency (Hz)	Uninterpolated Frequency (Hz)	Interpolated Frequency (Hz)	Uninterpolated Difference (Hz)	Interpolated Difference (Hz)	Error Ratio Magnitude
975.0	976.56	976.36	+1.56	+1.36	1.15
976.0	976.56	976.56	+0.56	+0.56	1.00
977.0	976.56	976.56	-0.44	-0.44	1.00
978.0	976.56	976.66	-1.44	-1.34	1.08
979.0	976.56	977.06	-2.44	-1.94	1.26
980.0	976.56	977.76	-3.44	-2.24	1.54
981.0	976.56	979.66	-4.44	-1.34	3.31
982.0	986.33	983.53	+4.33	+1.53	2.83
983.0	986.33	985.23	+3.33	+2.23	1.49
984.0	986.33	985.93	+2.33	+1.93	1.21
985.0	986.33	986.23	+1.33	+1.23	1.08
986.0	986.33	986.33	+0.33	+0.33	1.00
987.0	986.33	986.33	-0.67	-0.67	1.00

likely that a difference of one to two Hz in the signal will be discernible.

4.3 Sifting Algorithm

A sifting algorithm was implemented to identify and classify the notes that were played. The course of the algorithm is as follows. The frequency spectrum derived from the FFT is scanned to find peaks above a certain amplitude threshold. Once a peak is found, its frequency is compared with other peaks already placed in the note list to determine if it is an integer multiple of these peaks. If so the peak is classified as a harmonic or overtone of that note and placed in its array of overtones. The algorithm then progresses to see if the peak might be an overtone of any of the other possible notes in the note list. If, once the note list has been traversed, the peak has not been classified as an overtone, it is classified as a possible note.

Once all of the peaks out of the frequency spectrum have been classified as possible notes or overtones, the note list is scanned to eliminate any peaks which might be noise. To accomplish this, the amplitude of the peak and the number of peaks placed in its overtone array are tested.

To check the amplitude of the possible notes, the magnitude of the highest peak is used to normalize the magnitude of all of the peaks. If a peak does not have a magnitude of 0.15 once normalized and the peak doesn't have at least two overtones, the peak is classified as noise and is eliminated from the note list. If a peak in the note list satisfies both of these qualifications, then it is considered to be a note and is passed back to the calling function to be named.

4.4 Tuning Overtones

The data in Table 3 and Table 4 indicate that the interpolation algorithm detects a frequency with an absolute error less than 2.25 Hz. This error is quite significant in the lower end of the desired detection range. At higher frequencies, however, the error will be tolerable. Perhaps an example will illustrate this point.

A musician plays a note at 110 Hz, which is an A2 perfectly in tune. The minimum value that the

interpolation routine would indicate for this tone is 107.75 Hz. By searching a database, it is known that a frequency of 107.75 Hz should be 110 Hz to be in tune. The tuning quality of this note is calculated by

$$Q_f = \frac{f_c - f_T}{f_T - \left(\frac{f_T}{\sqrt[12]{2}}\right)} \qquad (16)$$

$$Q_s = \frac{f_c - f_T}{(\sqrt[12]{2})(f_T) - f_T} \qquad (17)$$

where Q_f is the quality of a note if it is flat, Q_s is the quality of a note if it is sharp, f_c is the frequency of the note played by the instrument, and f_T is the frequency of the note in even-tempered tuning that most closely matches f_c.

For the example under discussion, Equation (16) yields a quality of -0.36. The magnitude of the quality must be greater than or equal to 0.05 to be classified as in tune. Therefore, the tone played is characterized as very flat, when it was actually perfectly in tune.

The musician plays another note at 880 Hz, which is an A5 perfectly in tune. The minimum interpolated value for this tone is 877.75 Hz. Using Equation (16), the quality of this tone is computed as -0.05 which would be characterized as in tune.

With this in mind, and remembering that tones have a predictable harmonic structure, tuning will be more accurate by tuning a particular overtone. Continuing the example, the musician again plays a note at 110 Hz. The seventh overtone of this tone is 880 Hz. Searching the database for the fundamental tone, returns that the tone should be an A2. Then, by tuning the seventh overtone, the fundamental can be classified as in tune.

This algorithm is solid as long as the overtone which is tuned is also a note in the database. Since the 2.25 Hz maximum error associated with the interpolation algorithm becomes less significant for each successive overtone, tuning to the highest detected overtone will provide the best result. However, only the 2^n-1 overtones of a tone correspond directly to notes in the database of notes found in even-tempered tuning.

This problem was investigated further by looking at the overtones for every tone in the desired detection range. All the overtones that occur within the bounds of the maximum frequency in the database were examined. The frequency of each overtone was determined and then the database was searched to see which tone most closely matched the overtone in frequency value. Then Equation (16) or Equation (17) was used to see how the overtone related to the tone found in the database.

Continuing the example for an A2, The eighth overtone occurs at 990 Hz. In even tempered tuning, no tone occurs at 990.00 Hz. If the database is searched for this value, a B5 will be returned which has frequency value 987.77 Hz. The quality of this note would be calculated as +0.0380. This quality represents an inherent error, ε, that is introduced when tuning the eighth overtone of an A2. If the 990 Hz frequency had been a fundamental tone, then the quality would accurately indicate that the tone was slightly sharp.

The data collected from this process was compiled and analyzed to see a remarkable trend. The error associated with a particular overtone is independent of the fundamental tone. The eighth overtone of any in tune note has a quality of 0.0380. Table 5 shows the error corresponding to a given overtone.

A fundamental tone, f_f, will have an overtone frequency, f_o, that will be used for tuning f_f. f_o will be compared to a frequency, f_c, from the database. If the inherent error, ε, is greater than 0, it represents the fraction of the difference between f_c and the next note in even tempered tuning that must be added to f_c in order to get f_o. This is seen by Equation (18).

$$f_0 = f_c + \varepsilon[(\sqrt[12]{2})(f_c) - f_c] \qquad (18)$$

If ε is less than 0, it represents the fraction of the difference between f_c and the previous note in even tempered tuning that must be subtracted from f_c in order to get f_o. This is seen by Equation (19).

$$f_0 = f_c - |\varepsilon|\left(f_c - \frac{f_c}{\sqrt[12]{2}}\right) \qquad (19)$$

Equations (20) and (21) are derived by solving Equations (18) and (19) respectively for f_c.

Table 5: Deviation of Overtones from Even-Tempered Scale

Overtone Number	Deviation from Even-Tempered Scale (%)
0	0.00
1	0.1021
2	1.9750
3	0.0511
4	-13.1604
5	1.9750
6	-29.9504
7	0.0511
8	3.8636
9	-13.1805
10	-46.5735
11	1.9580

$$f_c = \frac{f_0}{1 + |\varepsilon|(\sqrt[12]{2} - 1)} \qquad (20)$$

$$f_c = \frac{f_0}{1 + |\varepsilon|(1/(\sqrt[12]{2}) - 1)} \qquad (21)$$

If the overtone number, n, associated with the frequency f_o that is to be used in determining the quality of the fundamental tone, f_f, is known, then these equations allow f_o to be modified to a corrected value f_c by using the ε associated with the given n. Then, the new frequency, f_c, can be used for determining the tuning quality. When this frequency is compared to a frequency in the database, f_T, there will be no inherent error, ε, introduced. *Q*, the quality given by Equation (16) or (17), will accurately represent the quality of f_f.

4.5 Interval Detection

Musicians are often interested in the groups of notes or chords that they are playing. An option to identify the musical interval between two notes was implemented. When called for the first time, the interval detection code creates an interval list of all the notes currently in the note list. The interval between each of the notes in the interval list is then calculated.

Each successive time that the interval detection code is called, the note list and the interval list are compared to see if any new notes have been played. New notes are added to the current interval list and all intervals are recalculated. A function outside the interval detection code clears the interval list when the user desires to investigate a different combination of notes.

5. Project Database

To test the program during the process of development, a database was created. It consists of sine waves generated by a digital function generator and musical notes digitally recorded using a microphone and a sampling program called *Network Audio*. Files in the database range from containing single frequency data without instrument overtones to multiple frequency data with instrument overtones.

File names for the database were standardized to the following format: **<source>_<# of principle frequencies>_<tuning><note>**.raw. **<source>** is a three-character abbreviation for the instrument. **<# of principle frequencies>** is a numeral denoting the number of principle frequencies (i.e., the number of notes played) in the sample. **<tuning>** is a single-character denoting whether the sample is sharp, flat, or in tune: +, -, ~, respectively. **<note>** is a three-character denotation of the note name in the sample. For example, an A220 were represented as A3~. The file name for a sample collected from a trumpet playing a sharp A440 would be: tru_1_+A4~.raw. If the sample were a C261, but were in tune: tru_1_~C4~.raw. For a sample from an instrument capable of playing multiple tones at once, such as a guitar playing sharp A440 and in tune C#554 above it would be: acg_2_+A4~_~C5+.raw.

The archive was constructed of data to be used in three phases of the project. The first phase contained single tones to test the tuner function of the software. For

Phase I of the project, three instruments were used in the data archive: a guitar, a bass guitar and a trumpet. Each instrument was recorded in three sets of cases. For each case, a note was played in tune, then that note was played out of tune flat, and then it was played out of tune sharp. Phase I of the archive contains 36 samples in the range of interest.

In Phase II, two instruments, the guitar and bass guitar, were each recorded playing each of the twelve intervals less than an octave. Phase II of the archive contains 24 samples for interval testing.

Phase III of the archive contains sequential notes designed to test if the software is effective in detecting note changes. Again, three instruments were used, the guitar, bass guitar, and trumpet. For each file, a sequence of three notes was recorded. Each instrument played three sequences of intervals. The Phase III archive contains 27 samples.

Phase IV is an archive of chord data recorded from the guitar and the bass guitar. Each instrument played a major, minor and diminished triad based on three tones. The Phase IV archive contains 18 samples.

All of the data in the database was sampled at 20 KHz. Once the decision was made to run the program at a sampling rate of 10 KHz, the database files were down sampled to 10 KHz. This was accomplished through the use of a program called *sox*.

6. Results

Of 36 Phase I files, we named the correct note 88% of the time. Of these, 62.2% were tuned correctly. The note name in the acoustic guitar files were identified correctly 100% of the time and the tuning was correct 90.3% of these times. The note name in the bass guitar files were identified correctly 80.7% of the time and the tuning was correct 52% of these time. The note name for the trumpet files was identified correctly 87% of the time and the tuning was correct 83% of these times. This indicates that our algorithm is effective at frequencies greater than 130 Hz, but is less effective at lower frequencies.

Of 24 Phase II files, 33% of the notes and intervals were identified correctly. For any given two notes, the interval was always classified correctly. Of the 12 acoustic guitar files, 67% were correct. The four files that did not yield correct results could be special cases of overtone interaction or could be attributed to inaccurate naming of the files. None of the files for the bass guitar yielded correct results. The lower tone of each file was recorded an octave too low, placing it out of the range of interest. This accounts for the poor performance for these files.

Of the nine files in Phase III, all three notes in six of the files were identified correctly without any problems. One bass guitar file contains notes that were outside of the range of interest as defined by Table 2. Another bass guitar file causes an error in the software. Listening to this file indicates that there was a large amount of distortion present and this could account for the software error.

The files in Phase IV were not evaluated critically. However, the results for the Phase IV data are expected to be comparable to the results of the Phase II data. The addition of notes played simultaneously is not expected to adversely affect the performance of the software.

7. Future Research

It might prove useful to investigate frequency domain representations of our signals other than the FFT. Some alternatives might be spectral estimation and Prony's method [14] [15]. Estimating the spectrum with these techniques could eliminate the need for polynomial interpolation.

Reducing computation time for one spectrum will be key to integrating the software into a real-time music notation program. Faster FFT algorithms might yield this. Also the spectral estimation techniques could do this.

8. References

1. B. Vercoe, *MIT Media Lab Csound Manual*. http://www.leeds.ac.uk/music/Man/Csound/pref.
2. *Listen's Sound and MIDI Features*. http://www.imaja.com/Listen.html.
3. H. Chamberlin, *Musical Applications of Microprocessors*. Hayden Books., Hasbrouck Heights, NJ, USA, 1985.
4. S. Lehman, *Harmony Central: MIDI Tools and Resources*. http://www.harmony-central.com/MIDI/.
5. J. C. Brown, "Musical fundamental frequency tracking using a pattern recognition method." *The Journal of the Acoustical Society of America Vol 92, No. 3*. American Institute of Physics, Woodbury, NY, USA, 1994.
6. A. H. Benade, *Horns, Strings, and Harmony.* Dover, NY, 1960.
7. C. Taylor, *Exploring Music: The Science and Technology of Tones and Tunes*. Institute of Physics Publishing, Bristol, England, 1994.
8. J. G. Proakis, D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications Third Edition*. Prentice Hall, Upper Saddle River, NJ, USA, 1996.
9. R. E. Ziemer, W. H. Tranter, and D. R. Fannin, *Signals and Systems: Continuous and Discrete, 3rd Edition*. MacMillan, New York, NY, 1993.
10. W. W. Smith and J. M. Smith, *Handbook of Real Time Fast Fourier Transforms*. Institute of Electrical and Electronic Engineers, Inc., New York, NY, 1995.
11. Saidi, Ali. "Decimation-In-Time-Frequency FFT Algorithm." *ICASSP94 Digital Signal Processing. vol III*. Institute for Electrical and Electronics Engineers. New York, NY, 1994.
12. H. Guo, G.A. Sitton, C.S. Burrus. "The Quick Discrete Fourier Transform." *ICASSP94 Digital Signal Processing. vol III*. Institute for Electrical and Electronics Engineers, New York, NY, 1994.
13. N. S. Asaithambi, *Numerical Analysis Theory and Practice*, Saunders College, Fort Worth, TX, USA, 1995.
14. M. B. Priestley, *Spectral Analysis and Time Series, Vol. 1*. Academic Press, London, 1981.
15. S. L. Marple, Jr., *Digital Spectral Analysis With Applications*, Prentice Hall, Englewood Cliffs, NJ, USA, 1987.

9. Appendix--Raw Data from Analysis of Project Database

Table 6: Results of Analyzing Data from Phase I of Project Database

Filename (*.raw)	Number of FFT's per File	Number Correctly Identified	Number of Correctly Identified
acg_1_+C4~	18	18	18
acg_1_+D4~	16	16	16
acg_1_+E5~	7	7	7
acg_1_-C4~	18	18	18
acg_1_-D4~	16	16	16
acg_1_-E5~	8	8	8
acg_1_~C4~	20	20	7
acg_1_~D4~	24	24	22
acg_1_~E5~	7	7	6
bag_1_+A2~	28	27	1
bag_1_+A3~	22	22	22
bag_1_+C3~	29	29	26
bag_1_+C4~	28	28	28
bag_1_+E2~	31	19	19
bag_1_-A2~	30	30	30
bag_1_-A3~	22	22	22
bag_1_-C3~	27	0	0
bag_1_-C4~	20	20	0
bag_1_-E2~	28	0	0
bag_1_~A2~	31	30	3
bag_1_~A3~	26	26	18
bag_1_~C3"	32	30	0
bag_1_~C4~	21	21	0
bag_1_~E2~	26	16	0

Table 6: Results of Analyzing Data from Phase I of Project Database

Filename (*.raw)	Number of FFT's per File	Number Correctly Identified	Number of Correctly Identified
tru_1_+B4-	18	18	4
tru_1_+C4~	19	18	18
tru_1_+C5~	17	17	16
tru_1_+G4~	19	19	19
tru_1_-B4-	18	18	18
tru_1_-C4~	14	5	5
tru_1_-C5~	16	5	5
tru_1_-G4~	19	16	16
tru_1_~B4-	22	22	7
tru_1_~C4~	19	19	0
tru_1_~C5~	18	18	0
tru_1_~G4~	19	19	0

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
acg_2_~C4~_~A4+	**C4**	**A4#**	**m7**
	A4#	**A4**	**P0**
acg_2_~C4~_~A4~	E3	B3	P5
	E3	G4#	M3
	B3	B3	P0
	B3	G4#	M6
	G4#	G4#	P0
acg_2_~C4~_~B4~	B3	A4#	M7
	A4#	A4#	P0

* these files cause a software error

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
acg_2_~C4~_~C4+	C4	C5#	m2
	C5#	C5#	P0
acg_2_~C4~_~C5~	B3	F6#	P5
	F6#	F6#	P0
acg_2_~C4~_~D4+	**C4**	**D4#**	**m3**
	D4#	**D4#**	**P0**
acg_2_~C4~_~D4~	**C4**	**D4**	**M2**
	D4	**D4**	**P0**
acg_2_~C4~_~E4~	**C4**	**E4**	**M3**
	E4	**E4**	**P0**
acg_2_~C4~_~F4+	**C4**	**F4#**	**TT**
	F4#	**F4#**	**P0**
acg_2_~C4~_~F4~	**C4**	**F4**	**P4**
	F4	**F4**	**P0**
acg_2_~C4~_~G4+	**C4**	**G4#**	**m6**
	G4#	**G4#**	**P0**
acg_2_~C4~_~G4~	**C4**	**G4**	**P5**
	G4	**G4**	**P0**
bag_2_~C3~_~A3+	B2	G3	m6
	B2	A3#	M7
	G3	G3	P0
	G3	A3#	m3
	A3#	**A3#**	**P0**

* these files cause a software error

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
bag_2_~C3~_~A3~	A2	B2	M2
	A2	G3	m7
	A2	A2	P0
	B2	B2	P0
	B2	G3	m6
	B2	A3	m7
	G3	G3	P0
	G3	A3	M2
	A3	**A3**	**P0**
bag_2_~C3~_~B3~	B2	G3	m6
	B2	B2	P0
	G3	G3	P0
	G3	B3	M3
	B3	**B3**	**P0**
bag_2_~C3~_~C3+	*		
bag_2_~C3~_~C4~	*		
bag_2_~C3~_~D3+	C3	G3	P5
	C3	A3#	m7
	G3	G3	P0
	G3	A3#	m7
	A3#	A3#	P0
bag_2_~C3~_~D3~	C3	A3	M6
	C3	D4	M2
	A3	A3	P0
	A3	D4	P4
	D4	D4	P0

* these files cause a software error

Table 7: Results of Analyzing Data from Phase II of Project Database

Filename (*.raw)	Low Note	High Note	Interval
bag_2_~C3~_~E3~	F2	E3	M7
	F2	B3	TT
	E3	**E3**	**P0**
	E3	B3	P5
	B3	B3	P0
bag_2_~C3~_~F3+	F2#	C3	TT
	C3	C3	P0
bag_2_~C3~_~F3~	F2	C3	P5
	F2	G3	M2
	C3	C3	P0
	C3	G3	P5
	G3	G3	P0
bag_2_~C3~_~G3+	G2#	C3	M3
	G2#	G2#	P0
	C3	C3	P0
	C3	G3#	m6
	G3#	G3#	P0
bag_2_~C3~_~G3~	G2	C3	P4
	C3	C3	P0

* these files cause a software error