18-551, Spring 2005
Group 8, Final Report
I Want to Hear <u>YOU</u>!!!

Abigail Cyntje- aoc@andrew.cmu.edu
Jude Foster- jdf@andrew.cmu.edu
Fan Yang- fany@andrew.cmu.edu

Background

As a musician, or even as a casual music listener, have you ever listened to a polyphonic audio recording of a song you really like and think, "Man, I'd love to learn how to play that, but I can't hear all the notes in that keyboard part!" Or how about listening to a recording and think, "I wish I could get rid of that awful-sounding trumpet part!." It's easy to accomplish these things when you're in the studio while the song is still being recorded, but once the track leaves the recording studio and is made into a permanent sound file, you no longer have control of the individual instruments within the recording. While you can isolate individual sounds and "instruments" in electronically-produced music via MIDI, it is an extremely challenging problem to achieve this reliably with actual physical instruments. There have been research projects and papers written which outline the problem that we are trying to solve, and many are similar in concept. However, few, if any, attempt to perform instrument identification without transcription in MIDI, but rather use instrument identification as a means to get to MIDI.

While previous years' projects in 18-551 have been able to filter a polyphonic recording of multiple instruments into MIDI, we are attempting to isolate a single instrument within a polyphonic recording, and play back that isolated signal independently of the rest of the signal. We indeed made considerable progress in solving this problem, using audio processing techniques to profile instruments by their harmonic spectra, and isolating these target frequency profiles from the rest of the audio file through filtering. We then were able to playback this filtered profile independently from the other instrumentation.

What previous years' projects have done

Group 4, Spring 2000 – Auditory Scene Analysis (Cocktail Party Effect) In their demo, they separated an opera singer's voice from a background of string instruments. Our project should prove to be more complex than this because the characterization of the human voice is much different from other musical instruments in its harmonic spectrum, and therefore can be distinguished more distinctly.

Group 6, Spring 2004- MOZART: Making Orchestra Zippy; Automatic Reliable Transcription. Our project differs from this one in that while they took a recording and converted it to MIDI for transcription, we will not use MIDI, but rather analyze the raw signal and playback our final output as a WAV file.

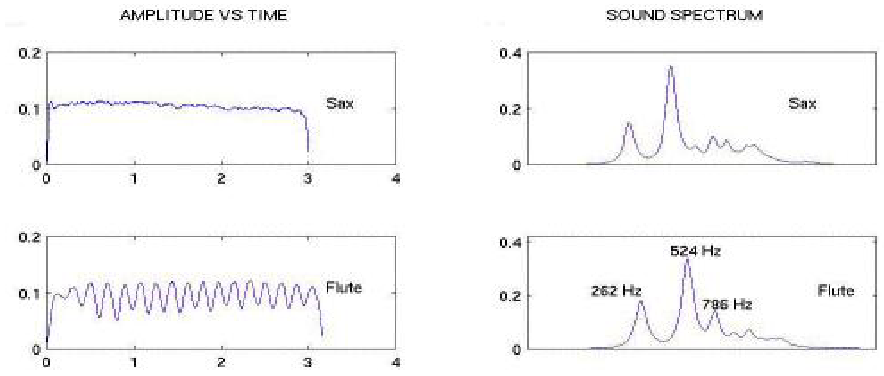
Group 9, Spring 2000- Real-Time Automated Transcription of Live Music into Sheet Music This project is an extension of the aforementioned MOZART in that it takes a transcribed MIDI file and converts it into sheet music.

Our project implementation consists of five main processes which make up our Pre-processing, Filtering, and Playback stages:

- Play input .wav file, and store samples' amplitudes in .txt file
- Based on the frequencies present, select comb filter coefficients to be applied to input
- Send all data to EVM for processing
- Perform filtering calculations on EVM to create new filtered samples
- Store resulting samples in file, and send back to PC

Pre-processing

In our implementation, we wanted to use a test/training song which would simulate the characteristics of a high-quality polyphonic audio recording, making our project relevant to music listeners. So we looked at the characteristics of studio-quality recordings and chose our training song parameters accordingly. We decided to use "Mary Had A Little Lamb" as our test song to keep things simple, and recorded three musicians, two alto saxophone players (we only used one for the final recording), and the other, an alto flute player, under controlled conditions in a recording studio in the basement of the Purnell Center for the Arts at Carnegie Mellon University. We chose these instruments for our recording, deliberately staying away from string instruments, which can produce widely-varying spectra based on, among other things, the characteristics of the individual strings. Thus we chose common instruments which weren't as subject to such variables. The saxophone and flute were ideal for this project because their timbres are quite different, as we discovered looking at the frequency spectrum for each instrument for the range of frequencies we recorded [1].



We made a digital recording of the instruments using a Sony C48 cardioid studio microphone, at a 44.1 kHz sampling rate, quantized to 16-bit stereo. To have notes from which to profile the instruments and create our database of music samples, we recorded each musician playing the notes of a major scale over two octaves; between C4 and C6 for the flute, and between A3 and A5 for the saxophone. (This range of octaves include the same frequencies [262 and 1047 Hz], but the notation differed because the saxophone is a transposing instrument in the key of Eb). Using a sound level meter, we measured each musician's loudness so that all of the musicians would have a uniform sound pressure level.

One feature of professional studio recordings that we took into account for this project is the notion of spatial location. Most recordings don't have all the instruments panned dead center; humans have excellent horizontal spatial hearing recognition, so recording engineers take advantage of this characteristic of human hearing. Some instruments are panned extreme left, some extreme right, and others everywhere else in between. Of course, this varies with the recording engineer and artist's preferences, but this is a tool used to provide the listener with a wider stereo effect, and to accent certain instruments in a particular spatial location. In our recording, we panned the saxophone slightly left at about a 3:1 left/right channel energy ratio, and panned the flute slightly right at about a 1:3 left/right channel energy ratio to achieve this similar effect.

We recorded "Mary Had A Little Lamb" in the frequency range of A3-E4(sax)/ C5-G5(flute), which falls in the 523 Hz-784 Hz range. Because of the simplicity of "Mary Had A Little Lamb", only 4 fundamental frequencies are included in the song, making it easier for us to analyze; the 4 fundamental frequencies present were at 523 Hz, 587 Hz, 659 Hz, and 784 Hz. The length of the recording was approximately 18.63 seconds, which gave us 821,661 recorded samples using our sampling rate of 44.1 kHz.

Once we finished recording all of our samples, we saved each channel as an uncompressed .wav file, and converted those .wav files into .txt files to be used as our input into Matlab for frequency analysis. Looking at the FFT of the individual notes in the song, we noticed that the saxophone and flute didn't always play in tune with each other, making their fundamental frequencies slightly different. This turned out to be a blessing in disguise as it allowed us to create filters that would be able to separate the two instruments' frequencies easier.

Using the function *firpm,* we were able to calculate the filter coefficients for the FIR filter we wanted to apply to our signal. We then used these filter coefficients to create as series of narrow band-pass filters at the fundamental frequency as well as at interval harmonic multiples, and applied the resulting set of filters to each sample using a series of tap delays. Using the knowledge of when each fundamental frequency would be played during the song, we applied a custom set of filters based on the fundamental frequency present at a given point in the song. We adjusted each filter's bandwidth as necessary depending on how close the sax and flute frequencies were to each other, and which harmonics we wanted to filter.

Transfer to EVM and Filtering

After building and testing the filters in Matlab, we implemented them in C so that it could interface with our input signal on the DSP. The DSP we used for data processing was the Texas Instruments TMS320C6701 Evaluation Module (EVM). We transferred both our data and filter coefficient data one sample at a time to the EVM for processing. To account for time and memory constraints on the EVM (8 MB on-board memory, 64kB on-chip memory), we decided to cut our signal in half and only filter the first half of our song. Our filtering function was then applied to each sample, and the resulting filtered sample was stored as an element in an array.

Once the filtering was completed on our data, we transferred it back to the PC and saved the data as a .txt file.

Playback

Originally, we tried to achieve instant playback of each filtered sample in real-time. After many failed attempts at this (and time constraints), we decided to play the filtered result as a whole once the entire result array has been transferred back to the PC into the .txt file. After the entire .txt file was written, we went back to Matlab and used the *dlmread* function to convert the .txt file data into data that could be played back as audio. Finally, we used *wavplay* to play our final filtered result.

Data Analysis

At first, we considered using a Short time fourier transform or Wavelet transform to analyze the musical signal and use the analyzed information to help us filter out notes. We decided then considered using the wavelet transform because of its higher time resolution at high frequencies and higher frequency resolution at lower frequencies. This is suitable for music analysis because scale in music is logarithmic. For example, the first octave has a frequency range of 16.35Hz but the seventh octave has a frequency range of 2093 Hz. We considered using subband encoding to implement the discrete wavelet transform[2] because of its low computational demands.

We finally decided to chose to implement a filter for either the saxophone or flute in our sound clip of "Mary Had A Little Lamb" without real time frequency analysis. The filters were designed with frequency analysis and timing analysis done using short time fourier transform and fast fourier transform.

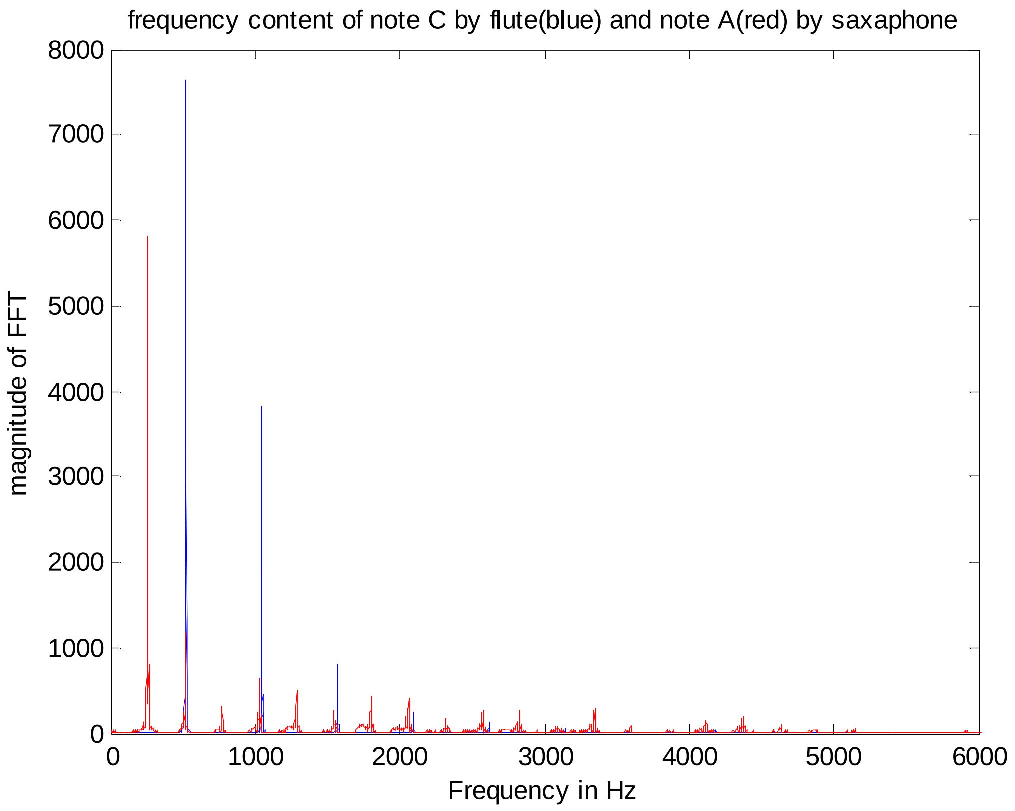
When designing the frequencies for our bandpass filters, we used a fast fourier transform in matlab to determine the fundamental frequency of each note based on a short wav clip of the notes by themselves. The flute played the notes C, D, E, and G while the saxophone played the notes A, B, C sharp, and E.

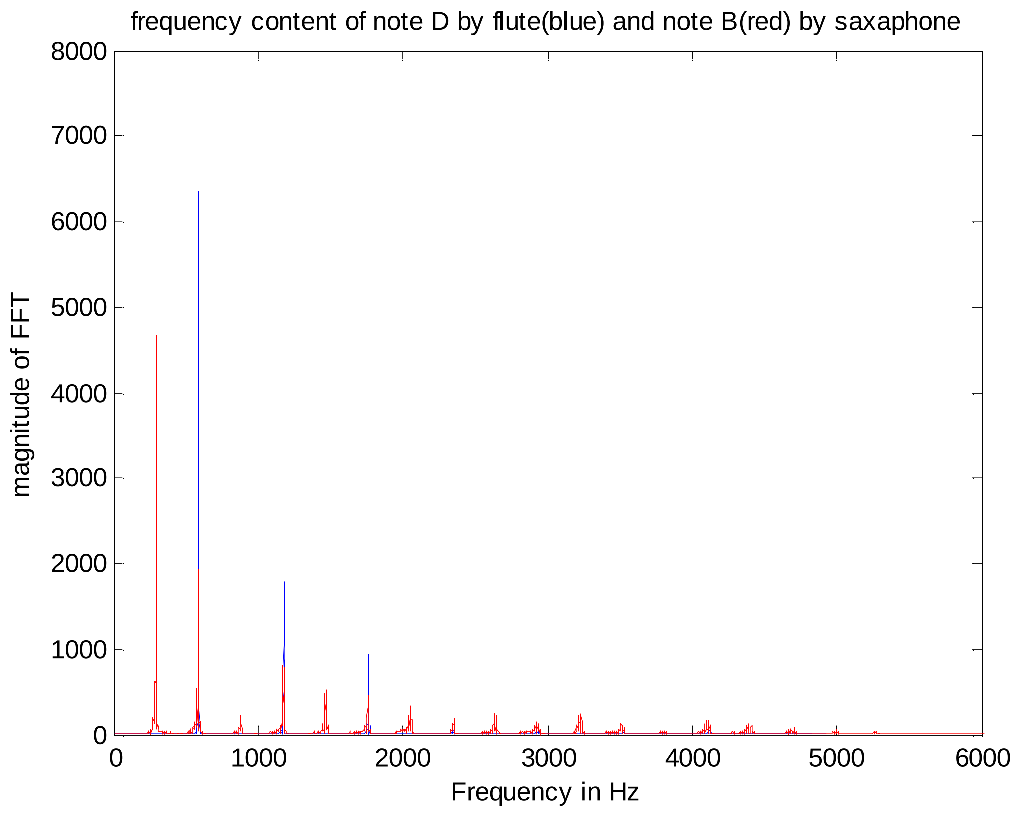
We analyzed the signal using a 44100 point FFT so that the frequency resolution was 1hz. When the magnitude of the FFT was plotted, the frequency content could easily be read.

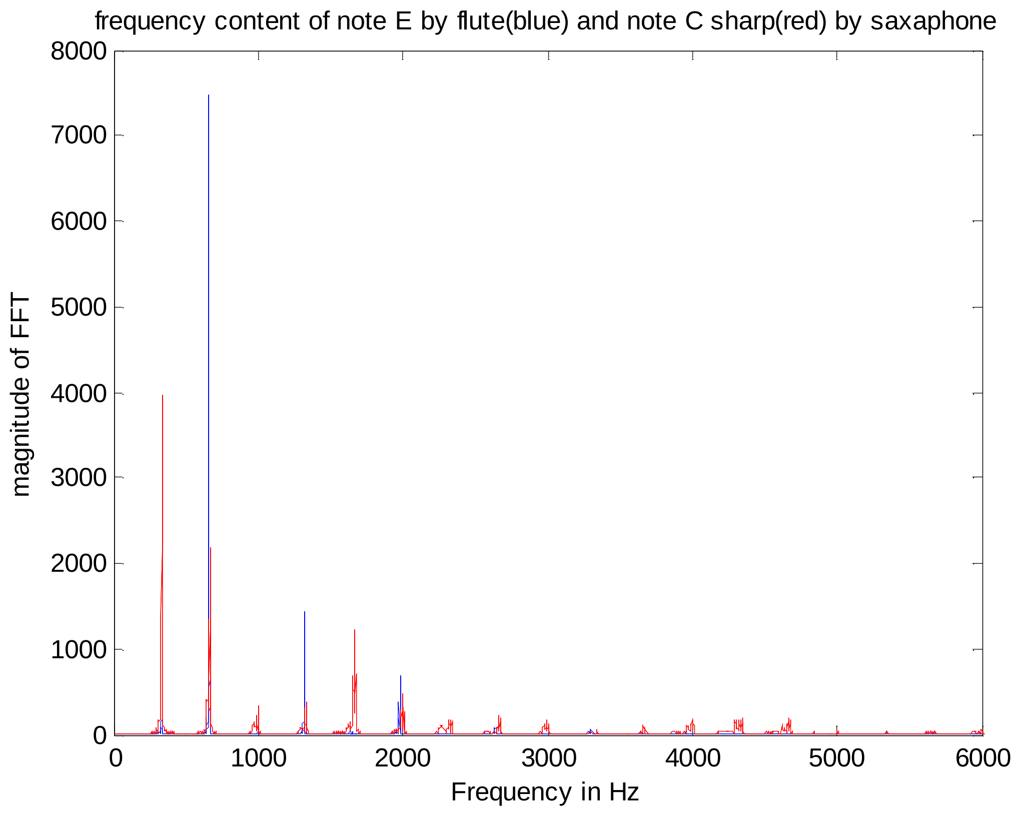
We used the following matlab function to plot the FFT for saxophone C Sharp:

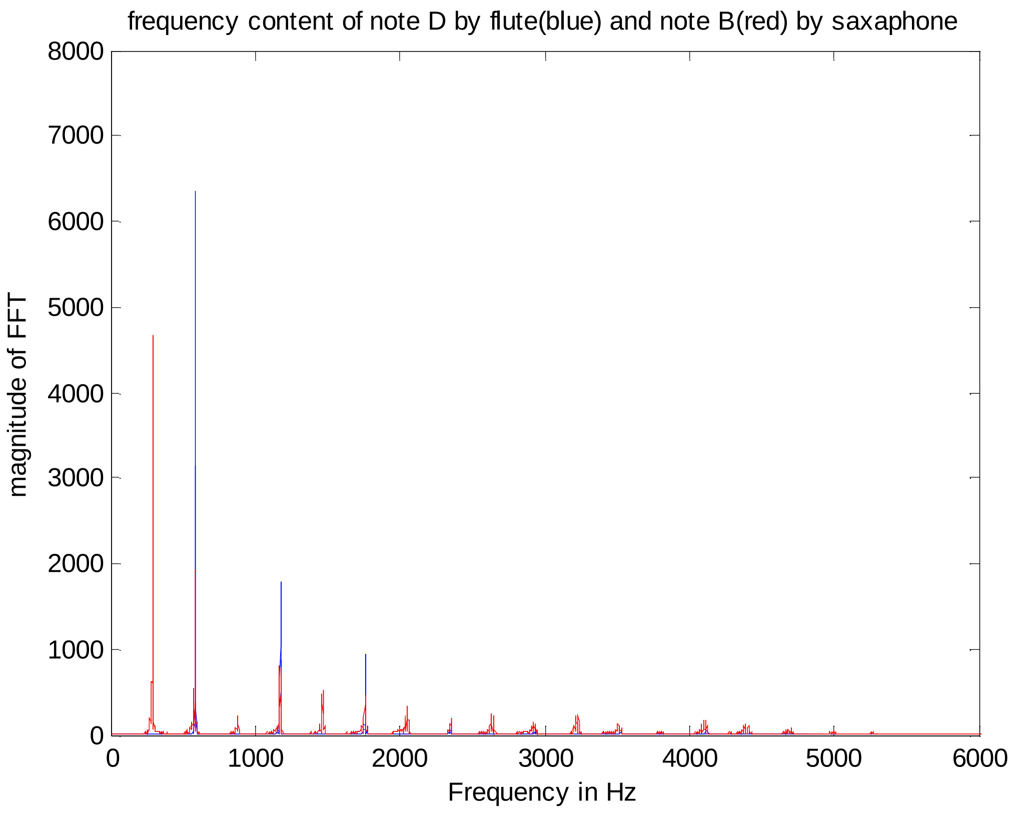
```
plot(abs(fft(sax_c_sharp,44100))
```

The number of filters we used depended on the relative amplitudes at each harmonic of the saxophone and the flute. For the Flute, we used about 4-6 filters because the amplitudes of the harmonics decreased steeply. For the flute, the 4th harmonic had an amplitude of 15% of the fundamental while the 5th harmonic had an amplitude of about 2% of the fundamental. For the saxophone, the 4th harmonic of the saxophone had an amplitude that was about 10% of the fundamental and the 10th harmonic had a amplitude 5% of the fundamental.
The following graph shows the frequency and amplitude distribution of a note played by the flute, along with the same note played by the saxophone at a higher octave.




frequency content of note D by flute(blue) and note B(red) by saxaphone
magnitude of FFT
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000
5000
6000
Frequency in Hz


frequency content of note E by flute(blue) and note C sharp(red) by saxaphone
magnitude of FFT
8000
7000
6000
5000
4000
3000
2000
1000
0
0
1000
2000
3000
4000
5000
6000
Frequency in Hz



The naming convention of the notes on the two instruments are different but the frequencies played are the same. Note A on a saxophone corresponds with note C on a flute. For each note that the saxophone played along with the flute, the saxophone was exactly one octave above the flute. This means that the harmonics of the flute occurs at the same frequencies as the even harmonics of the saxophone. The table below shows the fundamental frequencies of the notes played in Mary Had a Little Lamb. The actual recording contained notes that were either slightly sharp or flat. The frequency content deviated from the listed values by around 5Hz for the fundamental, and up to around 100Hz for the higher harmonics of the saxophone.

Fundamental Frequencies of notes [3]

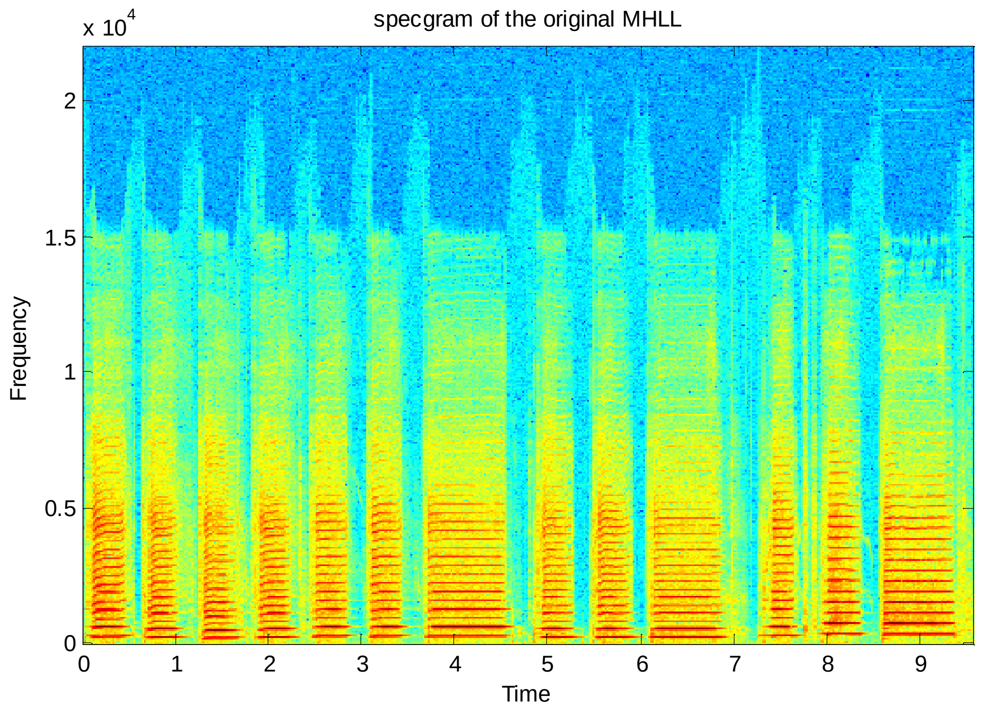
Flute c	523.25hz	Sax a	261.63hz
Flute d	587.33hz	Sax b	293.66hz
Flute e	659.26hz	Sax c sharp	329.63hz
Flute g	783.99hz	Sax e	392.00hz

A spectrogram was used to approximate the length of each note in the wav file of Mary Had a Little Lamb. Red represents high intensity and blue represents low intensity. The spectrogram information was used to determine the length each filter needed to be applied. The short notes are about .6 seconds long and the long notes are about 1.2 seconds long. The spectrogram also shows us the frequency content over time. We used this to check that the frequencies were constant over the .6 second and 1.2 second time frame during which each note was played.

The matlab function that displayed the spectrogram:

```
specgram(MHLL,2000,44100)
```

We selected a 2000 point FFT because it provided a good tradeoff for time resolution vs. frequency resolution for our purposes.



Design of Filter:

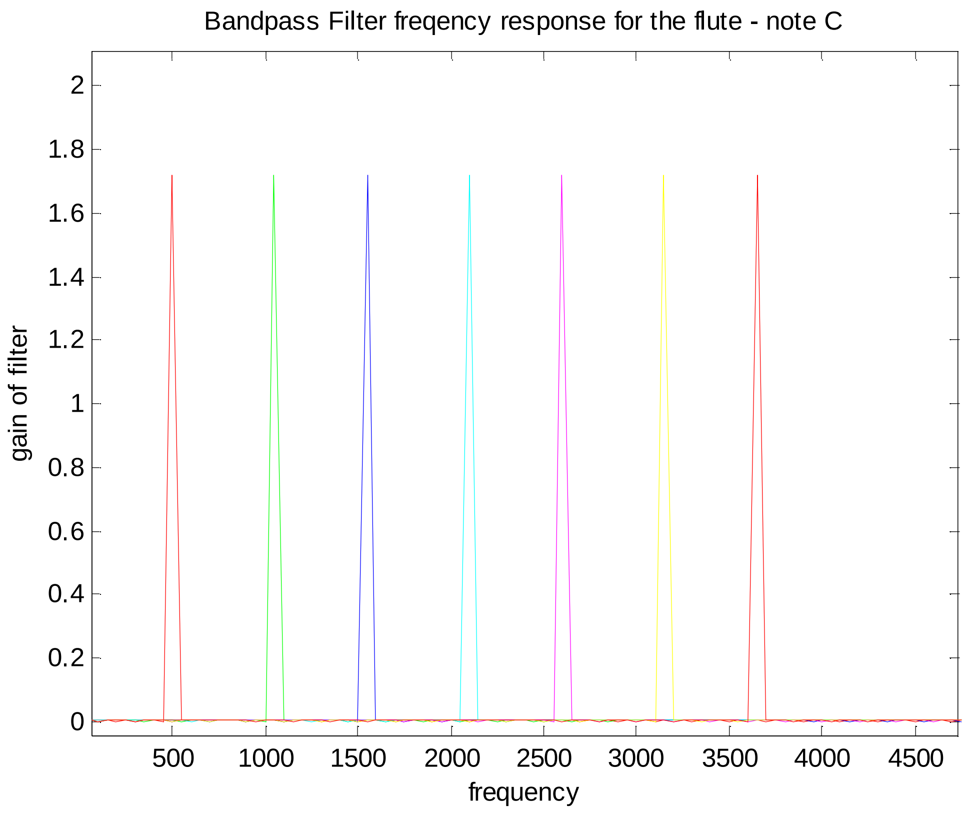
All our filters had the same specifications of 3db in the passband, 40db in the stopband. The filters had a bandwidth of 100Hz and a gain of 1.72. The order of each filter was estimated using the matlab function firpmord to be 4631, and the FIR filter was designed using the matlab function firpm. Once each filter was designed, all 4632 coefficients of the FIR filter was outputted to a text file for input. We considered changing the filter gain at different frequencies so that the filtered signal would have amplitudes proportional to the instrument being filtered. This however, would change the bandwidth of our filter and we did not want to make this compromise.

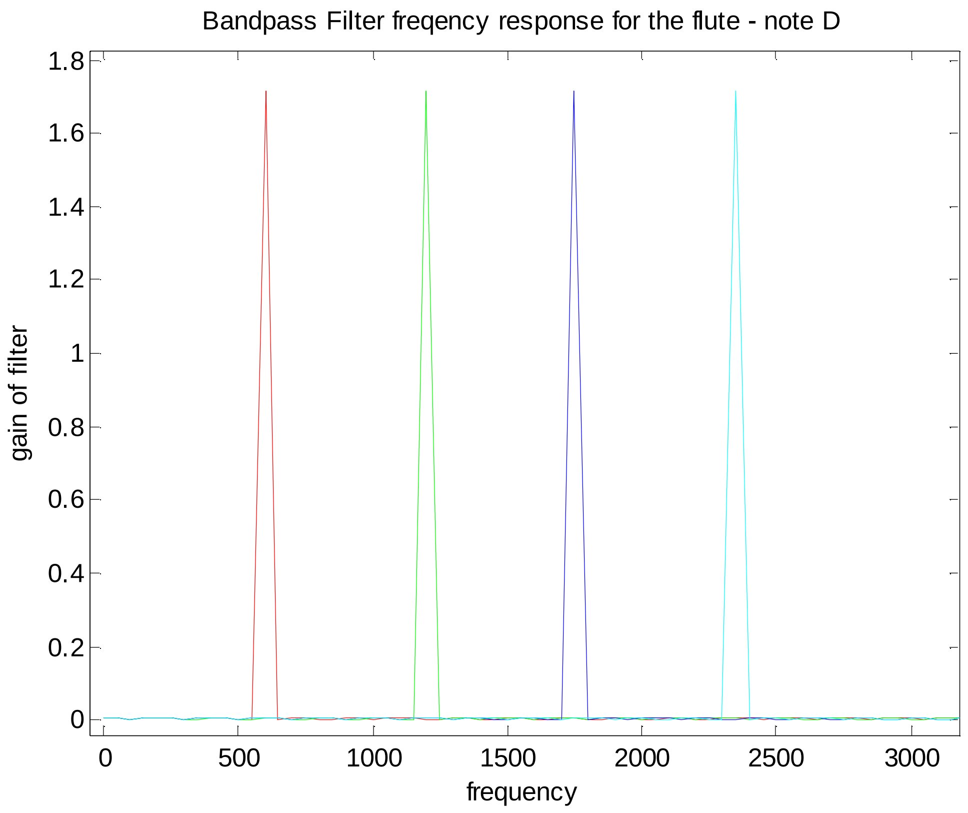
When we designed the set of filters, we chose the design of a filter we had at 350Hz as our reference filter, and we offset the values that we passed to the firpmord and firpm functions in matlab by 50Hz at a time because the filter characteristics changed when we varied the frequency by any amount less than 50. With a bandwidth of 100Hz, our filters were used to easily pass any frequency that we wanted.

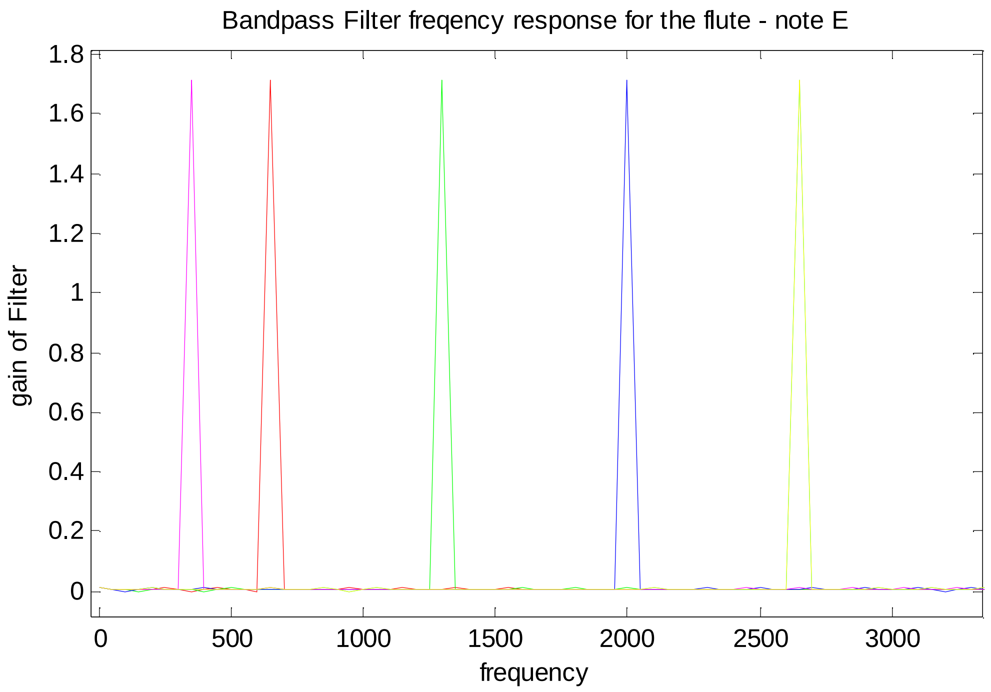
We obtained code from Professor Stern for comb filters, but we did not use them because by then, our bandpass filters were already implemented and working properly.

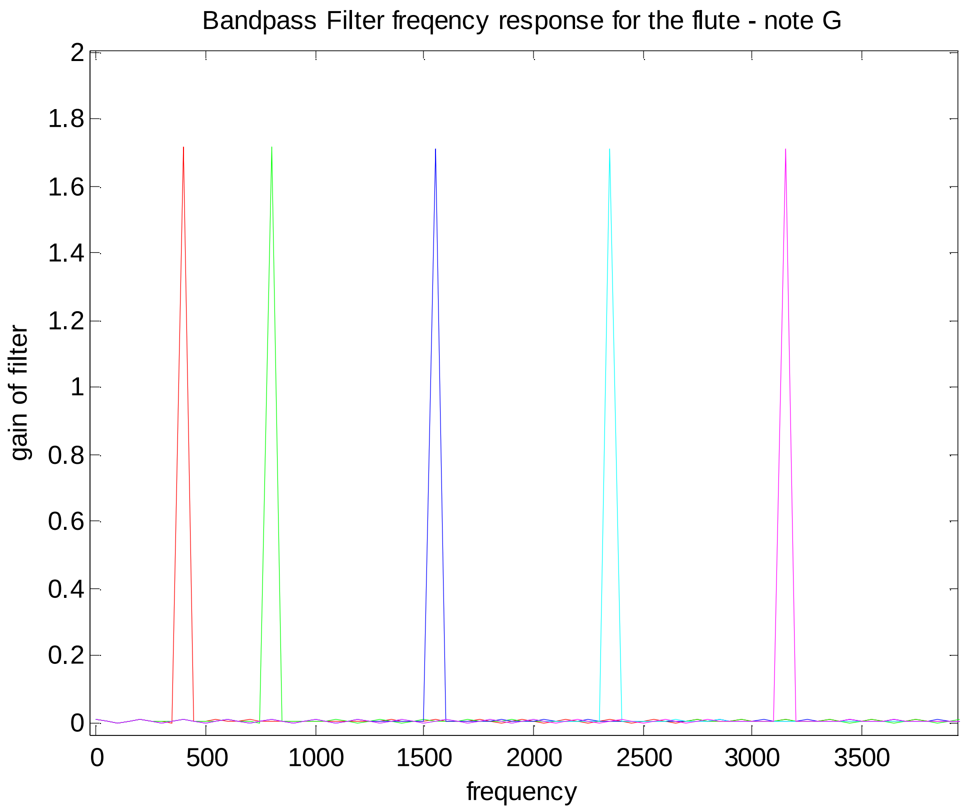
The following shows the frequencies of the filters that were used for the flute and the saxophone. Not all the harmonics were filtered out because some higher harmonics had higher amplitude than lower hamonics. As previously described, our filters had center frequencies that are multiple of 50Hz:

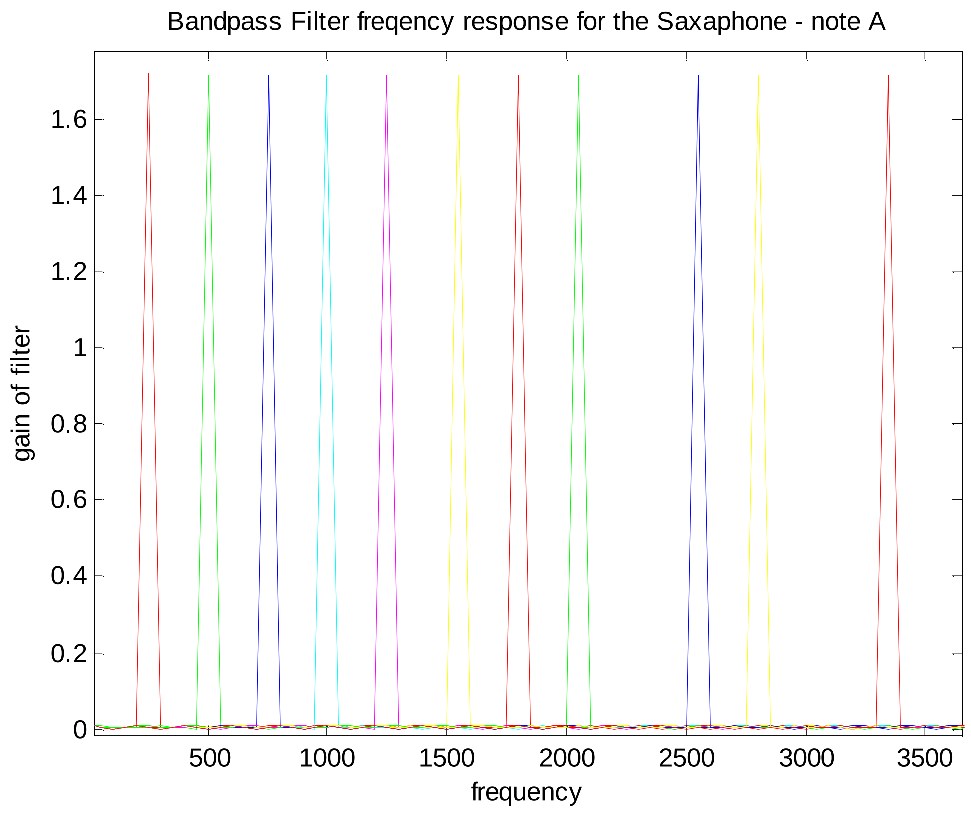
Flute C	Flute D	Flute E	Flute G	Sax A	Sax B	Sax C#	Sax E
500	600	350	400	250	300	350	400
1050	1200	650	800	500	600	650	800
1550	1750	1300	1550	750	900	1000	1200
2100	2350	2000	2350	1000	1150	1350	1600
2600		2650	3150	1250	1450	1650	2000
3150				1550	1750	2000	2400
3650				1800	2050	2300	3850
				2050	2350	2650	4350
				2550	2650	3000	4750
				2800		4000	
				3350		4350	

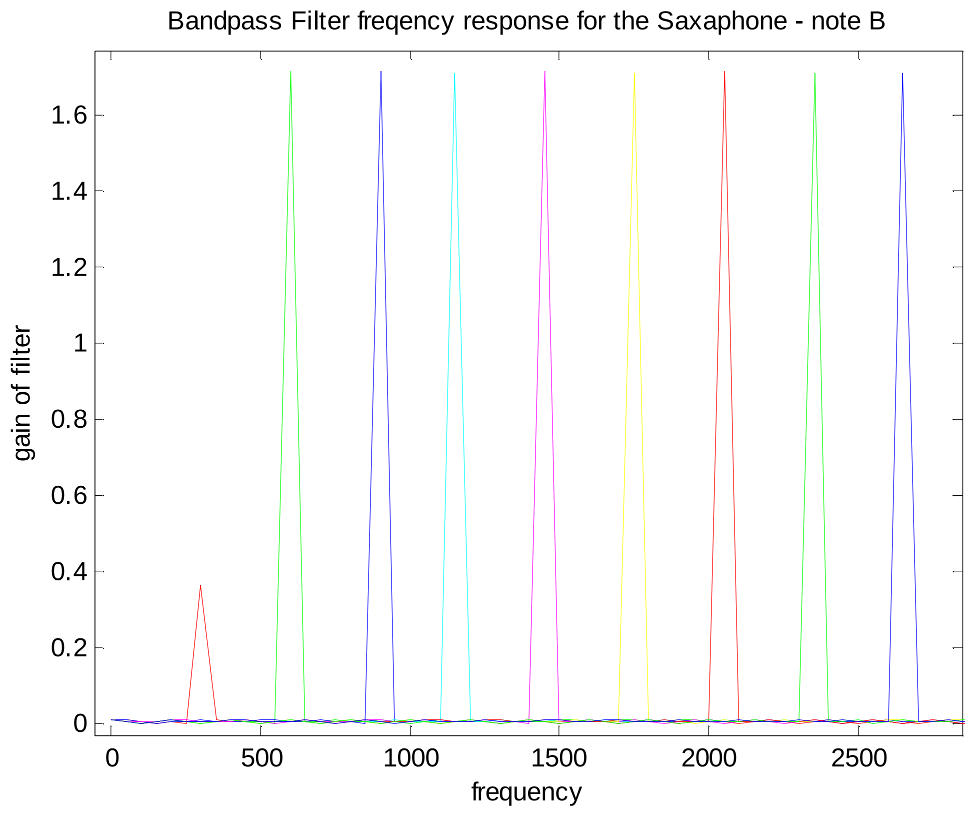

Bandpass Filter freqency response for the flute - note C
gain of filter
frequency
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
500
1000
1500
2000
2500
3000
3500
4000
4500

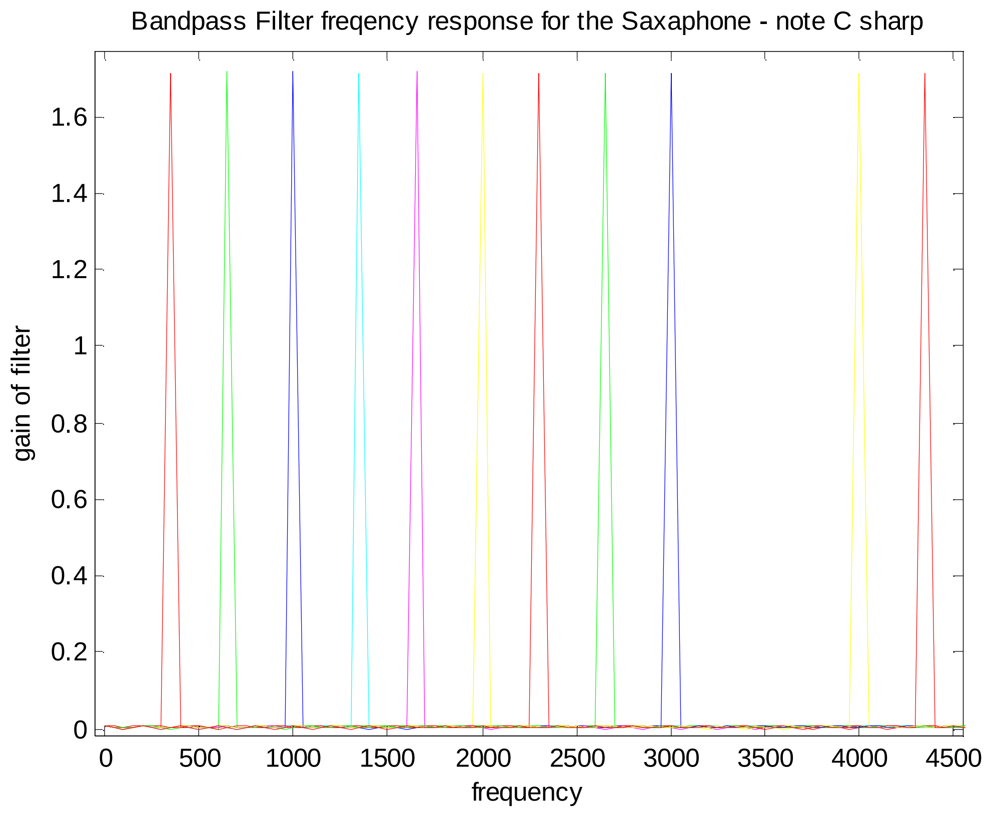

Bandpass Filter freqency response for the flute - note D
gain of filter
frequency
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
0
500
1000
1500
2000
2500
3000

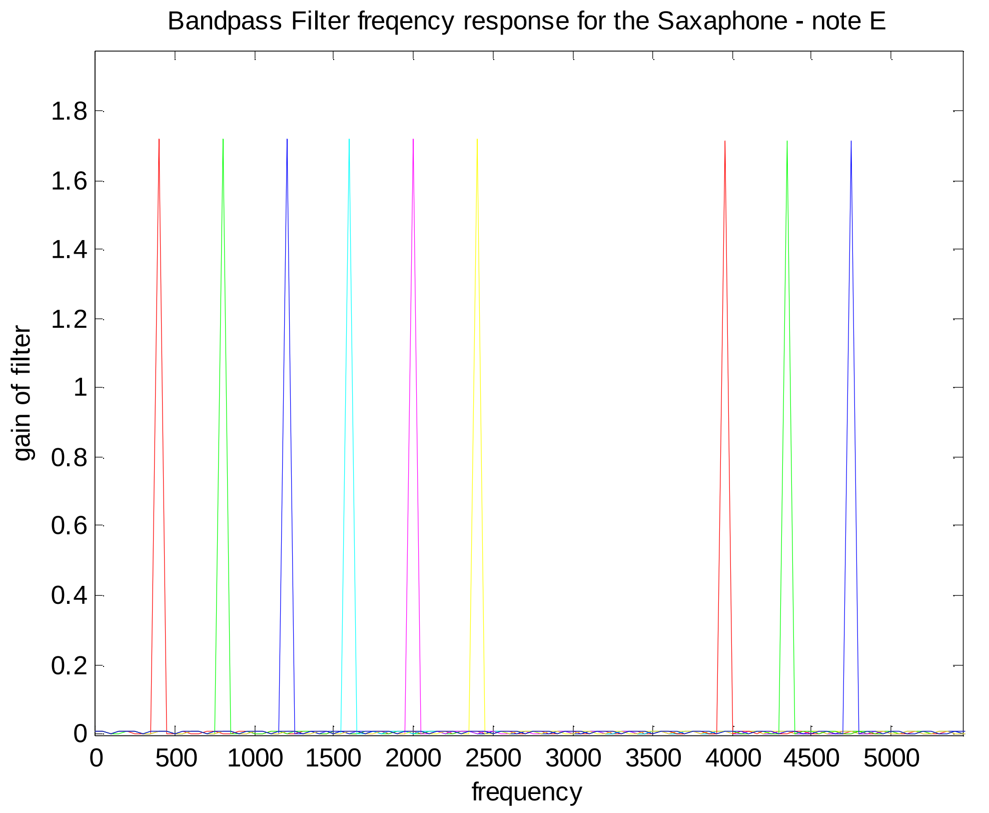

Bandpass Filter freqency response for the flute - note E
gain of Filter
frequency
0
500
1000
1500
2000
2500
3000
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8


Bandpass Filter freqency response for the flute - note G
gain of filter
frequency
0
500
1000
1500
2000
2500
3000
3500
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Bandpass Filter freqency response for the Saxaphone - note A
gain of filter
frequency
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
500
1000
1500
2000
2500
3000
3500


Bandpass Filter freqency response for the Saxaphone - note B
gain of filter
frequency
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
0
500
1000
1500
2000
2500


Bandpass Filter freqency response for the Saxaphone - note C sharp
gain of filter
frequency
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
0
500
1000
1500
2000
2500
3000
3500
4000
4500


Bandpass Filter freqency response for the Saxaphone - note E
gain of filter
frequency
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

Panning Algorithm

The Panning Algorithm was the first all-encompassing algorithm we tried to implement, and would've have worked had we implemented our filters in the frequency domain rather than using tap delay filters in the time domain. This is an algorithm developed by Creative Labs, which exploits the notion of recording different instruments with different left/right channel energy ratios. [4] This ratio, or the spatial panning index as they called it, is what they used to identify instruments. First the signal was split into its respective left and right channels for individual processing. Next, the STFT of each channel was taken, and the ratio of the magnitudes of the STFTs was stored in a time/frequency matrix. The values of the matrix were then compared to a ratio chosen by the programmer and stored into a Boolean matrix. This Boolean matrix represents the values in time/frequency where the chosen energy ratio is present in the signal. The Boolean matrix is then multiplied by the STFT matrix of each channel, which filters out the signal at ratios not present in the Boolean matrix. The inverse STFT is taken of this newly filtered matrix, and the resulting vector in the time domain is the recording of only instruments recorded at the ratio chosen by the programmer. Therefore, you could choose any ratio between 0 and 1, and filter all the instruments not playing with the chosen L/R frequency energy ratio. Naturally, this algorithm only works if you don't have collisions between instruments in both the time and frequency domain that have been recorded at the same L/R ratio.

Other Algorithms/Methodologies Explored

The heart of our project is the filtering process and the spectral analysis that is used to create the filters. We researched many different methods used for spectral analysis, and we found that none of them best fit our needs and were somewhat extraneous for what we wanted to achieve. Listed below are a few of these methods, how they could've helped, and why we ultimately looked at a simple FFT instead of using that method.

Short Time Fourier Transform (STFT)

The general problem with FFT is that it only indicates the presence of various frequencies within the signal. It doesn't include information about when different frequencies occur or how they change over time. STFT addresses this by computing the FFT for small overlapping windows of audio; this produces a slowly time-varying FFT. Using a window of 256 samples and an overlap of 220 samples, the same 5 seconds are represented as 3056 adjacent FFTs. Each FFT contains 257 numbers evenly spaced across the frequency range [0 11,025]Hz. This totals 528,392 numbers to represent the same 5 second signal. Each column in the image represents the coefficients of the FFT for that window. [5]

However, we found that windowing and using the STFT on our input greatly increased the amount of calculations and time needed to implement. Also, because we knew beforehand which frequencies would be present in the signal at a given time, we used the FFT to look at the isolated note, and that was sufficient to implement our filtering.

Constant-Q Transform (CQT)

The transform is similar to STFT but, the window length varies as function of frequency so that a constant number of periods are within the window at each frequency. The CQT has a geometrically distributed frequency resolution. It is better suited for music signal analysis, since we see a mirror of such a distribution in frequency resolution in direct comparison to the music scale. The STFT is tailored to match the contraction of the complex exponential as:

$$S^{CQ}(t,\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} s(\xi)h\left((t-\xi)\frac{\omega}{\omega_0}\right)\exp(-j\omega\xi)\,d\xi$$

Here the h(.) is the sliding window and ω0 is the reference frequency at which the window is unaltered. If the fundamental frequency is estimated by some peak picking technique, then this transform gives the peaks indicating the pitch of the note. [6]

The efficient implementation of CQT relies on FFT in such a way that it practically combines several bins of FFT to produce a logarithmic frequency resolution. It therefore fails to interchange the weaker frequency resolution at the high end to a better time resolution, which would be desirable to model human hearing. On the other hand, straightforward calculation of CQT proved to be too laborious to be practically useful to us.

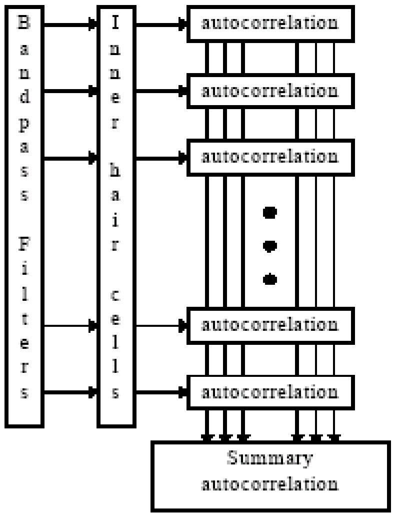
The Discrete Wavelet Transform (DWT)

The Wavelet Transform (WT) is a technique developed as an alternative to the STFT to overcome problems related to its frequency and time resolution properties. More specifically, unlike the STFT that provides uniform time resolution for all frequencies the DWT provides high time resolution and low frequency resolution for high frequencies and high frequency resolution and low time resolution for low frequencies. In that respect it is similar to the human ear which exhibits similar time-frequency resolution characteristics. [7]

If we were trying to achieve high-fidelity results for our project, the DWT might have been appropriate to use. However, the code is complex, and time and efficiency issues eventually led us away from this analysis method.

Fundamental Frequency Estimation (FFE)

The Fundamental Frequency Estimation (FFE) algorithm uses Summary Autocorrelation to show the overall periodicity properties of the signal. A diagram of how this is achieved is displayed below: [8]



As was the case with many of the other algorithms, a lot of the processes involved were overkill for the purposes of our project. What we did use from this algorithm, the band-pass filters were enough for us to get reliable results. This is attributed to the fact that there was little or no overlap of fundamental frequencies in our recording between the sax and flute.

Timing Analysis

Function/Data Profiled	**Transfer Time(secs)**
switch_style	0
switch_instr	0.891
fn_note1	21.352
fn_note2	0
fn_note3	0
fn_note4	0.015
size1	0.016
size2	0
size3	0
size4	0
note1	11.207
note2	6.878
note3	11.348
note4	7.878
Dsize	0
Data	17.742
Rsize	0
Result(+computation)	792.7

Memory issues:

One of the major issues in speed performance was due to the capacity of the on-chip memory. The chip could hold as much as 64KB of data, but with the other code that was on-chip (i.e filtering), our on-chip capacity was reduced to about 44KB.
The largest block size we could fit on chip was: 4410 samples*4 bytes=17.64KB
However, the block size that we needed was: 13230 samples*4 bytes=52.9KB

Rough Calculation of Result

Each filter in our "comb filter" which is really a series of band pass filters require 4632 coefficients. Thus the smallest block we can calculate correct results using a sampling rate of 44.1kHz and a note size of .6s is 13230 samples or half of a note. Below is a rough estimation of how long it would take to filter each note for a particular instrument.

Each note of an instrument requires a certain amount of filters to capture all the dominant harmonics present, we will call this number *filter_num*. Also each filter has 4632 coefficients, we will call this number *coeff_size*. The block size we require to compute the correct results is 13230, we will call this number block_size. In order to compute the results we must run all the filters on each sample, we will call each sample n. Since our block size is too large to fit on the ON_CHIP_DATA memory location we were forced to call all our samples from off chip which requires 15 cycles, we also had to do this for the coefficients before we optimized.

Not Optimized:

For each note the number of cycles to compute the results is

for n <= coeff_size the # cycles = 2*coeff_size*(coeff_size + 1)*(15 +15)

since each sample takes 15 cycles and each coefficient takes 15 cycles

for n > coeff_size the # cycles = 2*(block_size – coeff_size)*coeff_size*(15+15+15)

since each sample takes 15 cycles and we need two samples each time and each coefficient takes 15 cycles.
= filter_num * [2*coeff_size*(coeff_size + 1)*(15 + 15) + (2*(block_size– coeff_size)*coeff_size*(15+15+15)]

Thus our code takes about 36.54s per filter

Timing Estimate for flute

Note	# of Filters	Time in mins
Note1	6	3.654
Note 2	4	2.436
Note 3	7	4.263
Note 4	5	3.045

Timing Estimate for Saxophone

Note	# of Filters	Time in mins
Note1	9	5.481
Note 2	11	6.699
Note 3	9	5.481
Note 4	11	6.699

Since we ended up using only the first 3 notes of the song to process, the overall time it would take to filter these 3 notes is: **flute:** (3.654+2.436+4.263)=<u>10.353 mins</u>. **Sax:** (5.481+6.699+5.481)=<u>17.661 mins</u>.

Optimized:

For each note the number of cycles to compute the results is

for n <= coeff_size the # cycles = 2*coeff_size*(coeff_size + 1)*(15)

since each sample takes 15 cycles and each coefficient takes 1 cycle

for n > coeff_size the # cycles = 2*(block_size – coeff_size)*coeff_size*(15)

since each sample takes 15 cycles and we need two samples each time and each coefficient takes 1 cycle.

= filter_num * [2*coeff_size*(coeff_size + 1)*(15) + (2*(block_size– coeff_size)*coeff_size* (15+15)]

Thus our code takes about 22.75s per filter

Timing Estimate for flute

Note	# of Filters	Time in mins
Note1	6	2.275
Note 2	4	1.517
Note 3	7	2.654
Note 4	5	1.896

Timing Estimate for Saxophone

Note	# of Filters	Time in mins
Note1	9	3.413
Note 2	11	4.171
Note 3	9	3.413
Note 4	11	4.171

Since we ended up using only the first 3 notes of the song to process, the overall time it would take to filter these 3 notes is: **flute:** (3.654+2.436+4.263)=10.353 mins. **Sax:** (3.413+4.171+3.413)=10.997 mins.

Ideal Situation:

If we could fit both the data samples and coefficients on chip!

For each note the number of cycles to compute the results is

for n <= coeff_size the # cycles = 2*coeff_size*(coeff_size + 1)*

since each sample takes 1 cycle and each coefficient takes 1 cycle

for n > coeff_size the # cycles = 2*(block_size – coeff_size)*coeff_size

since each sample takes 1 cycle and we need two samples each time and each coefficient takes 1 cycle.

= filter_num * [2*coeff_size*(coeff_size + 1) + (2*(block_size– coeff_size)*coeff_size]

Thus our code takes about .919s per filter

Timing Estimate for flute

Note	# of Filters	Time in mins
Note1	6	.092
Note 2	4	.061
Note 3	7	.107
Note 4	5	.077

Timing Estimate for Saxophone

Note	# of Filters	Time in mins
Note1	9	.138
Note 2	11	.168
Note 3	9	.138
Note 4	11	.168

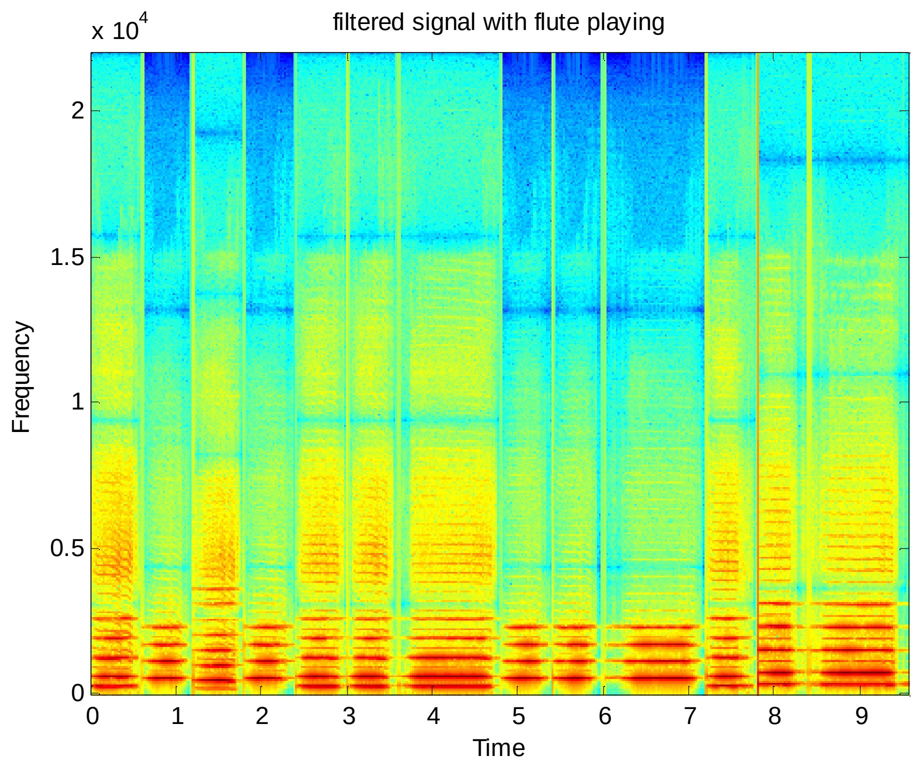
Since we ended up using only the first 3 notes of the song to process, the overall time it would take to filter these 3 notes is: **flute:** (.092+.061+.107)=0.26 mins. **Sax:** (.138+.168+.138)=0.44 mins.

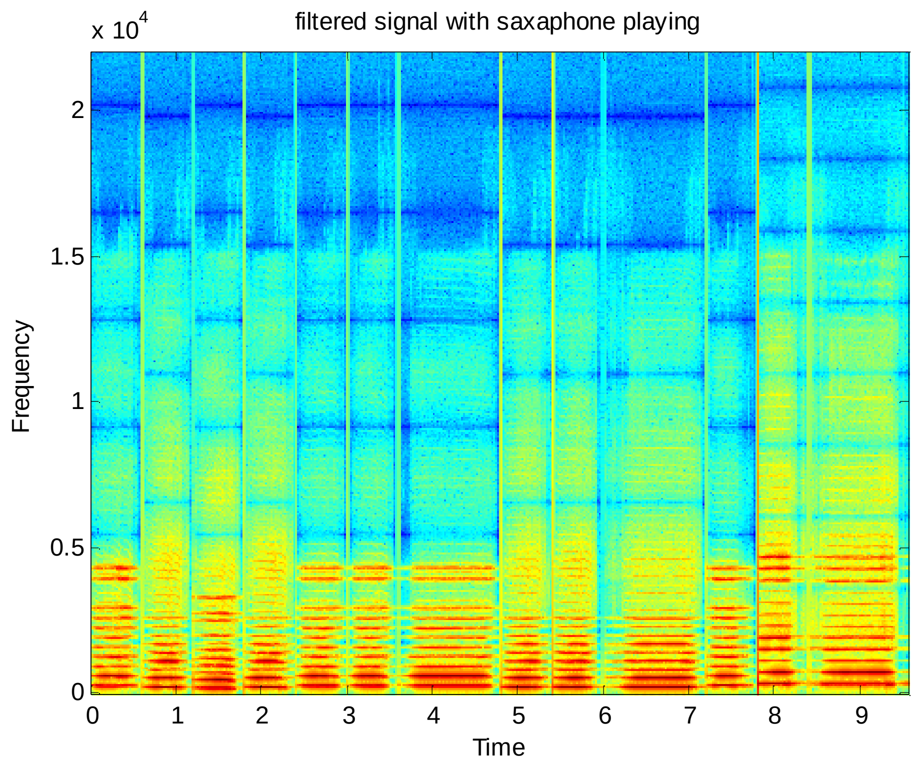
Analysis of result

The filtered result of the flute and saxophone sounded good, and we checked the results using the spectrogram. The following two are spectrogram of the filtered flute and the filtered saxophone versions of Mary Had a Little Lamb. The spectrograms shows for the flute, that 4-6 frequencies were filtered depending on the note(refer to table of frequencies above), and the saxophone, there were 8-11 filters.

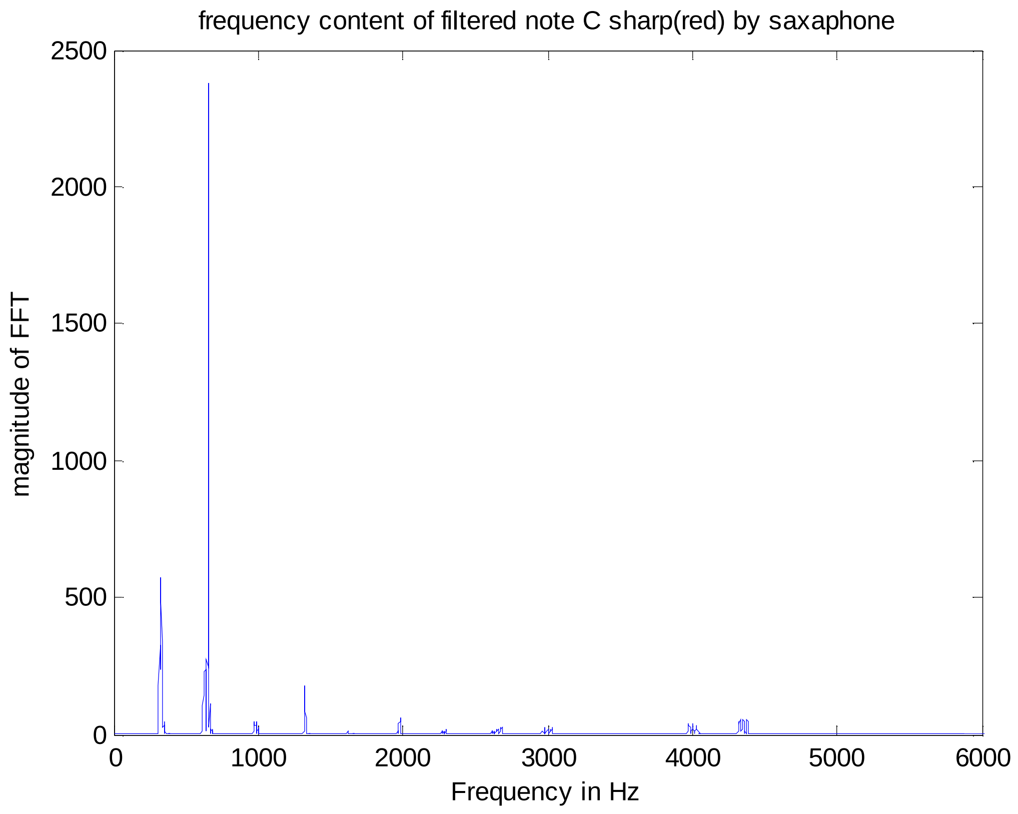
Some filters worked better than others because our filters had a bandwidth of 100 Hz and we designed it to vary at 50Hz intervals. This made the filtered frequencies overlap with actual frequencies better for some notes, and worse for others.

The sharp vertical lines that appear in the spectrogram in between each note represent the boundary of each note that is processed by different filters.


filtered signal with flute playing
x 10^4
Frequency
Time
0
0.5
1
1.5
2
0 1 2 3 4 5 6 7 8 9

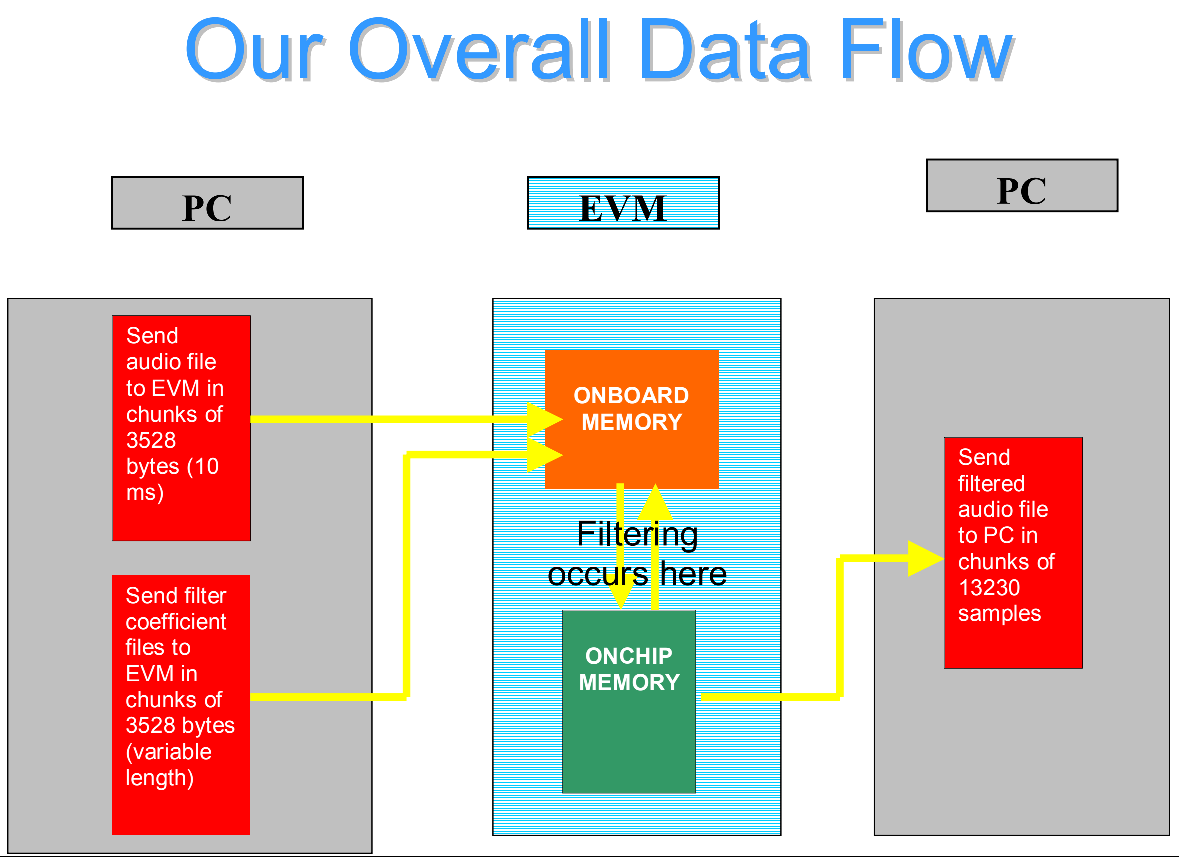

filtered signal with saxaphone playing
x 10^4
Frequency
Time
0
0.5
1
1.5
2
0 1 2 3 4 5 6 7 8 9

The filters that we used had a gain of 1.7 for all the frequencies used. Therefore for the frequencies that overlapped between the flute and the saxophone, most of the combined signal remained in the filtered flute and filtered saxophone signal. This was especially true for the 2nd and 4th harmonic of the saxophone, which corresponded to the fundamental and 3rd harmonic for the flute. The graph below shows that for these two frequencies, the filtered saxophone signal contained much of the signal from the flute at those frequencies. In future work on this project, we will try to find a way to scale the filters of the harmonics at set percentage gains compared to the fundamental so that the relative amplitudes are the same as the original signal.



We could also have implemented another filter design, possibly with IIR filters because it has better performance with fewer coefficients so that less computation is required. Another possibility is to use Professor Stern's comb filter, which might also reduce the amount of computation required.

If we were to continue this project in the future, we would explore using the Panning Algorithm because that shows much promise for success when implementing the project in the frequency domain, rather than in the time domain like we did. Also, time constraints limited us from fully optimizing our filtering and EVM transfer code. However, our results show that the goal of source separation can be achieved reliably. Future attempts at this would include thorough testing on a wider variety of instruments, with varying panning indexes, amplitudes, and frequency ranges. While our project achieves successful results with two instruments, this is only a stepping stone to achieving a full-fledged implementation of audio source separation.


Our Overall Data Flow
PC
EVM
PC
Send audio file to EVM in chunks of 3528 bytes (10 ms)
Send filter coefficient files to EVM in chunks of 3528 bytes (variable length)
ONBOARD MEMORY
Filtering occurs here
ONCHIP MEMORY
Send filtered audio file to PC in chunks of 13230 samples

References:

[1] Acoustical Society of America - *Computer Identification of Musical Woodwind Instruments*
http://www.acoustics.org/press/139th/brown.htm
(analysis of frequency spectra of saxophone vs. flute)

[2]. The Wavelet Tutorial- By Robi Polikar
http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html
(useful for understanding of wavelet transforms)

[3]. Frequencies of a equal tempered scale
http://www.phy.mtu.edu/~suits/notefreqs.html
(provides musical frequency chart for conversion of notes)

[4] *Frequency-Domain Source Identification and Manipulation in Stereo Mixes for Enhancement, Suppression and Re-panning Applications*- Carlos Avendano
Creative Advanced Technology Center- 2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, October 19-22, 2003, New Paltz, NY
(Introduces and explains the algorithm of using the spatial panning index to identify and isolate instruments)

[5] The Georgia Tech College of Computing
http://coweb.cc.gatech.edu/csl/66
(explanation of the STFT)

[6] *Toward Automatic Transcription - Pitch Tracking In Polyphonic Environment* - Literature survey by Keerthi C. Nagaraj, March 2003
http://www.ece.utexas.edu/~bevans/courses/ee381k/projects/spring03/nagaraj/LitSurveyReport.pdf
(explanation of the CQT)

[7] *Audio Analysis using the Discrete Wavelet Transform* - George Tzanetakis, Georg Essl, Perry Cook
http://www.cs.princeton.edu/~gessl/papers/amta2001
(explanation of DWT)

[8] Fundamental Frequency Estimation technique
http://www.cs.tut.fi/sgn/arg/klap/multiplef0.pdf
(explanation of the FFE)

Other references (not included in text):
18-551 Class notes- Z-Transforms (ee551.3.125.pdf)- David Casasent, Spring 2005
(Very helpful when choosing a filter design)

Comb Filter for Building Reverberators by Scott Lehman
http://www.harmony-central.com/Effects/Articles/Reverb/

(Source contains C code for building a comb filter)