Objective and Description

The goal of this project is to estimate the notes present in an audio file of a monophonic musical performance. This is accomplished using autocorrelation to find the existing fundamental frequency of each uniform block of time in the audio file and binary search to find the nearest pitch in western tuning, using A = 440 Hz. The detected pitches are reported back as pitch class and octave.

Limitations

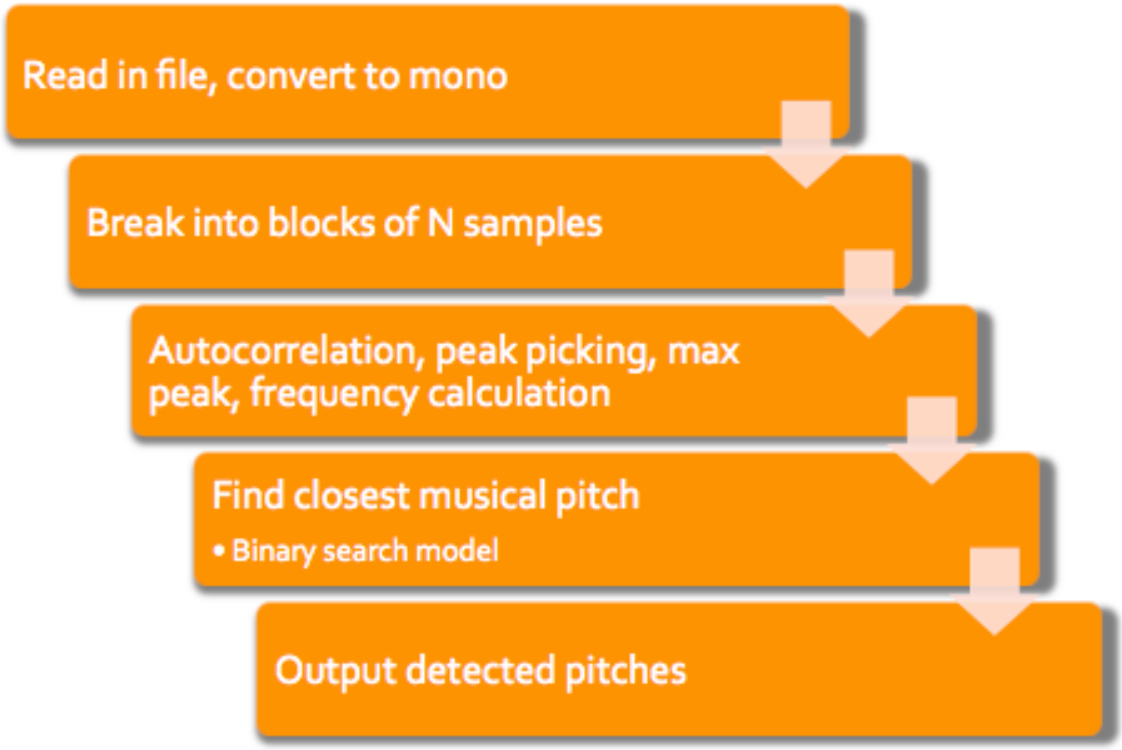
The scope of the project is limited to detecting monophonic pitch, as much more intensive methods are required for estimation of polyphonic pitch and would likely fall outside the time allotted for project completion. As currently configured, the program assumes pitch information in a multi-channel monophonic signal will be the same in all channels and as such operates on only the first channel. Because of the complexities of implementing the fast Fourier transform and its inverse in C (and the inability to get pre-existing code working in the context of this program), time-domain autocorrelation is used, which is a $O(n^2)$ process. As such, the program is recommended for shorter signals at this time.

Process

1. ### Parse command line, read in audio files and convert to mono.
   The file name to be opened is passed from the command line and opened using the sf_read() function of the libsndfile library (#include <sndfile.h>). The function loadBuffer() loads the first channel into a buffer for processing.
2. ### Run autocorrelation function.
   The function autocorr() in calculations.c splits the signal into numBlocks blocks of size N. N is a precompiler constant; numBlocks is calculated by dividing the number of frames in the signal by N. Each block is pre-processed by normalizing it using the normalize() function. Autocorrelation is then run on each block and the result is stored as a row in the two dimensional array rxx[][].
3. ### Pick peaks and find the maximum peak in each block.
   For each block of length N in rxx[][], the function pickPeaks() in calculations.c find the indices of all elements *n* such that $x[n-1] < n > x[n+1]$. The index of the maximum peak in each block is stored in the array maxPeaks[]. This index represents the lag in samples that produces the strongest correlation.
4. ### Find the closest note in the scale to the detected pitch.
   The function findNotes() defined in noteDef.c uses the structure Note defined in noteDef.h as struct tagNote{ int octave; float frequency; char * name }; to create an array of all possible notes. initNote() then fills the elements of the array with the fundamental frequency, pitch class, and octave of each note. Then for each detected frequency, getName() uses a binary search (which is not intended to actually find its target) to get a value within one half step of the closest note to the detected fundamental. The found note is compared to the note before and after it in the array of all notes, and the closest value to the detected frequency is returned as the estimated pitch.
5. ### Filter results.
   During the peak picking function, the power of the max peak in each block in dB is recorded in

the array rxxPower[]. This array is also passed to findNotes() and is used to filter notes whose level of correlation falls below the power threshold set by MIN_DB. Notes also must fall below MIN_PITCH and MAX_PITCH in order to be recorded, to eliminate errors caused by silence in the performance. Finally, notes that are detected in multiple consecutive blocks are reported only once by saving the previous pitch as a reference.

Block Diagram



Time Complexity

Due to the use of time domain autocorrelation, the time complexity is $O(n^2)$. The binary search adds $O(\lg n)$ time, but the additional time is insignificant when added to n^2.

Implementation

The program was written using C99 on OS X 10.9.5. The command line to compile the program is:

```
gcc main.c calculations.c noteDef.c –L/usr/local/lib –lsndfile –
I/usr/local/include –o pitch
```

.