Project Report

Music Auto-Transcription on Android Platforms

Qiuyuan Zhang

Hyunjun Hong

1. Introduction

1.1 Project Description

The final project for ECE420-Embedded Digital Signal Processing Lab is an android app that processes sequences of single note of instrumental sound into music score and displays them on the screen.

1.2 Market

This app is targeting toward the amateur musicians and composers who might find it time consuming or difficult to translate a piece of music into music sheet. Our application serves as a useful tool that allows professional and amateur musicians to allocate less time on transforming music into score while have more time on practicing and composing new art.

1.3 Constrains

Working with the time frame of the course, roughly one and half month, we decide to set realistic limitations on our project.

1.3.1 Range of acceptable frequencies

For the purpose of both eliminating octave error and displaying the note aesthetically, we choose to limit the range of acceptable notes from 110Hz to 880 Hz, namely Low A to High A. If the detected frequency is lower or higher than the acceptable range, the note will not be displayed.

1.3.2 Range duration recognition

Considering the processing ability of the android device, we decide to differentiate only two types of note, quarter note and half note. If a note falls below quarter note length, it will be displayed as a quarter note. If the note, on the other hand, falls above half note duration, we would keep it as half note.

1.3.3 Specific Instrument

While designing our application, we realize many parts of algorithm have to be hand tuned toward the specific instrument. For the final project, we geared our program toward piano sounds only, and used the IOS app called Piano Free with Songs, a virtual piano, for testing.

2. Design Work Flow

Through the 420 course, we learned a standard process of turning ideas into applications. The steps are as follows:

1) Choose the algorithm of interest
2) Model the algorithm in Mat lab or Python and analyze result

   If not satisfied, repeat step 1 and 2
3) Have a block diagram of the implementation
4) Implement the desired algorithm into digital signal processing chip
5) Debug
6) Optimization

2.1 Pitch Detection Algorithm – Cepstral

Our project aims to build a tone detector for sound in audible range generated by instruments, then output a score accordingly. For the specific requirements of our project, we chose to implement the Cepstral method with implementation methods introduced as a block diagram shown below.

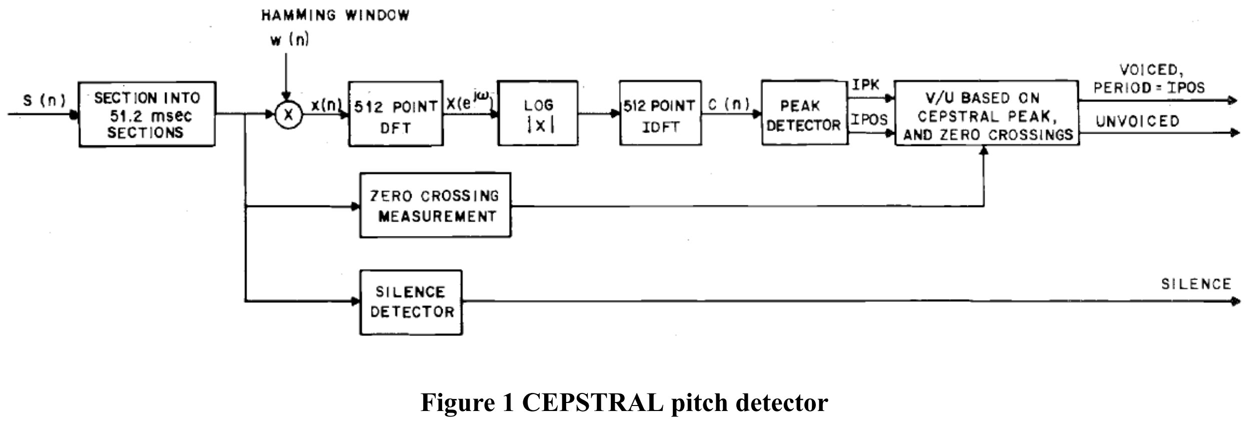


Figure 1 CEPSTRAL pitch detector

We have modeled this algorithm with Matlab. With an input of recorded piano note with 44.1k sampling frequency, we are able to detect the note frequency accordingly. The graphic output and specific contributions of each block is explained as follows.

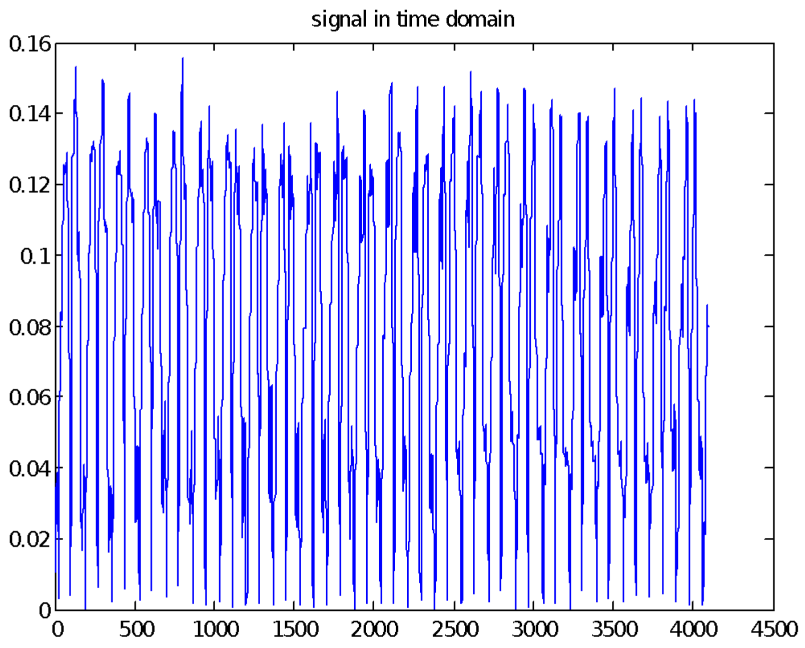


Figure 2 Time domain of piano note C

The algorithm introduced by the paper takes in the incoming data stream and separated it into frames of 512 samples, each 51.2 millisecond, is weighted by a 512 point Hamming window. Hamming window effectively reduces the nearest side lobe, which works well for identifying closely spaced sine waves, in our case, different piano sound frequencies. While implementing this method, we realize the result would have better result if we increase the sampling point. Therefore we increased our frame size to 4096 data points, and so did the hamming window.

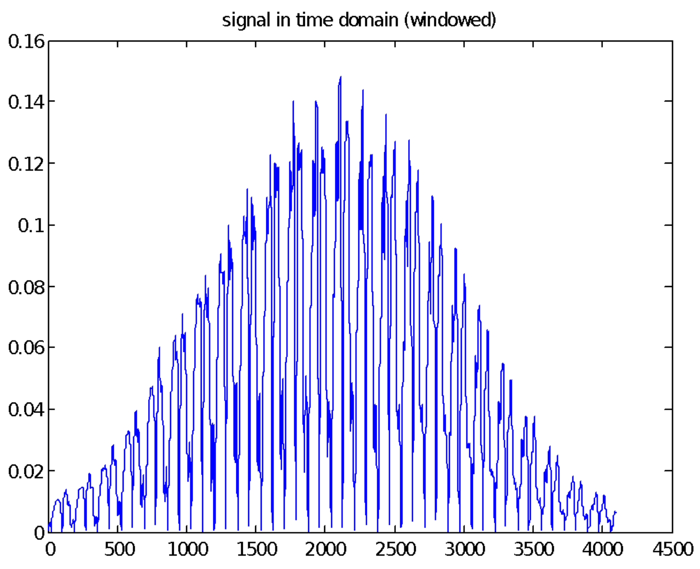


Figure 3 512 point hamming windowed signal

We take the DFT of the frame of 4096 sample points to transform data from time to frequency domain.

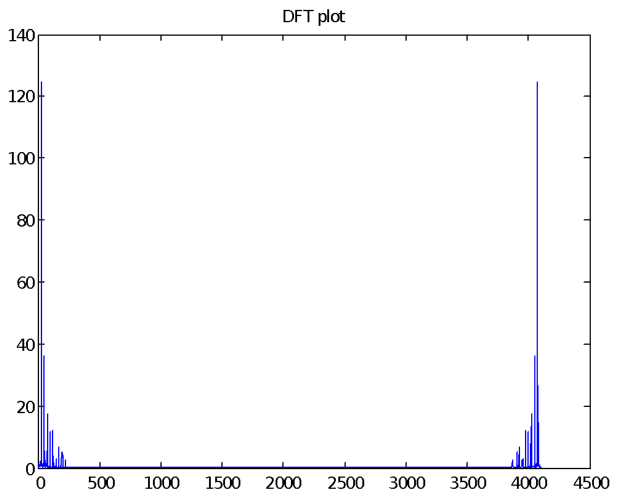


Figure 4 DFT of windowed signal

Then, we take the log of the absolute value of the DFT data to raise the resolution on the less prominent frequency peaks.

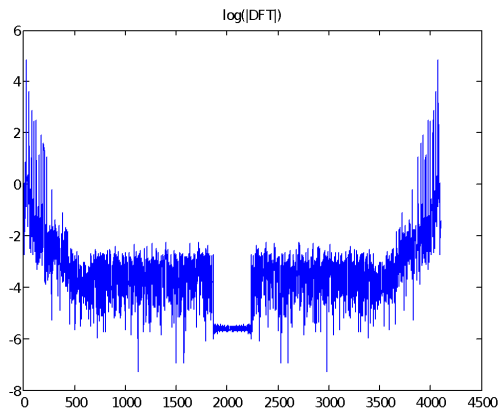


Figure 5 Log of abs of DFT signal

After successfully generating recognizable frequencies into prominent peaks in the frequency graph, we transform these data back to time domain using IDFT. Each peak in the IDFT domain corresponds to a single frequency, where the location of these peaks represents their periods.

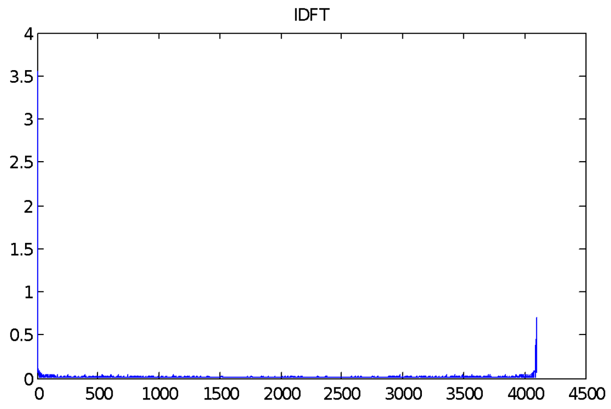


Figure 6 IDFT signal

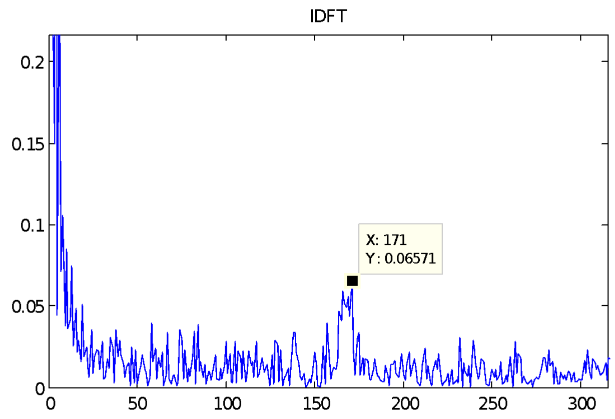


Figure 7 Zoomed in version of IDFT

To extract the frequency of the note, we take the location of the prominent peak as its period in the 92.9ms frame (which is calculated by: number of samples x 1/sampling frequency). Divide the sampling frequency 44.1 KHz by the extracted period is the true frequency of this note. Then we run a search through all the standard piano key frequencies provided, ranging from Low C to High C, to find the closest note.

For example, the C note has a prominent peak at 171 on the IDFT graph: 44.1k/171 = 257.89 Hz , very close to the provided C note frequency at 261.626 Hz. Then we can decide that it is indeed mid C. We created a look-up table for different pitches.

To optimize computation power, we also implemented the silence detector algorithm. When the amplitude of samples points in a frame does not exceed a threshold value, we would classify them as a silence sample before Cepstral computation.

2.2 Pitch Detection Algorithm – Autocorrelation

The pitch detection algorithm we used to determine the pitch of the signal follows the method described in the paper by Rabiner, Cheng, Rosenberg, McGonegal (L.M. Rabiner, MJ Cheng A.E. Rosenberg, C.A. McGonegal, "A comparative performance study of several pitch detection algprithms". IEEE ASSP, Vol.24, pp. 399-4 18, 1976 [1]). However, we made the algorithm simpler since the original algorithm is designed for the human voice and our project was about musical instruments. For example, the last section of the algorithm determines whether the voice signal represented in the current sample is a voiced sound or a unvoiced sound. We did not need this particular feature so we did not include it in the code. The block diagram outlining the algorithm is shown below.

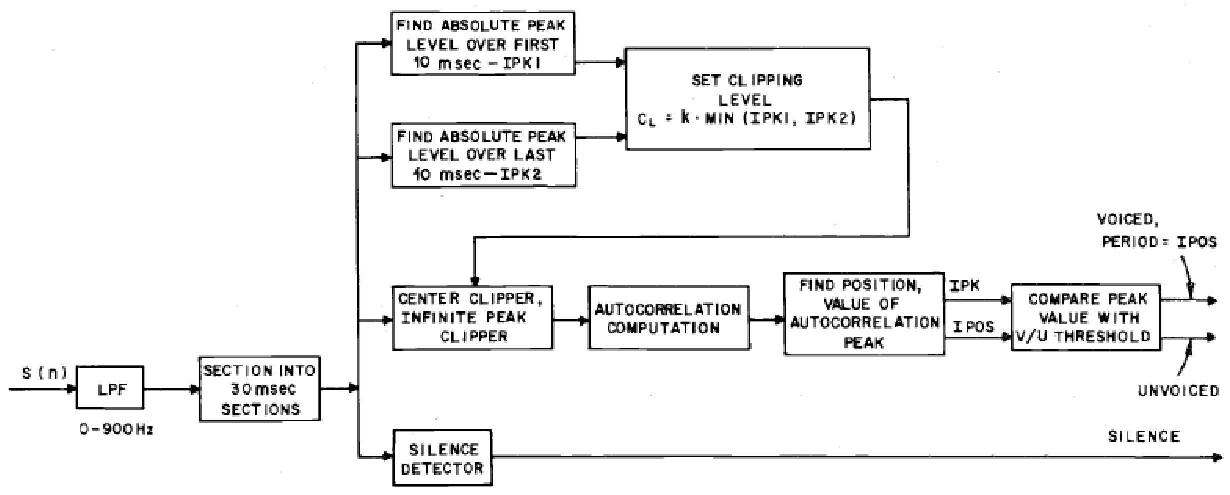

FIND ABSOLUTE PEAK LEVEL OVER FIRST 10 msec – IPK1
FIND ABSOLUTE PEAK LEVEL OVER LAST 10 msec – IPK2
SET CLIPPING LEVEL $C_L = k \cdot MIN(IPK1, IPK2)$
VOICED, PERIOD = IPOS
CENTER CLIPPER, INFINITE PEAK CLIPPER
AUTOCORRELATION COMPUTATION
FIND POSITION, VALUE OF AUTOCORRELATION PEAK
IPK
IPOS
COMPARE PEAK VALUE WITH V/U THRESHOLD
s(n)
LPF
0-900Hz
SECTION INTO 30msec SECTIONS
UNVOICED
SILENCE
SILENCE DETECTOR

Figure 8: Block diagram of the Modified Autocorrelation Method (from [1])

Our code breaks down the algorithm in six parts described below:

2.2.1. Separating the signal into blocks

Given a signal, it is necessary to select a section to determine to pitch of. We chose to use 1024 samples, which corresponds to 23ms worth of sound signal, to process each time the function is called.

2.2.2. Silence detection

Using the maximum amplitude of the selected block, we first determine whether if a sound is detected or not. If the maximum amplitude does not go over the threshold (empirically determined) we simply do not process the section and go on to the next section.

2.2.3. Determining the clipping level

Center-clipping is needed for more effective computation of autocorrelation. It also removes the possibility of the calculation error due to the unusually weighted samples. To determine the clipping level, the peak of the given section is divided in three, 1/3 of the 1024 samples each. The maximum from the first and third section is collected, which is multiplied by 0.64 (64%) to obtain the clipping level.

2.2.4. Center-clipping

Once the clipping level is determined, each sample from the section is compared to the level. If the sample is above the clipping level, it is assigned the value 1. If the sample is below the negative of the clipping level, it is then assigned to -1. Otherwise, it is assigned to 0. After center-clipping the samples take only one of the three values, which are -1,0, and 1.

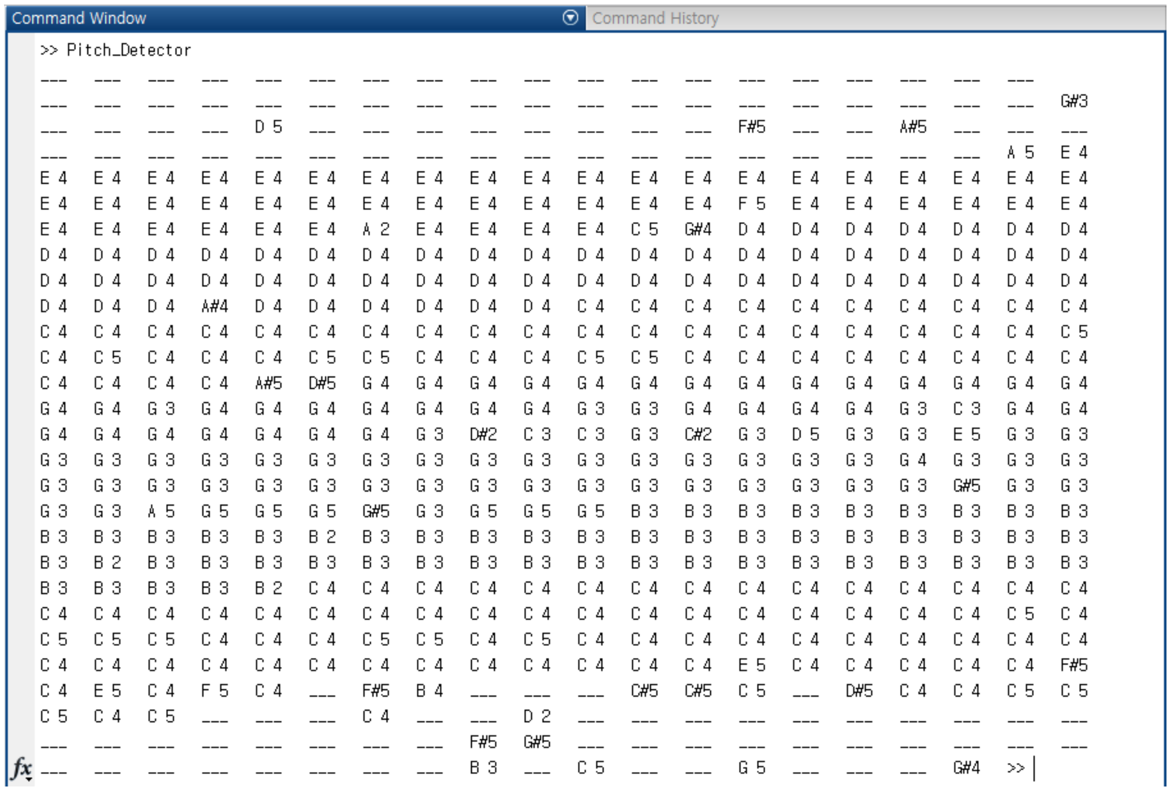
2.2.5. Autocorrelation operation

With the center-clipped samples, autocorrelation operation is carried out. In particular, all the samples outside the current section are assumed to be 0.

2.2.6. Determining the location of the peak and the related pitch

After the autocorrelation operation the location of the peak among the output is determined. The location corresponds to the period of the detected signal (in digital sense). To figure out the actual pitch, the sampling frequency (44.1kHz) is divided by the location.

We first tested the code in MATLAB with a recorded sound sample to test the functionality and the accuracy of the algorithm. We added a feature to print out the piano key corresponding to the frequency detected by looking up the name table. The result was very successful. It detected the frequency of the signal with around $\pm\pm$2 Hz error. Upon processing a sound recording of myself pressing the piano keys in the sequence of E-D-C-G(high)-G(low)-B-C, our code produced the result shown below.



```
>> Pitch_Detector
---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---
---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   G#3
---   ---   ---   ---   D 5   ---   ---   ---   ---   ---   ---   ---   ---   F#5   ---   ---   A#5   ---   ---   ---
---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---   A 5   E 4
E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4
E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   E 4   F 5   E 4   E 4   E 4   E 4   E 4   E 4
E 4   E 4   E 4   E 4   E 4   E 4   A 2   E 4   E 4   E 4   E 4   C 5   G#4   D 4   D 4   D 4   D 4   D 4   D 4   D 4
D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4
D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4   D 4
D 4   D 4   D 4   A#4   D 4   D 4   D 4   D 4   D 4   D 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4
C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 5
C 4   C 5   C 4   C 4   C 4   C 5   C 5   C 4   C 4   C 4   C 5   C 5   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4
C 4   C 4   C 4   C 4   A#5   D#5   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 4
G 4   G 4   G 3   G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 3   G 3   G 4   G 4   G 4   G 4   G 3   C 3   G 4   G 4
G 4   G 4   G 4   G 4   G 4   G 4   G 4   G 3   D#2   C 3   C 3   G 3   C#2   G 3   D 5   G 3   G 3   E 5   G 3   G 3
G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 4   G 3   G 3   G 3
G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G 3   G#5   G 3   G 3
G 3   G 3   A 5   G 5   G 5   G 5   G#5   G 3   G 5   G 5   G 5   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3
B 3   B 3   B 3   B 3   B 3   B 2   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3
B 3   B 2   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3   B 3
B 3   B 3   B 3   B 3   B 2   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4
C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 5   C 4
C 5   C 5   C 5   C 4   C 4   C 4   C 5   C 5   C 4   C 5   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4
C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   C 4   E 5   C 4   C 4   C 4   C 4   C 4   F#5
C 4   E 5   C 4   F 5   C 4   ---   F#5   B 4   ---   ---   ---   C#5   C#5   C 5   ---   D#5   C 4   C 4   C 5   C 5
C 5   C 4   C 5   ---   ---   ---   C 4   ---   ---   D 2   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---
---   ---   ---   ---   ---   ---   ---   ---   F#5   G#5   ---   ---   ---   ---   ---   ---   ---   ---   ---   ---
---   ---   ---   ---   ---   ---   ---   ---   B 3   ---   C 5   ---   ---   G 5   ---   ---   ---   G#4   >>
```

Figure 9: MATLAB output of the pitch detection code

After the successful implementation in MATLAB, the code was re-written in C to be used in Eclipse. "process.c" given in lab 4 was altered for our use. Before the actual application was made, we made another application which determines which piano key was pressed in real-time. After some calibrations, we succeeded in making it. A photo during the demo is shown below.

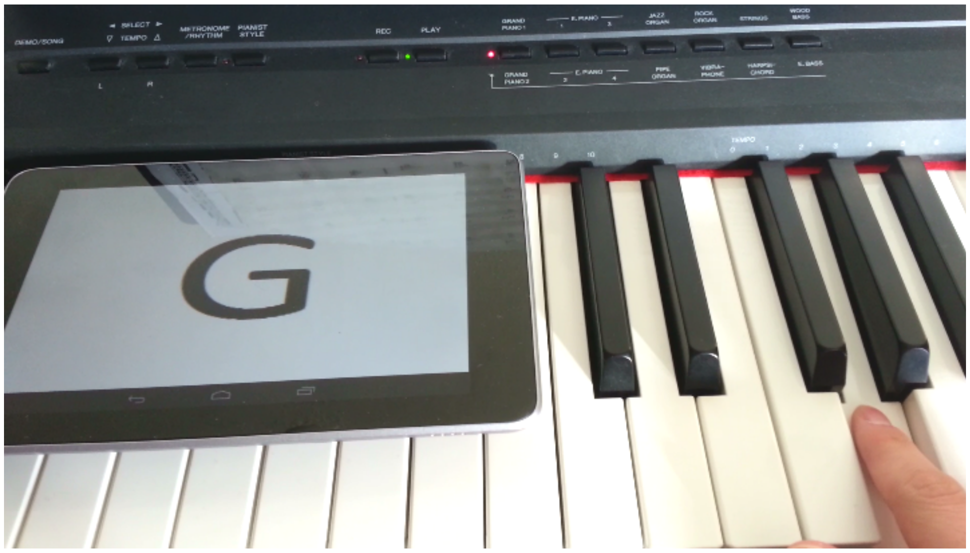


Figure 10: Pitch detection algorithm demo, detecting "G"

Through testing, we realized that Autocorrelation method is more reliable compare to the Cepstral method; therefore choose it as the desired algorithm for implementation. Then we come up with the block diagram of the overall implementation of our final project is shown below.

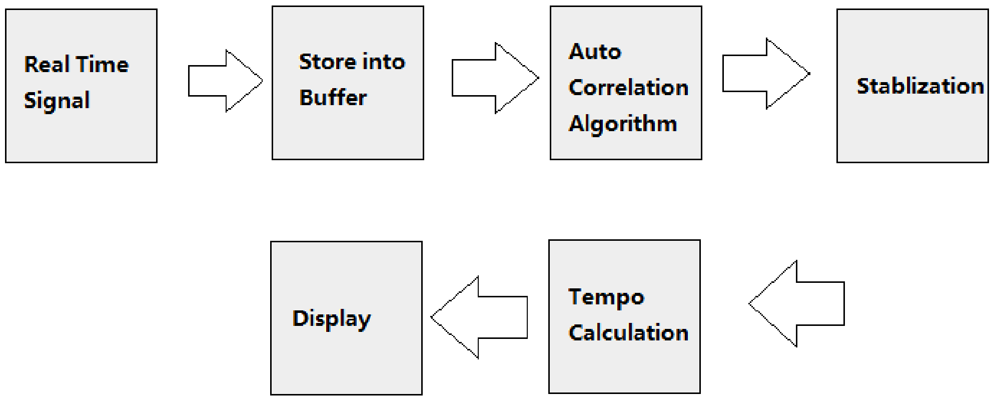


Figure 11: Block Diagram of Overall Implementation

2.3 Stabilization

Due to the over sensitivity of the Auto-Correlation algorithm, the outputted note is not stable. Instead, it flickers between octaves and nearby notes. To increase stability, we implemented a software denouncer, where a note will only be displayed if the same note is outputted consecutively. At this point, we are able to have an accurate recognition of piano notes.

2.4 Display

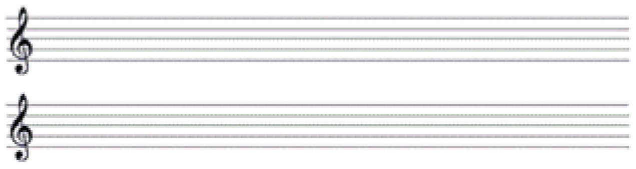


Figure 12: Example of Android Display

Though not required by the class, we decide to perfect our project by adding user friendly display system. We value the GUI as an important part of user experience and believe it will affect the ratings of our app directly.

2.4.1 Note

We made our displays with GIMP, a free photo editing software. Each picture is created with 85 by 85 pixel size and we allowed 8 notes to be displayed by row and there are two rows in total. Each part of the frame is given a position while each note picture is given a name. We then create a function that updates new note in new position accordingly. Position vector is incremented when there is a new note or duration of silence in between notes. The position vector wraps around once position reaches outside of the screen. The new note always starts with a quarter notes. The note updates at the same position if it is a half note instead of a quarter note.

2.4.2 Tempo

With limited time, we decide to use programmer defined tempo instead of a user defined one. We hand tuned a tempo by setting threshold on quarter and half note recognition based on the number of repeats of the same note before transitioning to a new note.

2.5 Optimization

The speed of note recognition dropped significantly after we implemented the several loops for display and tempo. We deleted unnecessary repetitions in the code and added a metronome output in both sound and visual sense, to guide the user with the preset relationship between note and silences in order for a quarter notes to be recognized. For example, the user should start a note at beat 1 and have a silence at beat 0.

3. Future Goals

If we could have more time to perfect our program, we would have added more features that increase the reliability and usability of the application.

3.1 Volume

The autocorrelation method we implemented requires a loud sound input to be reliable. To solve this problem, we would like to add a *dynamic range compressor* block right before running the pitch detection. Using proper attack and release threshold, the device should be able to pick up relatively lower volume sound input as well.

3.2 Tempo Accuracy

Instead of setting tempo by experimental value, we would like to use the *countdown timer* in the android platform to make an interrupt every time a quarter notes are reached. We could also allow the user to set the timer threshold to realize a user chosen tempo.

3.3 Chord

We would like to also add the ability to recognize chords; we may need to find better algorithms for such purpose.

3.4 Display and Save

The application should be more user-friendly and hopefully having an auto scrolling ability when number of notes is outside of the display range. The music sheet should be outputted in formats of widely used score reader applications.

4. Software and Hardware Documentation

- Asus/Google Nexus 7

- Android Jellybean 4.1.2.

- NVDIA Tegra Pack, version NDK r8e.

References:

[1] RABINER, LAWRENCE R., MICHAEL J. CHENG, AARON E. ROSENBERG, and D CAROL A. McGONEGA. "A Comparative Performance Study of Several Pitch Detection Algorithms." *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING* ASSP-24.NO. 5 (OCTOBER 1976): 399-418.