Violin Note Detector Using Digital Signal Processing

A Senior Project

Presented to

The Faculty of the Electrical Engineering

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

Of the Requirements for the Degree

Bachelor of Science

By

Nathaniel Kyle Mopas

December, 2009

TABLE OF CONTENTS

Section *Page*

Acknowledgements.....iv

I. Abstract.....1

II. Introduction.....2

III. Background.....3

IV. Requirements.....6

V. Design.....8

VI. Test Plans…….....15

VII. Integration and Test Results…….....17

VIII. Conclusion...,.....33

IX. Bibliography.....35

Appendices

A. Equipment Costs.....36

B. Schedule- Time Estimates.....37-39

C. Scilab Program Code.....40

D. Dec C++ Program Code.....41-49

E. Analysis of Senior Project Design…….50-55

List of Tables and Figures

Table *Page*

1. Table 1: Notes of first position and their corresponding frequencies.....6

2. Table 2: Equipment needed for project and their costs.....36

3. Table 3: Schedule Time Estimates Spring Quarter 2009…….....37-38

4. Table 4: Schedule Time Estimates Summer 2009…….....38

5. Table 5: Schedule Time Estimates Fall Quarter 2009……38-39

Figures

1. Figure 1: The violin with labeled parts……..……..…..2

2. Figure 2: Notes and corresponding finger positions……..3

3. Figure 3: Scilab simulation note A, G string, freq = 220.0 Hz, r = 0.9999995…...10

4. Figure 4: Scilab simulation note A, G string, freq = 220.0 Hz, r = 0.9999…….11

5. Figure 5: Before filtering, note A on the G string, frequency is 220.0 Hz…… 18

6. Figure 6: Before any filtering, note B on the G string, frequency is 246.94 Hz…..19

7. Figure 7: Note A, G string after passing through the filter designed for note A,G string, pole radius r =0.9999995 and sample frequency = 44100 Hz……..19

8. Figure 8: Note B, G string after passing through the filter designed for note A, G string, pole radius r =0.9999995 and sample frequency = 44100 Hz……..20

9. Figure 9: Note A, G string before filtering sampling frequency = 44100 Hz….....21

10. Figure 10: Note B, G string before filtering sampling frequency = 44100 Hz…..22

11. Figure 11: Note A, G string after passing through the filter designed for note A,G string, pole radius r =0.9999995 and sample frequency = 44100 Hz……..23

12. Figure 12 Note B, G string after passing through the filter designed for note A, G string, pole radius r =0.9999995 and sample frequency = 44100 Hz……..24

13. Figure 13: Note A, G string before filtering with r =0.99995 and Sampling frequency = 11 kHz……...……...…25

14. Figure 14: Audacity recorded file of note A, G string, Sampling frequency = 11 kHz……...…25

15. Figure 15: Note A, output of difference equation ……..…26

16. Figure 16: Note A, absolute value output difference equation……...…27

17. Figure 17: State Machine of Note A with debounce problem……...…28

18. Figure 18: State Machine of Note A, after debounce problem was solved……...29

19. Figure 19: Display Screen showing the notes recognized, and their respective times…….30

20. Figure 20: Audacity file of recording of notes B,C,B on the G string……...31

21. Figure 21: Display Screen showing one A recognized, but the other notes were not……31

22. Figure 22: Audacity file of recording of notes four A's on the G string...……..…32

Acknowledgements

I would like to thank my senior project advisor Dr. DePiero for all his help, advice, and patience during the course of this senior project. Additionally I would like to thank my family, friends, and roommates for their support and encouragement.

Abstract

The objective of my senior project was to create a violin note detector using digital signal processing methods learned from EE 419/459. At the core foundation of my approach was designing sixteen band-pass filters, each designed for a specific note on the violin. The range of notes chosen was for the first position of the violin, which limits the selection to the first four notes on each string. The steps in the process to completion included designing difference equations of filters, simulating filters, recording music notes, and using software development environment from EE 419/459 to program filters using information gained from simulations.

After programming, the output of the filters were tested in three different methods to observe if the band-pass filters were working in identifying the correct note they were designed for. Two methods were sending the digital output WAV file through spectrum analyzer and spectrogram, both of which were downloaded along with the software development environment. A third method was programming a few lines of code that was able to send all the information regarding the number of samples, input, and output of the difference equations to Excel in table form to view exactly how the filters were working. The results on the output showed that the filters were able to recognize the note if one note was played only, and adding more notes created complications with note recognition. From these results, I concluded that my filter design contains very sensitive parameters and with some note frequencies being closer together, the filters have more difficulty recognizing the note.

Introduction

The goal of my senior project was to create a note recognizer for the violin. I arrived at my idea for a senior project by taking the advice from Dr. Cirovic during EE 460: to combine electrical engineering with something I love, which was playing the violin. Electrical engineering students and professors suggested using a Digital Signal Processing method because I had recently learned about musical applications in EE 419/459. Upon multiple meetings with Dr. Depiero to figure a way to use DSP, we arrived at pitch recognizer, and at its greatest ambition would work in real time by using the DSK board and software from EE 459 lab. However, because of numerous difficulties in using recorded pitches combined with many parameters that had design tradeoffs with one another, we minimized the scope of the project and brought its focus to working at recognizing recorded violin notes. Shown below is a picture of a violin and the labeling of its parts.

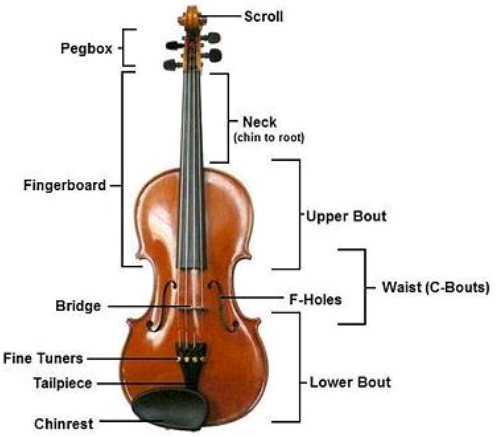


Figure 1: The violin with labeled parts

Background

Both the violin and other musical instruments have frequencies closer together on the low end and frequencies getting further apart as the note moves up in frequency. Even though it may have been easier to pick random notes with greater differences in frequencies thus making the band-pass filter design easier, the choice still remained to focus on the first position of the violin. This was because the project was kept in mind as potential aid to individuals starting to play the instrument who have trouble with correct finger placements corresponding to specific notes. Therefore the direction was chosen to the first position because it is the fundamental position for violinists where initial learning occurs and many musical pieces contain their notes. Below are the notes and finger positions of the first position on the violin.

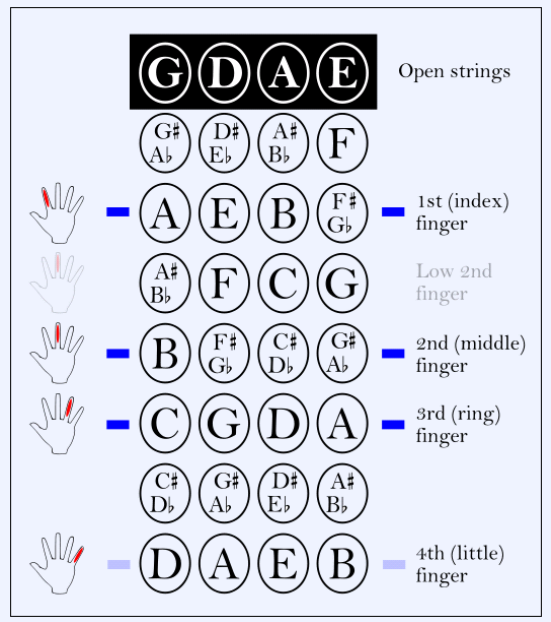


Figure 2: Notes and corresponding finger positions

A strong component to this project was using DSP software development environment. This environment was a free downloadable program from Dr DePiero's website and became very helpful because of its capabilities which included WAV file Input/Output, Forward and Reverse FFTs, random signal generation, BMP image Input/Output (color and grayscale) finding eigenvalues and eigenvectors, matrix arithmetic, and solving simultaneous equations. However from this environment I used the WAV project capabilities which functions as a series of dragging and dropping of input WAV files over the executable file of the program code which generates digital filter output. The code is created in C language and utilizes a starter program created by Dr. DePiero which simulates an A/D converter on the input WAV file, reads in the samples one at a time, simulates D/A converter, and generates an output WAV file. After this area of code, Dr. DePiero leaves the student to edit the main function of the code which is defaulted to read in WAV files and send them right back out without any filtering being done. In this portion of code the student can program the difference equations of their filters with their respective outputs being sent to the output WAV file. Other portions of the code that are already in the start up code create the black pop up screen that prompts the user for "<cr>" thus waiting for the user to confirm by pressing any key on the key board before generating the output WAV file.

This output WAV file is important because it is the result of the filtering and the user can choose what method to view the output in order to observe how their filters are working. Included within the software development environment are the

spectrum analyzer simulator and the signal image processing tool. Both of these allow again dragging and dropping of WAV files to observe respectively the magnitude versus frequency plot, and the spectral density versus time plot. Additionally for programming, Dr. DePiero's website contained several lines of code which were inserted in the beginning and end of the main program edited by the student. These specific lines of code allow generation of the data tables in Excel hence allowing exact plots based on the values from the difference equations.

Requirements

Table 1: NOTES OF FIRST POSITION AND THEIR CORRESPONDING FREQUENCIES

G string notes	
Note	Frequency (Hz)
A	220
B	246.94
C	261.63
D	293.66

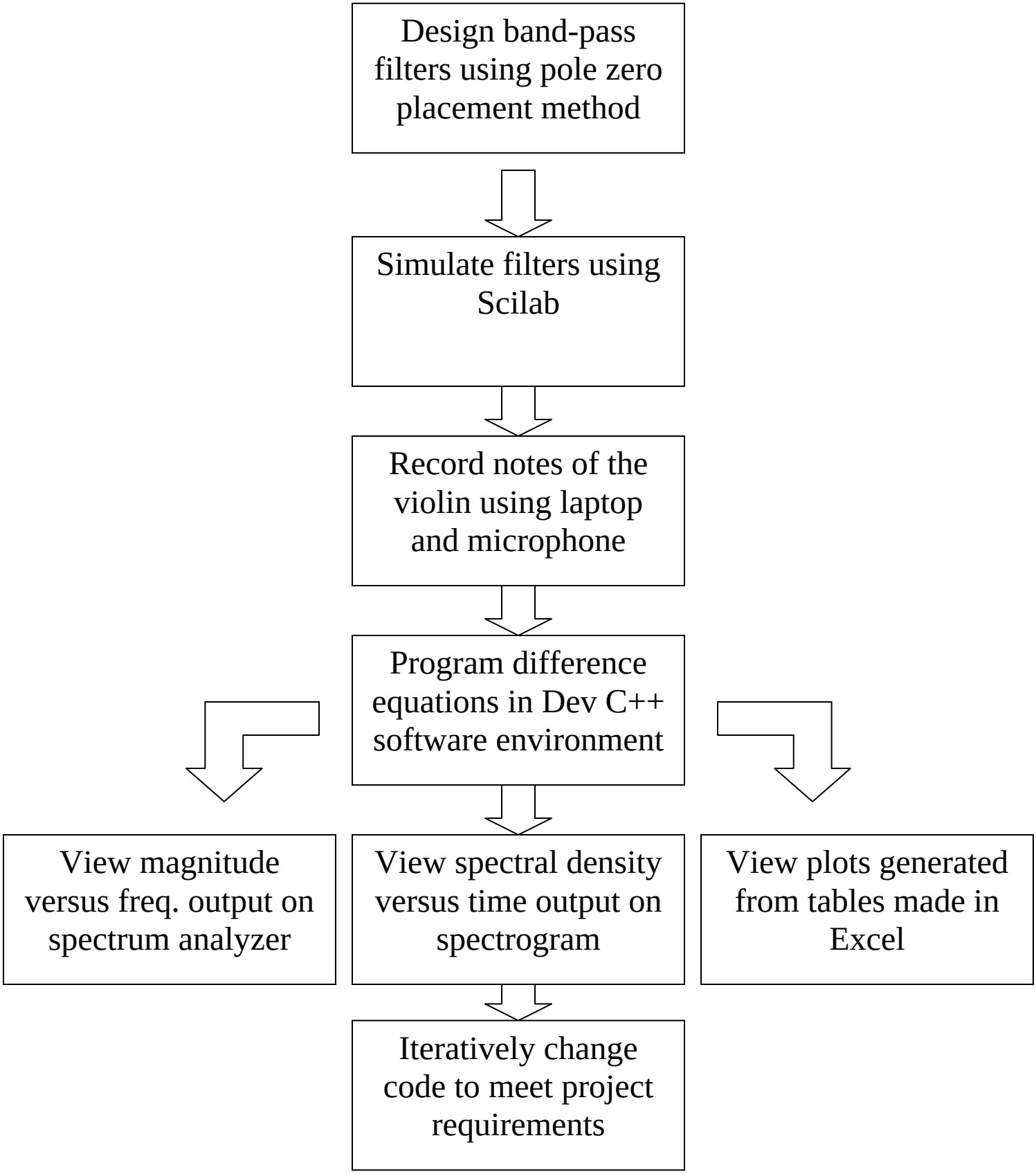
D string notes	
Note	Frequency (Hz)
E	329.63
F#	369.99
G	392
A	440

A string notes	
Note	Frequency (Hz)
B	493.88
C#	554.37
D	587.33
E	659.26

E string notes	
Note	Frequency (Hz)
F#	739.99
G#	830.6
A	880
B	987.77

From viewing the table above, the notes of the lower frequencies are in general closer to one another than the notes at the higher frequencies. A further requirement of these band-pass filters were that they all needed to be extremely precise in filtering all other frequencies but the note they are designed to recognize. By performing in this manner, the band-pass filter would not filter out both the note that it is designed for and a closer note thus creating confusion when recognizing what note it actually being played. The method needed to make this happen was the pole zero placement method for creating difference equations of filters, and this will be described in the following design section.

Design


Design band-pass filters using pole zero placement method
Simulate filters using Scilab
Record notes of the violin using laptop and microphone
Program difference equations in Dev C++ software environment
View magnitude versus freq. output on spectrum analyzer
View spectral density versus time output on spectrogram
View plots generated from tables made in Excel
Iteratively change code to meet project requirements

Pole Zero Placement Method

As previously mentioned the core design of this project was the design of the sixteen band-pass filters. The closeness in frequencies needed to create difference equations of band-pass filters with high Q value. The method of design was chosen as "pole zero placement" method from EE 419/459 class. This method basically has the user design the poles based on the sampling frequency and the frequency of the note, and the user chooses the pole radius. The pole radius has a proportional effect on the Q value, with radii more precisely closer to 1 causing a higher Q value, and the radii less precise causing a lower Q value. The difference equation generated from this design method is shown below, although more in depth information about this method can be viewed through Dr. DePiero's website listed in my bibliography.

$$\mathrm{H}(z) = Ko * \frac{1}{(1 - re^{+j2\pi Fo} z^{-1})(1 - re^{-j2\pi Fo} z^{-1})}$$

When performing the mathematical process of foiling the equation becomes analogous to-

$$H(z) = \frac{B(z)}{A(z)} = \frac{B_o + B_1 z^{-1} + B_2 z^{-2} + B_3 z^{-3}}{1 + A_1 z^{-1} + A_2 z^{-2} + A_3 z^{-3}}$$

Scilab bandpass filter simulations

This is a convenient form because the next step in the design was simulating the filters in Scilab. A few simple lines of code shown in the appendices describe how to use the A and B coefficients of the denominator and the numerator, along with the pole radius, sample frequency, and number of samples can generate a simulation of the band-pass filter. Additionally the Scilab simulations helped show how the pole radius previously chosen in the pole zero placement method directly affects the magnitude versus frequency plot. As shown below when the pole radius more precisely closer to one (example 0.9999999… etc) the magnitude response focuses more on filtering everything but the note frequency it is designed for, however the height of the magnitude is very low. Here is an example of note A of the G string with note frequency = 220.0 Hz and the following parameters

Pole radius = r = 0.9999995 Sampling frequency = S = 44100

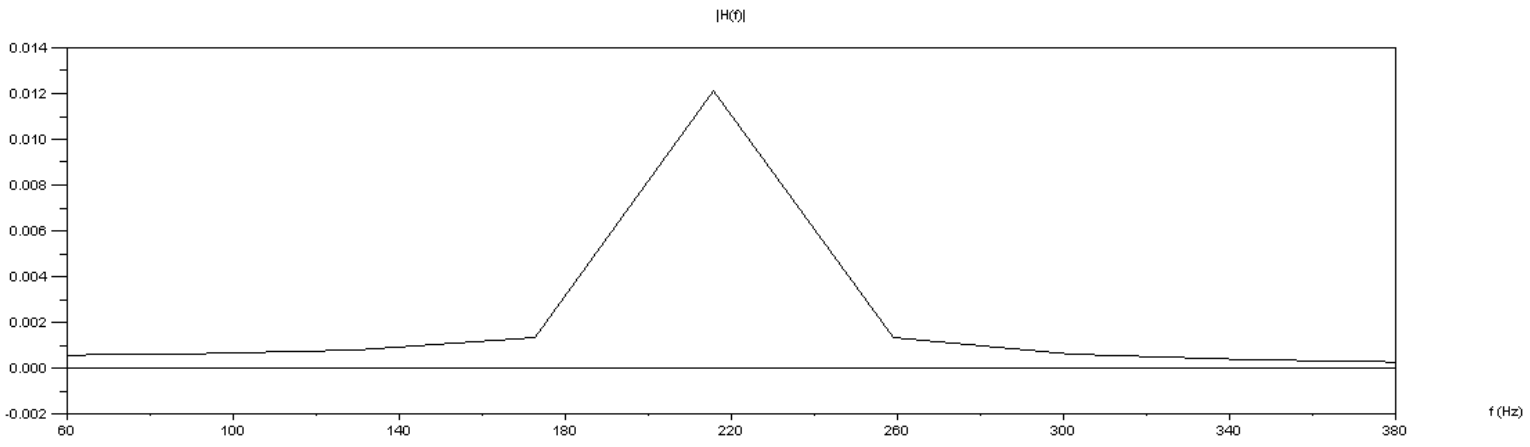


Figure 3: Scilab simulation note A, G string, freq = 220.0 Hz, r = 0.9999995

In this plot, it is observed that the band-pass filter is centered at the desired frequency of 220.0 Hz, however the strength of the magnitude is very low, peaking at approximately 0.012. The next note in line is the note B on the G string at a frequency of 246.94 Hz and although the magnitude at this frequency is not cutoff by the band-pass filter, the magnitude is very low, approximately 0.006. Similarly, when the pole radius is made less precise, for example setting it at r = 0.9999 while all other parameters are kept the same the plot becomes much different as shown below.

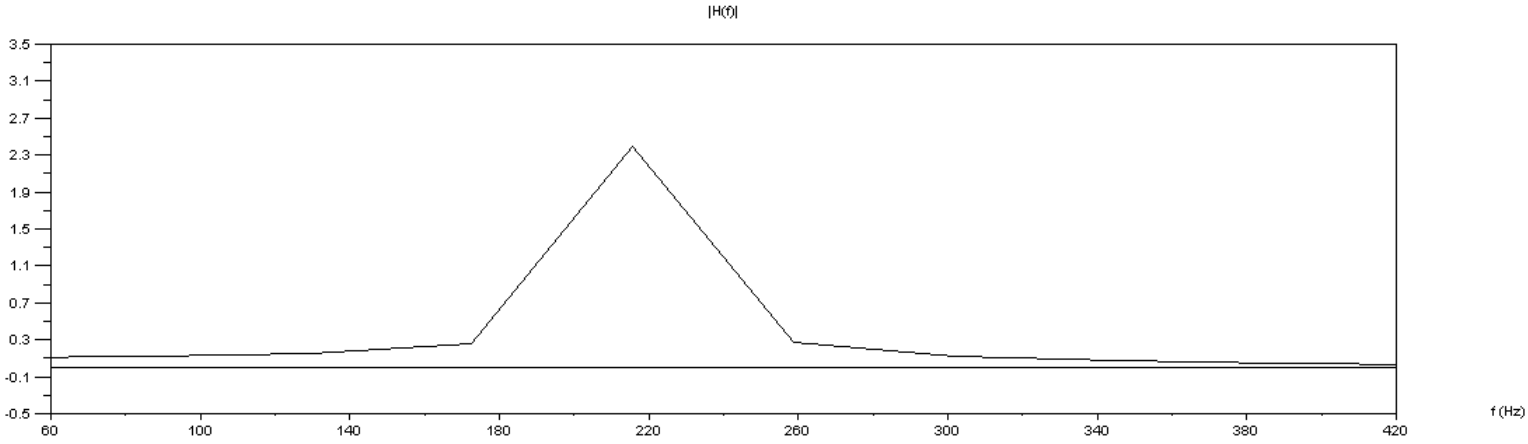


Figure 4: Scilab simulation with note A, G string, freq = 220.0 Hz, r = 0.9999

As before it is observed that the band-pass filter is centered at 220 Hz, but the magnitude at this frequency is much greater at approximately 2.4, while the next note B at frequency of 246.94 Hz has magnitude of approximately 1.2. From these two examples there a design trade-off can be observed between the pole radius and the respective magnitude versus frequency response. When the pole radius is set much closer to 1 for example at 0.9999999, the band-pass filter is more precise at filtering

all other frequencies but the note that it is designed for, but the height of the magnitude is very low thus affecting if the output WAV file can even be heard. On the other hand, when the pole radius is set less precise than 1 (in the example here r = 0.9999) the magnitude at the center frequency is approximately 200 times greater (2.4 versus 0.012. This affects the output WAV file again with not only the desired center frequency's note definitely being heard but the next notes in sequence also being heard hence showing that the design of filter is not performing as required.

Recording notes

The program I used to record was Audacity and Sound Recorder Accessory. Audacity provided recording and editing of recorded notes while Sound Recorder Accessory allowed me to change the sampling frequency and save as a new WAV file. Examples of Audacity can be seen in Figure 20 with three notes and Figure 22 showing four notes. In this step of the design process it became of greatest importance to optimize recording of the violin notes because my filters were designed around each note specifically. To accomplish this, I needed to purchase a tuner to tune my violin each time I recorded. Tuning the violin is a process of adjusting the pegbox and fine tuners shown in Figure 1. In order place the ideal recording settings, I used the practice rooms in the music building because they were isolated and removed background noises. When actually recording the note, I chose to pluck the strings because it was easier to achieve crisp and precise music output instead of bowing which had more potential to mess up. Looking at Figure 2, there existed possibilities of accidently bowing the wrong note at the same time as the right note by

bowing multiple strings at once. It would prove difficult to bow the exact way for each note, each time with regards to how much rosin is put on the bow, the angle of the bow and which part of the bow was being used (top versus middle versus bottom). Plucking the note removed these additional variables that would need to be considered.

Programming Equations in Dev C++ Environment

As described earlier in the background, this portion of the design took up the bulk of the work because this was the final step before testing the output of the filters. One issue to consider prior to programming the difference equation was the fact that my difference equation was IIR (infinite impulse response) while the code in the starter program was FIR (finite impulse response).

My difference equation after pole zero placement and Scilab was-

$$H(z) = \frac{B(z)}{A(z)} = \frac{B_o + B_1 z^{-1} + B_2 z^{-2} + B_3 z^{-3}}{1 + A_1 z^{-1} + A_2 z^{-2} + A_3 z^{-3}}$$

This easily translated to the IIR difference equation-

$$y(n) = B_0 x(n) + B_1 x(n-1) + ... + B_M x(n-M) - A_1 y(n-1) - A_N y(n-N)$$

However the starter code had the FIR difference equation-

$$y(n) = B_o x(n) + B_1 x(n-1) + ... + B_M x(n-M)$$

The IIR could become programmed using series of if statements connecting the output with the previous output as viewed in the code of the appendix.

Another portion of the program code that needed to be designed for was a method of measuring the strength of the output. An integration of the output value

over a number of samples would yield a value of zero because the output sinusoid fluctuated equally between positive and negative values due to the nature of the vibrations of the plucked note. Thus the focus was put to creating a new difference equation-

out2 = (0.01 * fabs(out)) + (0.99*out2prev1);

out2prev1 = out2; (out2prev1 initialized to 0 in beginning)

This programmed equation focuses on using the absolute value of the output combined with its previous value as a measure of the strength of the output signal. From this difference equation the user can view in table form what value of out2 fluctuates around thus creating a threshold for a state machine. The state machine is then designed to become active high on the rising edge of the out2 (when out2 first rises above the threshold) and active low on the falling edge (when out2 falls below the threshold) hence signaling when the note is recognized by the band-pass filter.

The final step of the block diagram for design was the iterative changing of code and observing the affects it has out the output. This needs to be factored into the design and work process because this project has iterative steps within each step of the design such as changing parameters of Scilab code and viewing its effects on the magnitude response, or re-recording notes because the violin may become slightly out of tune.

Test Plans

From the description in the background section the three methods of viewing the output were briefly described. The test plan was to observe the output on the spectrum analyzer, spectrogram, and excel plot. Concerning the spectrum analyzer, similar to Scilab this plot allows the user to observe how strong the magnitude is at a specific frequency. Another thing to keep in mind was not only the magnitude of the note at the frequency the filter was designed for, but the magnitude of the notes below and above note designed by the band-pass filter.

When looking at the spectrogram generated by the Signal and Image Processing tool, one thing to keep in mind is the brightness of the peaks at specific frequencies. With both the spectrum analyzer and the Signal Image Processing Tool, the user can see what note is being recognized and the extent of how the nearest notes are filtered out. What makes the spectrogram special was that the user could view all frequencies excited by the input and output WAV files and further determine what note is played by observing the harmonics (multiples of the fundamental frequency).

In Excel, the data of the number of samples and values of the input and output difference equations can be seen in table form, thus allowing generating of the plots to observe exactly what the input and output waveforms look like based on the musical input and output. From this output observations, the thing to look out for is strength of the input and output based on how strongly the note was plucked and how this influences length of the note (the number of samples the note is sustained for).

Although the objective of the project was designing band-pass filters to recognize one note, another step of the test plan was to move onto more difficult cases of recording notes to see if the bandpass filters can pick up the note they are designed to recognize. The difficult cases included recording one note multiple times and recording alternating notes.

Integration and Test Results

In the original design, the A and B coefficients were generated by the combination of the pole zero placement filter design method and the Scilab simulations. The connection of this step was hand calculating the A and B coefficients using pole zero placement method and inputting these values in the freqz_fwd function of Scilab. Since this project involved an iterative process of changing parameters and seeing how it affects the output, to continue doing hand calculations to get A and B coefficients every time the sampling frequency and/or the pole radius was changed would be tiresome and inefficient. The integrating step here was to have Scilab calculate the coefficients based on simplifying the difference equation to a step where each coefficient would correspond to a portion of mathematical calculations.

Similarly a method of integrating the Scilab simulation program code and the Dev C++ program code, was working with the variable “fs_hz”. This was set in the starter code already written by Dr. DePiero as the sampling frequency. Using this variable allowed the generation of the A and B coefficients with the same mathematical calculations within the program code itself of Dev C++ without having to jump two sets of code. This did not completely skip the Scilab step because whenever there was the need to go back and change the pole radius and sample frequency, the simulation of the performance of the filter could quickly be generated.

Spectrum Analyzer output results

Moving onto the output results, the first thing to observe was how changing of the parameter of pole radius affected the spectrum analyzer's reading on the bandpass filter's output. The following diagrams below show the spectrum analyzer's reading on the plucked notes of A and B on the G string, and then what happens to each after passing through the filter designed for note A.

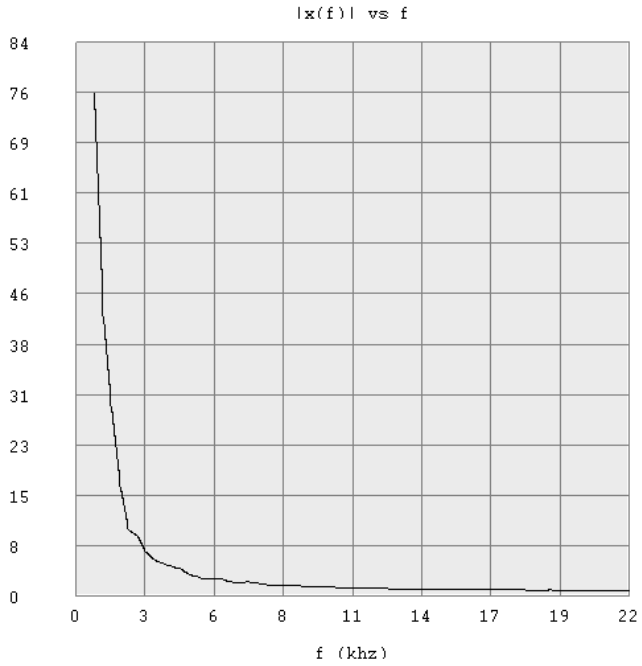


Figure 5: Before any filtering, note A on the G string, frequency is 220.0 Hz

In this figure although the horizontal frequency scale is in kHz, highest value occurs approximately where f = 220.0Hz. This note recognition is further enforced by the weaker magnitude strengths of the notes in sequence after note A on the G string. For the next note B on the G string at frequency =246.94 Hz, a similar graph can be observed as at the top of the next page.

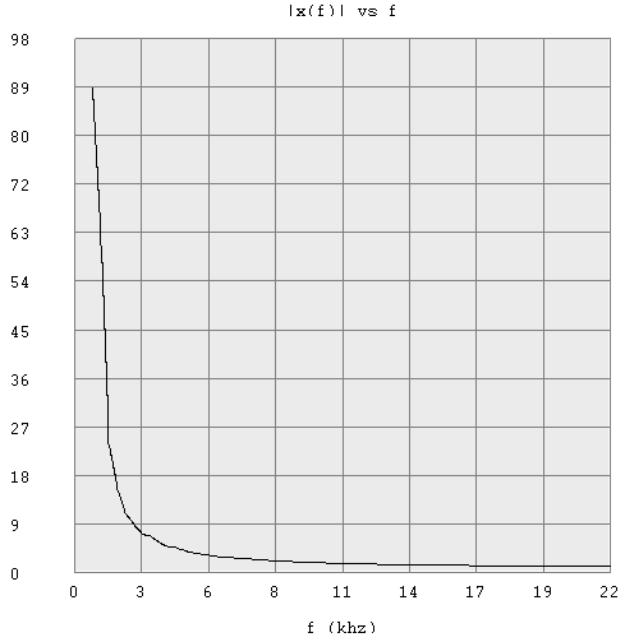


Figure 6: Before any filtering, note B on the G string, frequency is 246.94 Hz

This plot is hard to determine because of the frequency scale again, but like the note A before it, the next notes in sequence have less magnitude strength.

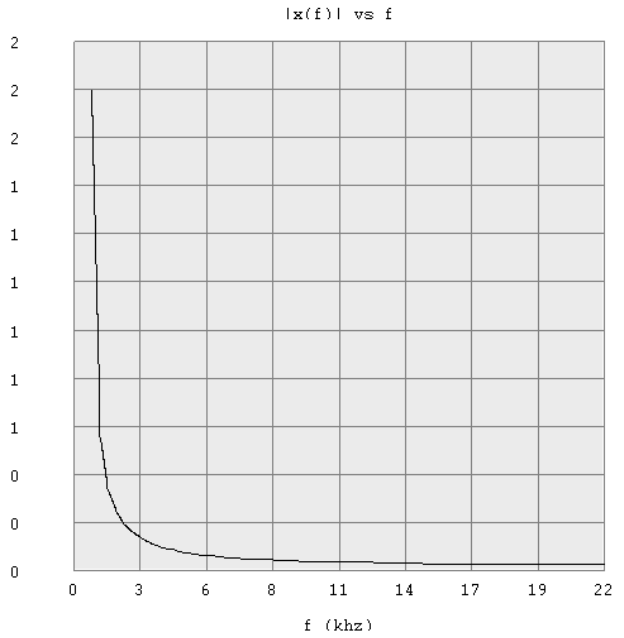


Figure 7: Note A, G string after passing through the filter designed for note A,G string, pole radius r =0.9999995 and sample frequency = 44100 Hz

In this plot, the magnitude is much lower than the note A before the filtering and this was anticipated in the Scilab simulation magnitude frequency response. This occurred because the pole radius was extremely close to 1, thus being very small. As a result the frequency was recognized, but not much of the original signal was allowed to pass through. Notice magnitude height here is much greater than Scilab Figure 3, while changing the pole radius on Scilab Figure 4 brings the height closer to this value of Figure 7.

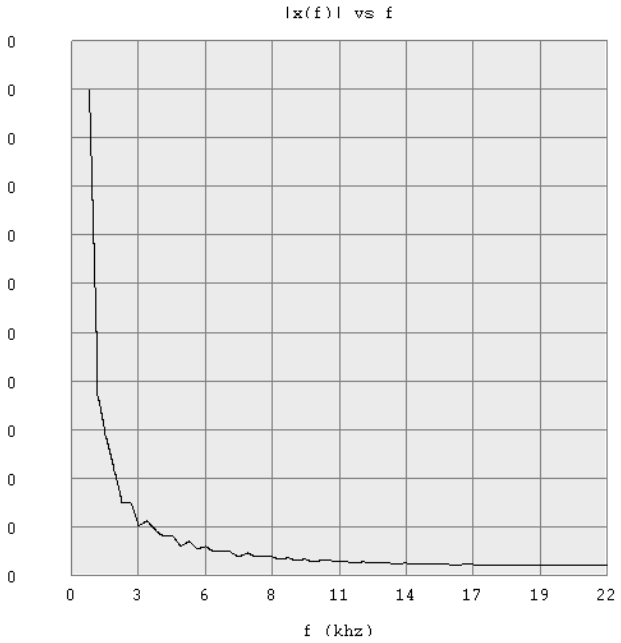


Figure 8: Note B, G string after passing through the filter designed for note A, G string, pole radius r =0.9999995 and sample frequency = 44100 Hz

This note is nearly filtered out as can be observed by the magnitude strength at the frequency of the note B 246.94Hz. The zeroes do not mean the magnitude is zero

everywhere, but can be interpreted as decimal value less than 1. From the spectrum analyzer, only a rough idea of how effective the filters are can be seen because of the frequency scale being in kHz. The scale makes it hard to determine the frequencies on the lower end since as seen from Table 1, all the frequencies of the notes that the filters are designed for are less than 1000 Hz.

Signal and Image Processing Tool- Spectrogram

As a result of the spectrum analyzer being difficult to interpret, the next tool used was observing the spectrogram of the Signal and Image Processing tool. Below is the spectrogram of note A of the G string.

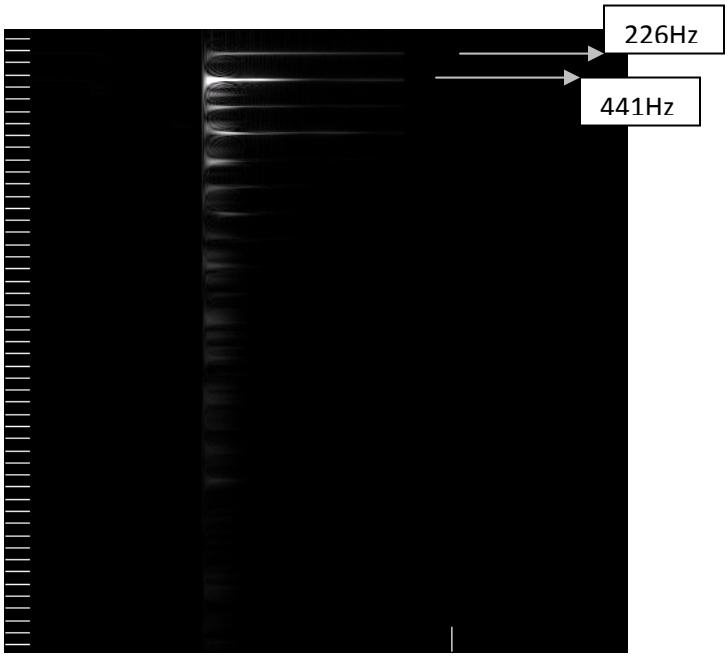


Figure 9: Note A, G string before filtering sampling frequency = 44100 Hz

In this figure the brightest spikes occurred at the fundamental frequency close to 220Hz, and at the next highest harmonic approximately 440Hz. The explanation of

the harmonic being much brighter than the fundamental frequency is because the sampling frequency is 44100 Hz which may have excited the 440Hz.

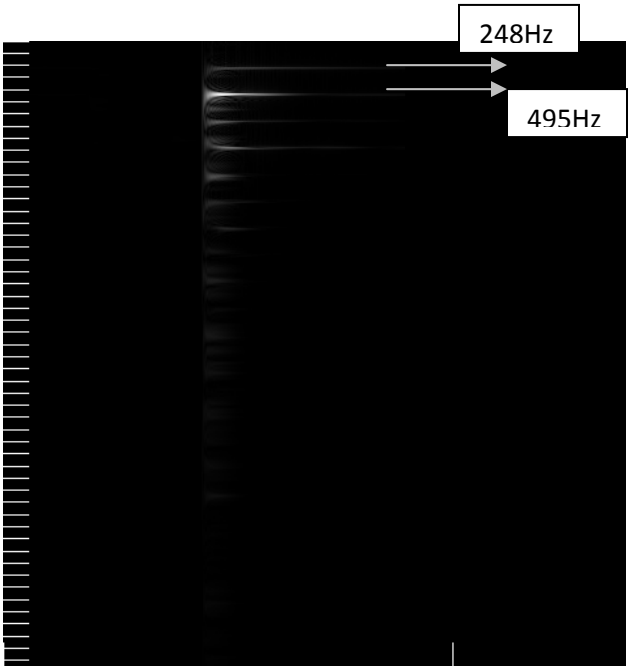


Figure 10: Note B, G string before filtering sampling frequency = 44100 Hz

Similarly, in this figure the brightest spikes occurring at the fundamental frequency close to 246.94Hz, and at the next highest harmonic close to 493.88Hz. The explanation of the harmonic being slightly brighter than the fundamental frequency is because the sampling frequency is 44100 Hz which is the approximate multiple of 440Hz as before being close to 493.88Hz.

Figure 11: Note A, G string after passing through the filter designed for note A,G string, pole radius r =0.9999995 and sample frequency = 44100 Hz

In this spectrogram, the only spike occurs at approximately 226Hz, which clearly shows that the band-pass filter designed for note A filters out everything around the note it is designed for. When comparing Figure 11 brightness to Figure 12, we can see the analogous relationship to Figure 7 and Figure 8 because Figure 7 had a height of 2 and a fraction, while Figure 8 had a decimal value below 1.

Figure 12 Note B, G string after passing through the filter designed for note A, G string, pole radius r =0.9999995 and sample frequency = 44100 Hz

For this spectrogram, note A frequency is barely recognized which was also seen in Figure 8 where the magnitude is a decimal value below one. This shows that the filter designed for note A filters out the notes around it and performs as required.

Excel output, input, and state machine plots

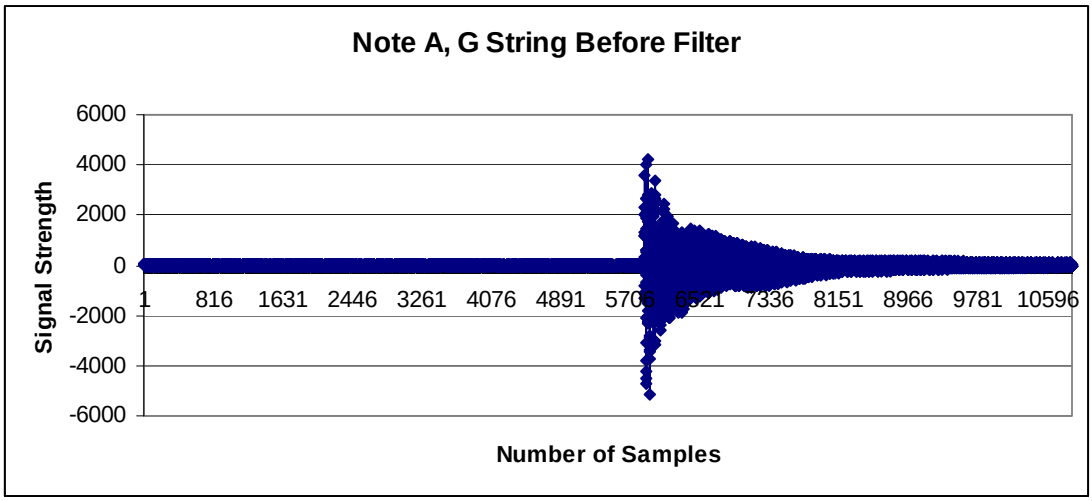


Figure 13: Note A, G string before filtering with r =0.99995 and Sampling frequency = 11kHz

When comparing this to the audacity recorded file, Figure 14, we can see that the occurrence of the plucked note is around the same time.

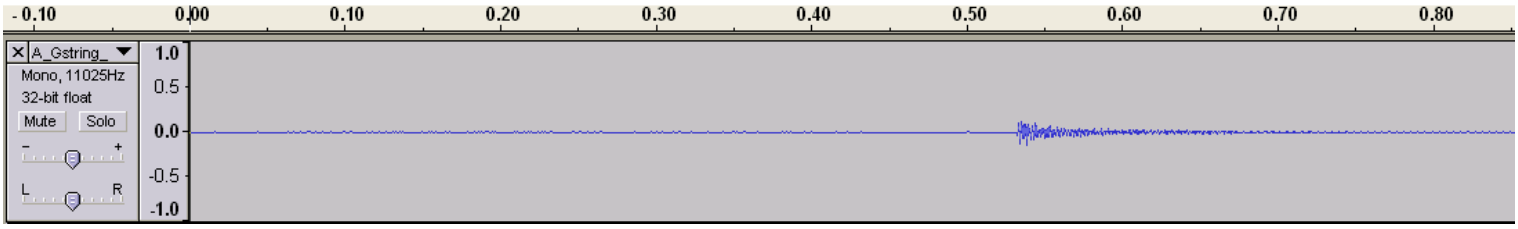


Figure 14: Audacity recorded file of note A, G string, Sampling frequency = 11 kHz

This can be confirmed by a approximating calculation using Excel number of samples

$$\frac{5800 samples}{11000 \frac{samples}{\sec}} = 0.527 \sec$$

When viewing the output difference equation values, they were both positive and negative. This occurred because the input consisted of plucked note vibrations that fluctuated both above and below the desired frequency just due to the nature of the vibrations. This can be easily observed in the sinusoidal output shown below

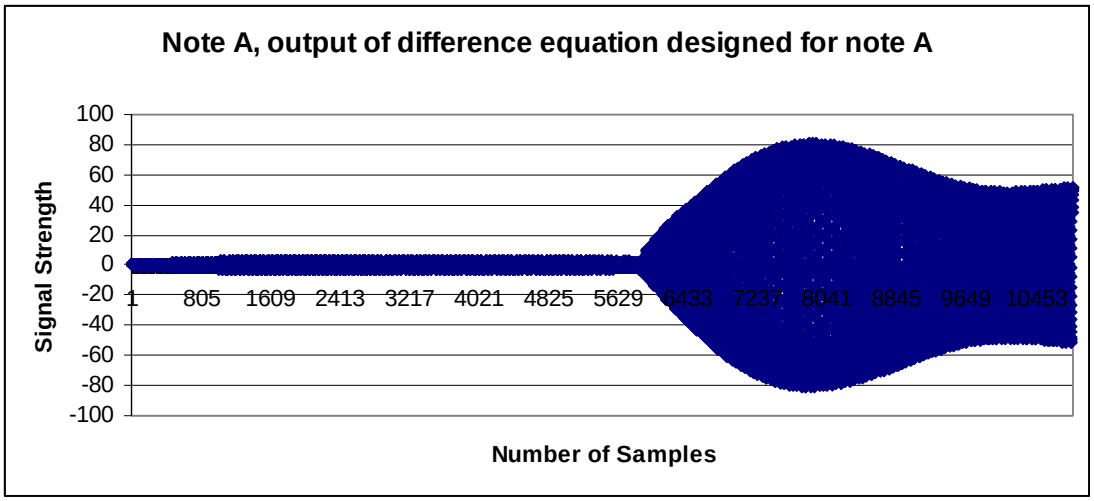


Figure 15: Note A, output of difference equation

In this figure above, it can be observed that the plot is a concentrated overlapping of sinusoids thus giving it the shape it has. To view this easily, just scroll up and down of this word document. The next figure has the difference equation containing the absolute value of the output plus its previous value as described in the background section.

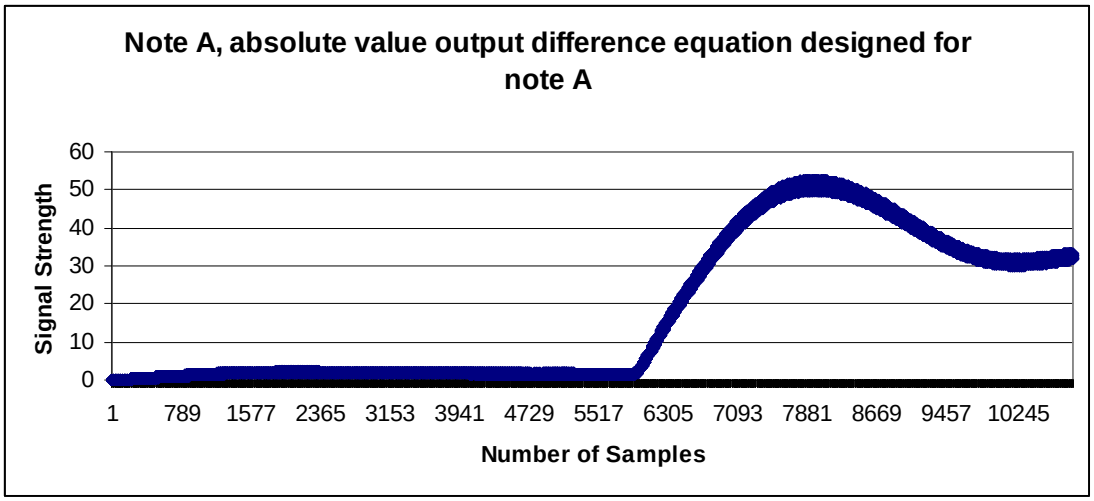


Figure 16: Note A, absolute value output difference equation

Compared to the figure shown at the bottom of the previous page, it can be seen that Figure 16 is the just the positive portion of the figure 15. One note to make here was that in both signals the output waveforms of the plots do not relax the signal strength back to zero. This occurred by way of recording since I just recorded one note, waited briefly and stopped the recording. Had I plucked, and waited longer, the signals may have been able to relax.

At this point in the project, a difficult problem was encountered. As already mentioned, the output of the signals fluctuated around a certain value that established the threshold, but because of the frequency of the note constantly fluctuated around its fundamental the problem of debouncing occurred. The input signal vibrations caused this and since this was not a clean and precise musical output (analogous to a piano or keyboard note), the band-pass filters recognized the note multiple times, despite the note being plucked once. Recognition was observed through the state machine plots which switched between active high and active low, as shown below.

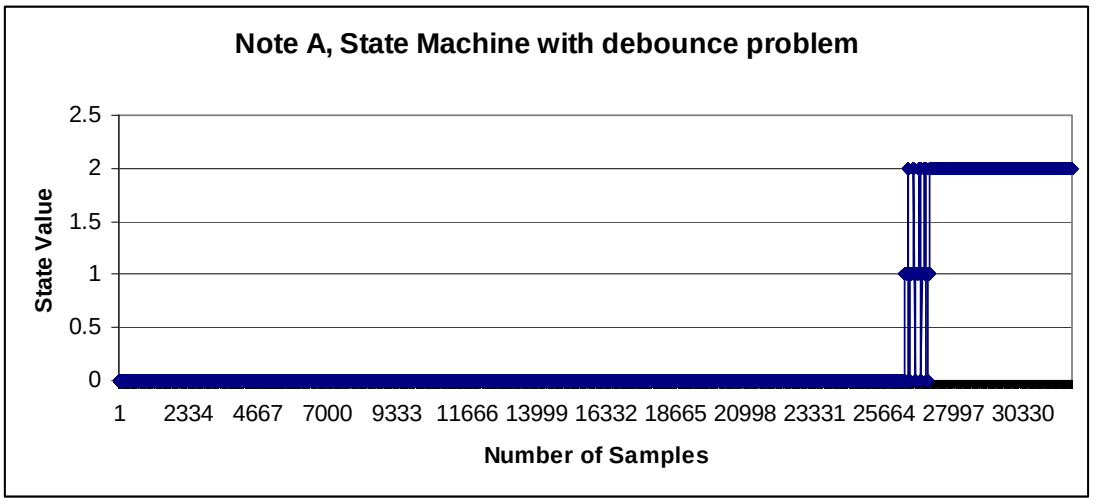


Figure 17: State Machine of Note A with debounce problem

The settings for this example were changed to pole radius r =09999995 and sampling frequency of 44100Hz because the debounce happened more frequently on the higher sampling rates since there was much greater number of samples generated.

The solution to this problem was creating another state machine that immediately followed the one already created for recognizing the note on the rising edge of the threshold value. For this state machine, there existed a low state zero, mid state one, and high state two. The state is initialized at zero, and when the rising edge of the threshold value occurs, a count is initialized at zero, and the state is moved to one. While in state one, the counter is incremented to a value chosen by the user to bypass the range of number of samples when the note was recognized multiple times. State two is only entered upon three conditions- when this counter expires by reaching the set max value, the state value is one, and the output difference equation

generating a value above threshold value. This state machine can be observed in the program code located in the appendix. Below is after debounce problem was solved.

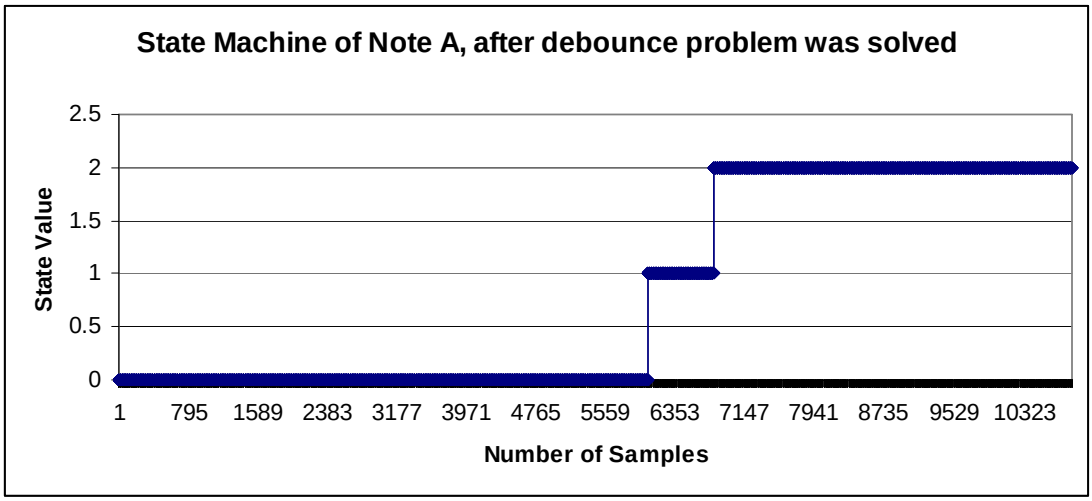


Figure 18: State Machine of Note A, after debounce problem was solved

Another method that was used in combination with the Excel Plot observations was the black pop up screen that occurs to prompt the user for "<cr>" before sending the output. This screen pops up from the code already in the starter program by my advisor. The screen displays the loaded WAV file name and location, created WAV file name and location, and both of their respective sample rates, number of samples, bits per sample and number of channels. The code of bandpass filter program itself displays the printf statement confirming that the Excel plots were generated, and name of the note recognized, and the time it was recognized. These boxes can be seen in both Figure 19 and Figure 21.

When the band-pass filters worked for one note at a time, the project then moved into checking if the notes could be recognized when more than one note was

played. The cases used here were one note being plucked multiple times, and alternating between two notes to see if both notes could be recognized. Unfortunately in both cases problems occurred that differed from the debounce problem.

In the case of the notes B, C, B being played on the G string under sample frequency of 11000Hz and pole radii r = 0.99999, the first and last B was recognized, but the first B may have excited the band-pass filter of note C by displaying note C happening at nearly the exact same time. The reason for this could have happened from the fact that the two notes are close in frequency thus plucking the note B on the G string excited the next note in sequence C. Since C was recognized early in the recording when really it occurred towards the middle, the band-pass filter did not have enough time to relax before recognizing the actual C being played, hence the state remained high for note C. Note B was recognized twice because there was a much greater time difference between the two. This is shown below.

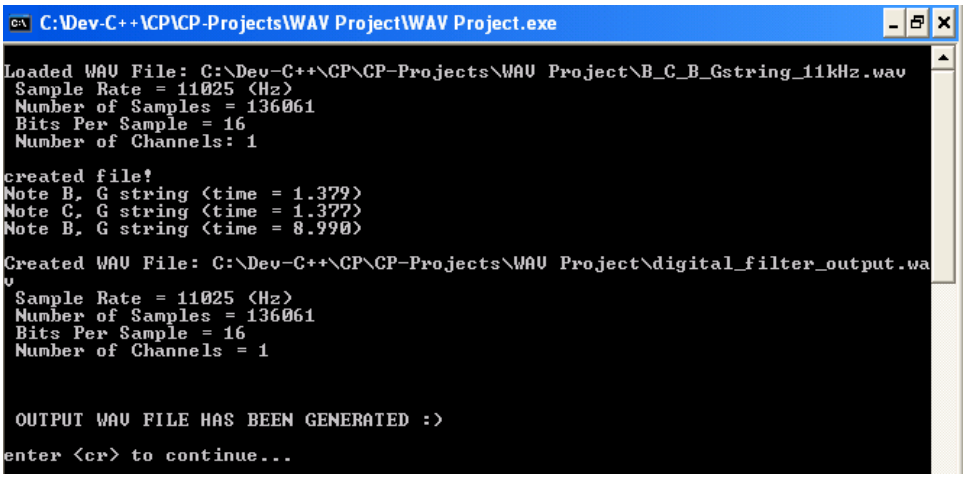


Figure 19: Display Screen showing the notes recognized, and their respective times

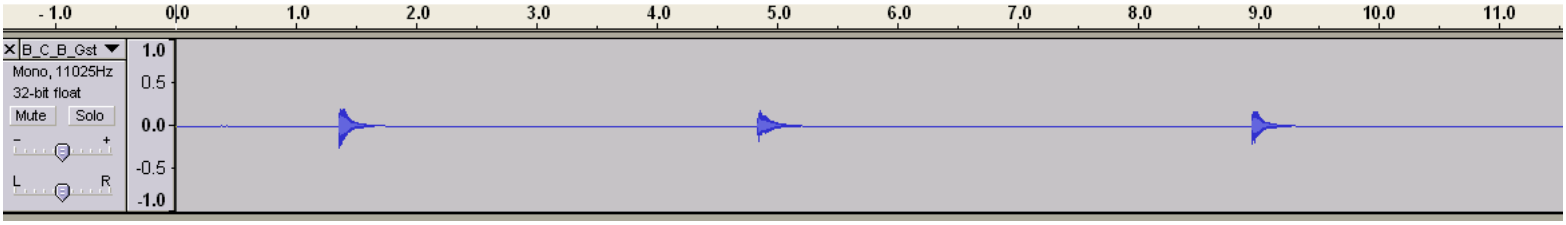


Figure 20: Audacity file of recording of notes B,C,B on the G string.

Both figures above show when the note was played and this was correct for the starting and ending note B as already explained but is incorrect for recognizing note C too early. A possible solution here would be to wait even longer between plucking notes and see what happens. The next case used was playing multiple A notes four times in a row and shown below are the display screen and the audacity file.

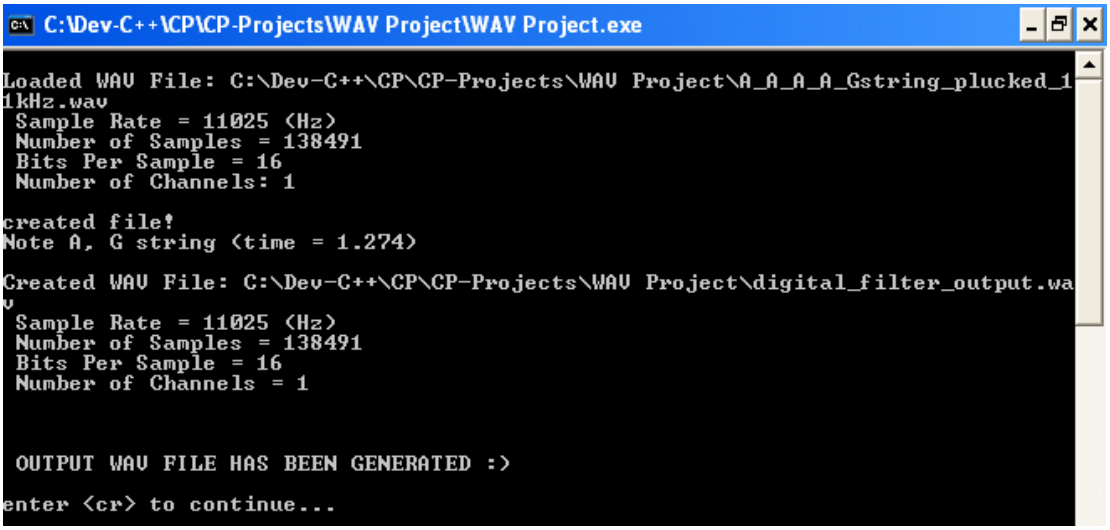


Figure 21: Display Screen showing one A recognized, but the other notes were not

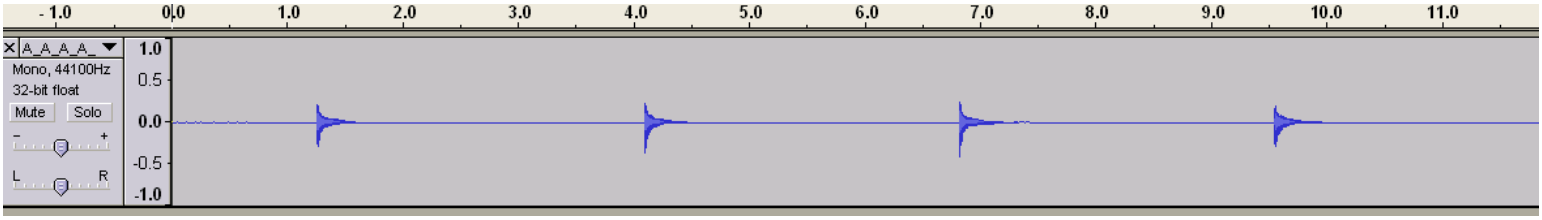


Figure 22: Audacity file of recording of notes four As on the G string

Here the first note was recognized but the remaining ones were not. A possible reason for this occurring was because as discussed before, there was not enough time between each note being plucked and the band-pass filter did not have enough time to relax before recognizing the next note.

Conclusion

From performing this project I learned that there were many parameters to consider when designing a project and many parameters have design trade-offs with one another. An example of this was the Scilab simulations where the sampling frequency and the pole radius were related. If the sampling frequency was large for example 44100 Hz, the pole radius needed to be more precisely closer to one, where I used 0.9999995. This was because there were more samples generated and greater chances of the debounce problem to happen within the larger range of samples. This was observed in Figure 17 with 44100 Hz sampling frequency, and not observed in Figure 18 with the 11000 Hz sampling frequency. Thus the more precise pole radius further narrowed down the frequency recognized by the band-pass filter. At the smaller sampling frequency, I chose 11000 Hz because I needed fewer points to be generated on Excel plots to see all the data at once. Using a sampling frequency of 44100 Hz resulted in the number of samples generated exceeding the capacity of Excel. Using 11000 Hz, the number of samples decreased, and the pole radius needed to be less precise in order to recognize the note designed for by the band-pass filter.

Other parameters that affected the output results were the limit put on the counter that solved the debounce problem, and the threshold value of the absolute value difference equation. If the counter was not high enough, the corresponding number of samples bypassed would not include all instances of the note being recognized multiple times, thus an iterative process of adjusting the count and

viewing the output was necessary for each separate WAV file. A closer observation of the output of the absolute value difference equation was necessary because the threshold value determined when the note was recognized with respect to time and how many times the note was recognized. If the threshold was put too high, the note would not be recognized, and if threshold was set too low the note would be recognized repeatedly hence causing the debounce problem to be more prominent.

Bibliography

Ambardar, Ashok. *Analog and Digital Signal Processing*. Pacific Grove, CA: Brooks/Cole Pub. Co., 1999. Print.

"Fred W. DePiero - Faculty & Staff - - Cal Poly." *Electrical Engineering Department*. Ed. Fred W. DePiero. Web. 29 Sept. 2009. <http://www.ee.calpoly.edu/faculty/fdepiero/>.

"Violin -." *Wikipedia, the free encyclopedia*. Web. 9 Dec. 2009. <http://en.wikipedia.org/wiki/Violin>.

"Playing the violin -." *Wikipedia, the free encyclopedia*. Web. 03 Dec. 2009. <http://en.wikipedia.org/wiki/Playing_the_violin>.

Equipment Costs

This project did not cost much to implement because I already had my laptop and acoustic violin. Both the Scilab and DSP software development environment were free downloads from Dr. DePiero's website, and I downloaded both for EE 419/459 class Winter Quarter 2009. The only two things I needed to purchase were the microphone for my laptop, and the chromatic tuner to help tune my violin before recording. The costs of each were as follows-

TABLE 2: EQUIPMENT NEEDED FOR PROJECT AND THEIR COSTS

Equipment Costs Summary	
Part	Cost
KORG Solo Tuner, Chromatic, CA-30 (multi-instrumental usage)	32.57
Logitech USB Desktop Microphone	45.87
Total	78.44

Schedule- Time Estimates

TABLE 3: SCHEDULE TIME ESTIMATES SPRING QUARTER 2009

Spring Quarter 2009		
Week	What was done	How many hours?
1	Research method of implementing senior project- FFT and spectrum analyzer	4
2	Research method of implementing senior project- FFT and spectrum analyzer	4
3	Research method of implementing senior project: Pole Zero Placement	4
4	Midterms week, did not research	0
5	Research method of implementing senior project: Using Analog Circuits	4
6	Research method of implementing senior project: Using Analog Circuits	4
7	Midterms week, did not research	0
8	Research method of implmenting senior project: Pitch Detection Algorithms	4
9	Research method of implmenting senior project: Pitch Detection Algorithms	4
10	Dead Week- Studied for Lab finals Did not research	0

11	Finals Week, did not research	0
Total hours of senior project		28

TABLE 4: SCHEDULE TIME ESTIMATES SUMMER 2009

Month	Week	What was done	How many hours?
July	1	Research real time methods of implementation	2
	2	Research real time methods of implementation	2
	3	Research non real time methods of implementation	2
	4	Research non real time methods of implementation	2
August	1	Pole Zero Placement filter hand calculations	2
	2	Pole Zero Placement filter hand calculations	2
	3	Sci Lab Simulations of filter calculations	2
	4	Editing Scilab code, observing effects	2
		Total hours of senior project:	16

TABLE 5: SCHEDULE TIME ESTIMATES FALL 2009

Fall Quarter 2009		
Week	What was done	How many hours?
1	recording music notes in practice room working with tuner to check my recordings	5
2	Reworking Scilab code after visiting office hours, starting to program difference equations	5
3	Simulating recorded notes using spectrum analyzer, and changing code to observe output	7
4	Midterms this week, only worked on difference	5

	equations code of Scilab and Dev C++ again	
5	EIT test during this week, just worked with recordings again to check accuracy	4
6	Working with spectrogram tool and running and editing code to observe change in output	6
7	Bringing Scilab, Spectrum Analyzer, and spectrogram tools all together and see how they affect one another and the output signal	5
8	Recording multiple notes cases- same note over and over, and alternating notes	5
9	Spent a lot of time with advisor and programmed state machine for absolute value difference equation, and state machine for solving the debounce problem. Also worked sending input/output, data to tables on Excel	15
10	Thanksgiving week, only worked on editing code to optimize filters	7
11	Gathered data of both good and bad cases created the poster for Expo on Friday	12
12	Finished finals early, spent remainder of week getting through final report	30
Total hours of senior project		106

Scilab Program Code

```
//Easy way to find coefficients
// Define r, f, S initally, use following equations
//to find A and B coefficients

r = 0.9999;
f = 220.0;
S = 44100;

A1 = -2*r*cos(2*3.141592654*(f/S))
A2 = r*r
B0 = 1-r

b = [B0];
a = [1, A1, A2];
N = 1024;
freqz_fwd(b,a,N,S);
```

This code acted similar to a .M file would in Matlab where the variables r, f, and S could be changed the outputted to Scilab to see the change in the output values of coefficients B_0, A_1, and A_2. Also the changes the parameters caused in magnitude versus frequency and phase versus frequency plots could be observed.

Dev C++ Program Code

Note: This code is a huge file because of generation sixteen band-pass filters, each having the same portion of code that perform similarly but have different variable names. A subroutine would have been a more efficient approach, but due to time constraints I had to stick with sixteen portions of same code. For simplicity sake and not to take up a large quantity of pages, I have shown two of the similar portions of code representing notes A and B of the G string. The code already given at beginning and end has been aligned all the way left while my code in the middle has been indented slightly from the farthest left towards the right side.

```
// main_wav.cpp : contains main routine for the console application.

/****** NOTICE: LIMITATIONS ON USE AND DISTRIBUTION ********\

 This software is provided on an as-is basis, it may be
 distributed in an unlimited fashion without charge provided
that it is used for scholarly purposes - it is not for
commercial use or profit. This notice must remain unaltered.

Software by Dr Fred DePiero - CalPoly State University

\********************* END OF NOTICE ************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cwrap.h>
#include <cwrap_math.h>

// main routine -
```

```
int main(int argc,char *argv[])
{
  long int num_samples;
  double fs_hz;
  int bits_per_sample;
  int num_ch;
  int n;
  double in,out;

  char ifname[512];
  char ofname[512];
  if(argc==2)
  {
       sprintf(ifname,"%s",argv[1]);
       sprintf(ofname,"%sdigital_filter_output.wav",get_directory(argv[1]));
 }
  else
  {
       sprintf(ifname,"noise_input.wav");
       sprintf(ofname,"digital_filter_output.wav");
  }

  // open_wav simulates an A/D
  //  samples are read in one at a time
  //  the sample rate, number of channels... are read from the wav header
  WAV_FILE *wav_in =
  open_wav(ifname,&num_samples,&fs_hz,&bits_per_sample,&num_ch);
  if(wav_in==NULL){ printf("can't create wav_in!\n"); wait_cr();  exit(-1); }

  // create_wav simulates a D/A converter
  //  this will generate a wav file rather than generating an actual analog voltage
  //  the wav file will have the format (sample rate...) specified via the arguments
  WAV_FILE *wav_out = create_wav(ofname,fs_hz,bits_per_sample,num_ch);
  if(wav_out==NULL){ printf("can't create wav_out!\n"); wait_cr();  exit(-1); }

////////////////////////////////////////////////////////////////////////

  // dsp students: don't change anything above this comment!!!!!!!!!!!
  // (dsp students: declare any needed variables here)
 // variables for difference equation and code for note A, G string
```

```
    double AGt;

    int AGstate_variable = 0;
    int AGprev_state_variable = 0;

    int AGsv = 0 ;
    double AGcount = 0;

////////////////////////////////////////////////////////////////////

   // variables for difference equation and code for note B, G string

    double BGt;

    int BGstate_variable = 0;
    int BGprev_state_variable = 0;

    int BGsv = 0 ;
    double BGcount = 0;

////////////////////////////////////////////////////////////////////

   // A and B coefficients for note A, G string
   // let r = 0.9999995 for 44100 Hz sample rate
   // let r = 0.99995 for 11000 Hz sample rate

   double outAG2;
   double outAG2prev1 = 0;
   double outprevAG1 = 0;
   double outprevAG2 = 0;


   double r = 0.9999995;
   double A1_AGstring = (-2.0*r*cos(2.0*3.141592654*(220.0/fs_hz)));
   double A2_AGstring = r*r;
   double B0_AGstring = 1.0-r;

   ////////////////////////////////////////////////////////////////////////////
   // A and B coefficients for note B, G string
   // let r = 0.9999995 for 44100 Hz sample rate
```

```
  // let r = 0.99995 for 11000 Hz sample rate

  double outBG;
  double outBG2;
  double outBG2prev1 = 0;
  double outprevBG1 = 0;
  double outprevBG2 = 0;


  double r1 = 0.9999995;
  double A1_BGstring = (-2.0*r1*cos(2.0*3.141592654*(246.94/fs_hz)));
  double A2_BGstring = r1*r1;
  double B0_BGstring = 1.0-r1;

////////////////////////////////////////////////////////////////////////////

    // FILE *fp = fopen("plot.csv","w");
    // These next lines of code help in sending the data of input WAV,
    // output difference equations, and state machine value at every n
    // sample to Excel

    FILE *fp = fopen("C:\\Documents and Settings\\Nathaniel
Mopas\\plot2.csv","w");
    if(fp==NULL){ printf("can't create plot.csv file!\n"); wait_cr(); exit(-1);
}
    printf("created file!\n");

 // Difference Equation values for input, output, and previous output
//generated

          for (n = 0; n < num_samples; n++)
   {

      in = read_wav(wav_in);

      if (n == 1)
      {
        outprevAG1 = out;
        outprevAG2 = 0;

        outprevBG1 = outBG;
```

```
        outprevBG2 = 0;

      }
      else if (n >= 2)
      {
        outprevAG2 = outprevAG1;
        outprevAG1 = out;

        outprevBG2= outprevBG1;
        outprevBG1 = outBG;

      }

      // All difference equations use fs_hz variable as sampling frequency

      //y(n) = B0_AGstring*x(n) -    A1_AGstring *  y(n-1)   -  A2_AGstring
*   y(n-2)
     out = ((B0_AGstring) * in) - ((A1_AGstring) * outprevAG1) -
((A2_AGstring) * outprevAG2);


 // Absolute Value of ouput difference equation, to determine signal
//strength and establish threshold value of out2 to recoginze the note being
//played

     outAG2 = (0.01 * fabs(out)) + (0.99*outAG2prev1);

     outAG2prev1 = outAG2;

////////////////////////////////////////////////////////////////////////////

   // state machine for determining value of out2 to recognize the note
   //change value of 5.0 depending on what is seen at Excel Output Data
   //tables

     if (outAG2 < 5)
     {
      AGstate_variable = 0;
     }

     else
```

```
      {
       AGstate_variable = 1;
      }

      if ( (AGstate_variable == 1) && (AGprev_state_variable == 0) )
      {
         AGt= (n/(fs_hz));

      }
    AGprev_state_variable = AGstate_variable;

////////////////////////////////////////////////////////////////////////////

   // state machine code for solving debounce problem

    if ((outAG2 >= 5.0) && (AGsv == 0))
    {
       AGcount = 0;
       AGsv = 1;
    }

   //increment counter as long as in sv1

    if (AGsv == 1)
    {
    AGcount = AGcount + 1;
    }

 // change what count equals depending on how long the debouncing
//occurs
    // for case of A B A on 44100 Hz sample rate, try count = 5000
    if ( (AGcount >= 5000) && (AGsv == 1) && (outAG2 >= 5.0) )
    {
    AGsv = 2;

    printf("Note A, G string (time = %1.3lf)\n",AGt);

    }

    // re-initialize counter to 0
    if ( AGsv == 2)
```

```
   {
   AGcount= 0;
   //count_sv2= count_sv2 +1  ;
   }

   // return from sv2 to sv0
   if ( (AGsv == 2) && (outAG2 < 5.0) )
   {
   AGsv = 0;

   }

////////////////////////////////////////////////////////////////////////////////


  // Code for B on the G string

  //y(n) = B 0_BGstring*x(n) -    A1_BGstring *  y(n-1)   -  A2_BGstring
*   y(n-2)
    outBG=((B0_BGstring) * in) - ((A1_BGstring) * outprevBG1) -
((A2_BGstring) * outprevBG2);

 // Absolute Value of ouput difference equation, to determine signal
//strength and establish threshold value of outBG2 to recoginze the note
//being played

   outBG2 =(0.01 * fabs(outBG)) + (0.99*outBG2prev1);
   outBG2prev1 = outBG2;


////////////////////////////////////////////////////////////////////////////////


    // state machine for determining value of outBG2 to recognize the note
    //change value of 5.0 depending on what is seen at Excel Output Data
   //tables

     if (outBG2 < 5)
     {
      BGstate_variable = 0;
     }
```

```
  else
  {
   BGstate_variable = 1;
  }

  if ( (BGstate_variable == 1) && (BGprev_state_variable == 0) )
  {
     BGt= (n/(fs_hz));

  }
 BGprev_state_variable = BGstate_variable;


////////////////////////////////////////////////////////////////////////////////


 // state machine code for solving debounce problem

 if ((outBG2 >= 5.0) && (BGsv == 0))
 {
    BGcount = 0;
    BGsv = 1;
 }

 //increment counter as long as in sv1
 if (BGsv == 1)
 {
 BGcount = BGcount + 1;
 }

 // change what count equals depending on how long the debouncing
//occurs
 if ( (BGcount >= 5000) && (BGsv == 1) && (outBG2 >= 5.0) )
 {
 BGsv = 2;
 printf("Note B, G string (time = %1.3lf)\n",BGt);

 }

 // re-initialize counter to 0
```

```
            if ( BGsv == 2)
            {
            BGcount= 0;
            }

            // return from sv2 to sv0
            if ( (BGsv == 2) && (outBG2 < 5.0) )
            {
            BGsv = 0;
            }

        ////////////////////////////////////////////////////////////////////////////////

// Printf statement that arranges order of columns of data in Excel


            fprintf(fp,"%d,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%d,%d,%d\n",n,in,out,outBG,o
            utAG2,outBG2,AGsv,BGsv);

             write_wav(wav_out,out);
            }

            fclose(fp);
////////////////////////////////////////////////////////////////////////
// dsp students: don't change anything below this comment!!!!!!!!!!!!

 // all done
close_wav(wav_out);

printf("\n\n OUTPUT WAV FILE HAS BEEN GENERATED :)\n\n");

wait_cr();
return 0;
}
```

Analysis of Senior Project Design

Summary of Functional Requirements

My project recognizes one of the sixteen notes of the violin's first position. This is accomplished by recording the plucked violin note and then sending the input signal through a C language program in Dev C++ software environment. The program performs an analog to digital conversion then sends the signal through a series of difference equations and state machines which results in a digital filter output. The output can be analyzed with a spectrum analyzer simulation, a signal image processing tool that displays a spectrogram, or with graphs on Excel generated by tables of data complied by the program.

Primary Constraints

A significant challenge to this project was the debounce problem. This hindered the project by creating a problem that was solved by a series of design trade offs each affecting one another. The debounce problem solution needed to take into consideration a count value for the number of samples run through to bypass the frame of time when the note was recognized more times than was actually played. However this count value varied for each note just because of the way I plucked the note and how much time I waited in between each pluck. Volume of each pluck determined the signal strength and the threshold value needed in the state machine that determined what value of the output difference equation was necessary to recognize the note being plucked. Lastly the plucking of the notes themselves were influenced by the position and close frequencies, since I had to be sure I was plucking

the correct note and not accidently moving too close in fingering position to the next highest or lowest note which would excite multiple note readings from my filters. As already shown in Figure 9 and Figure 10, there are many frequencies excited by the WAV file besides the fundamental and its harmonics, which can lead to the filters recognizing both the correct and incorrect notes. These parameters of count, threshold value, strength of pluck, position of finger are several amongst many that were closely intertwined with one another, therefore making this project sensitive and difficult to set parameters in the program that work with every case. Examples of varying strength of plucking include Figure 5 and Figure 6, both of which were different heights, notes close in frequency with note B slightly plucked louder than note A.

Another constraint aside from parameters included using Excel. Excel was limited in number of data points of rows that can be tabulated as 65,536 and using sampling frequency of 44100 easily exceed this with hundreds of thousands of sample points. Hence to see less data points as seen in Figure 13, Figure 14, and Figure 16 the sampling frequency was decreased to 11000 Hz to see the full waveforms.

Economic

The project did not have component parts needed to build anything, rather only the two pieces of equipment, the tuner and the microphone. The original estimated cost of the equipment at the start of my project was $50.00. I knew I needed a tuner to help tune the violin each time before recording as well as help confirm the note being plucked as it was recorded and I did not own a microphone. From Table 2,

actual final cost of the equipment was $78.44. The original estimated development time of my project was 75 hours since I figured it would be about half of the project time spent on development while the other half on research, preparing for presentation for the senior project expo, and putting together the final report. By the end of my project, and summing up the development time when observing Table 3, Table 4, and Table 5, the total came up to 84 hours.

If Manufactured on a commercial basis

The heart of this project stems from a starter program from my advisor, with the purpose in mind of being only used for education and not for commercial use or profit. This concept when programmed in a different software environment with the same or different programming language may be sellable. If this is the case, the program would most likely come from downloadable content from the internet with each download costing the user approximately how much a music tuner costs or even lower to attract the customers. Although this download may be limited to functionality so the other method of manufacturing this project would be that the software would come in a limited time frame of usage, analogous to student editions of engineering software programs like Matlab The number of devices to be sold per year would be in the millions due to the continual large interest in playing a musical instrument. Estimated manufacturing cost would be hundreds of thousands if the software was constantly worked on by experienced software developers. The estimated profits per year would be around hundreds of thousands because the number of people purchasing the download for this specific music software would

fluctuate due to competition of other music software note detectors/tuners. Estimated cost for user to operate the device would be around 100 to 150 dollars for a yearly activation of using the software.

Environmental

The environmental impact of my project even when extrapolating to more notes still cannot fully replace a violin tuner since it was not designed to recognize sharp and flat with respect to identifying the note. Rather this project acts as an aid only to recognize the violin note. Had the scope of this project included the ability to recognize sharpness or flatness with respect to identifying a note, it may be able to replace a tuner with just the musician using a computer and microphone combined with the program from the project. However this may be a bit ambitious seeing how the multiple parameters influence one another creating difficulty in just recognizing the note to begin with. When possibly replacing a tuner, this would decrease the quantity of broken or malfunctioning tuners thus reducing trash thus benefiting the environment.

Manufacturability

The issues and challenges with manufacturability include what happens with many things that can are downloadable or purchasable content. This project may be subject to pirating online, and multiple free downloadable copies circulating the internet thus destroying potential profits.

Sustainability

The issues and challenges when maintaining the completed device or system include the need for updates to add-ons to the software that may or may not be free after purchasing the yearly activation version. As a result the choice remains whether to charge more or not since this action would affect the desire and marketability of the product. This project impacts the sustainable use of resources by at least partially decreasing the quantity of music tuners manufactured allowing the components that make the tuner to be used in other pieces of equipment. Currently the project is designed for recognizing notes of the violin however an upgrade to this project would be to edit the software program to branch out and be usable in other musical instruments and increase the target audience. Performing this upgrade then brings up issues and challenges of taking away more of the music tuner market for other instruments hence creating a monopoly on the music tuner devices.

Ethical

Ethical implications relating to the design and manufacture include programming of the parameters to be limited in their functionality. An example would be to program the music note detector to recognize stringed instruments but then falsely advertise that the software works just as well on woodwind instruments when it may have not been tested and designed for them.

Health and Safety

A health and safety concern associated with use of this project is potential ear damage from users who may record their notes, and try to listen to the digital filter

output in loud environments. These users may rely on loud volumes of non noise cancelling headphones thus causing ear damage.

Social and Political

A social and political concern with the design of this project is that a programmer or user may try to hack the software or redesign it to create viruses or worms before pirating it to the internet. Any unsuspecting individual may download the software without know what damage it is causing to their computer. One social and political concern with this project is the division of people with those who want to legitimately purchase and own their version of the software versus those people who want to find it online for free and do not want to pay to use it.

Development

One of the tools I learned independently during the course of the project was the signal image processing tool that displayed the spectrogram. This provided an even more accurate method of reading the output and was a great intermediate step to see what exactly how the input and output of the filters were exciting other frequencies.