Music Coach: Real-time Evaluation of Music Performance using Nokia N900

David Johnson
Department of Computer Science
UCSB

djohnson@cs.ucsb.edu

Dianna Han
Department of Computer Science
UCSB

dianna@cs.ucsb.edu

ABSTRACT

In this paper, we describe the design and implementation of the Music Coach application that runs on the Nokia N900 mobile phone platform. The application plays the role of a music coach which listens to the user's musical performance and provides real-time feedback on both timing and pitch accuracy. It also utilizes the accelerometer in the mobile phone to detect and update tempo set by the musician during performance. Due to the limit on computational power, the main challenges of this project lie in real-time processing of audio input/output and visual feedback. Other challenges include light-weight accelerometer reading processing and accurate pitch recognition. We discuss these technical difficulties in detail and present our approaches to resolve these issues. Our final results show that the Music Coach application is both easy to use and helpful for entry-level musicians to improve their skills.

Keywords

Mobile phone application, real-time audio processing, pitch analysis, multi-modality input/output, real-time visual feedback, accelerometer.

1. INTRODUCTION

Smartphones, or mobile phones with computational power approaching that of PC's and a wide range of integrated sensing capabilities, are gaining popularity. Smartphones today outperform desktop computers from ten years ago in terms of processor speed, memory, and disk space; moreover, they have far more functionality than traditional desktop or laptop computers with additional items such as a camera, touch screen, and accelerometer. For example, Apple iPhone 3GS, one of the most popular and most successful mobile phones in the market, is equipped not only with GPS and accelerometer but also with proximity sensor and ambient light sensor.

In response to such a trend, numerous applications have been migrated to smartphone platforms, and a variety of applications are being designed and developed specifically for smartphones, for both entertainment and practical usage. In this paper we propose a new application called Music Coach that utilizes the built-in multi-modality capabilities and the portability of a smartphone to provide both musical training and entertainment to the user.

Music Coach, as the name indicates, listens to the user's musical performance and provides real-time feedback on both timing and pitch accuracy. With a pre-loaded music score, the application shows the user a sequence of notes to play with timing information, then checks if the notes are played at the correct pitch and time. The application can also evaluate pitch without pre-loaded music score by checking how close a note played is to its nearest discrete note. The application is valuable for users from beginners to advanced musicians and for higher range instruments categories. The exclusion of low range instruments such as double-bass is due to the long sample window needed to accurately determine pitch for these low frequencies. It is especially useful for pitch evaluation of instruments that do not have discrete notes such as cello, violin, trombone, or even human voice. Feedback is provided to the user both real-time and offline: alerts that inform the user of pitch and rhythm inaccuracies, and overall evaluation at the end of the performance.

As an additional feature, Music Coach also detects tempo set by the user shaking or rocking their smartphone. Audio alerts similar to a metronome will be played to the user during the performance in accordance with the current tempo setting. This allows the user to change the pace of music without explicitly specifying a number to the application.

Music Coach is currently developed on Nokia N900 smartphone platform, which is using Maemo 5 operating system and equipped with microphone, speakers, touch-screen, and accelerometers.

In the following sections we will cover details of the application. Section 2 gives an overview on the background of music recognition applications and techniques. Section 3, 4, 5, and 6 analyze the application requirements, discuss major challenges, and present our approaches, solution and results. Section 7 summaries the work and presents future improvements.

2. BACKGROUND

The idea of designing mobile phone applications that assist the user in their musical activities is not foreign to us. On Apple iPhone, GuitarToolkit[1] provides essential guitar utilities, including an amazingly accurate tuner and a library of over 500,000 chords; TyroTuner[2] is another microphone-based application specifically tailored to let the user tune a standard 6-string guitar. Besides utilizing audio input/output, other applications such as ZOOZBeat[3] also relies on accelerometer to detect user gestures and movements to enable easy composing and remixing music by shaking and tapping the phone.

Furthermore, applications on a non-mobile platform that provide feedback on user performance are also familiar to us. A widely known example would be a Karaoke system, which evaluates the user at the end of a song by giving a score.

In the Music Coach project we propose to combine these two categories of applications mentioned above by providing real-time feedback to musicians using a mobile phone platform which can change its tempo using the accelerometer. Music Coach aims more at musicians who need to evaluate their performance on real instruments. Such applications are not yet available to our knowledge.

3. REQUIREMENT ANALYSIS

For a system whose main task is to recognize pitch and timing accuracy of a musical performance done by instruments or via human voice, relevant requirements will include pitch range, frequency accuracy/resolution, sampling rate, and processing time.

3.1 Pitch Range

A pitch recognition system designed to work for all instruments and human voice would have to cover a wide range of frequencies from 20Hz to 4186Hz. Figure 3.1 shows the frequencies of all the white notes on a keyboard and the ranges of some selected instruments as well as human voice for equal temperament tuning. Note that names of a keyboard scale are shown as A0 to C8, where the letter represents the name of the note and the number represents the octave. If a note sounds an octave higher than another, its frequency doubles accordingly.

For most western music the tuning system follows an equal temperament system in which adjacent notes in a scale are all separated by logarithmically equal distances. Since this scale divides an octave into twelve equal-ratio steps and an octave has a frequency ratio of two, the frequency ratio between adjacent notes is then the twelfth root of two ($2^{\frac{1}{12}}$, or ~1.05946309). Tuning allows music to sound the same in any key. It enabled Bach to compose his well-tampered clavier in all 24 major and minor keys for harpsichords, which he tuned himself to an equal tempered scale. This was at the time when most of the instruments were using tunings that didn't allow them to play in any key.

As will be explained in following sections, this actually introduces a certain degree of complexity as optimal settings in the pitch recognition system are different for different instruments ranges.

3.2 Pitch Accuracy/Resolution

The accuracy requirements depend mainly on the pitch spacing between adjacent notes. As mentioned before, the pitch spacing between adjacent notes is logarithmic rather than linear. This means that the frequency difference between two adjacent notes of low frequencies will be smaller than that of high frequencies, which leads to higher accuracy requirements for pitch detection. For a particular instrument, its accuracy requirement will always be set to the frequency interval between its lowest two notes. This would resolve a detected frequency to its closet discrete note frequency. Consequently, if feedback on pitch accuracy is required at a higher resolution than adjacent notes in a 12-tone scale, a smaller frequency interval is required.

For example, an alto recorder's lowest frequency interval is 20Hz between F4 and F#4. Therefore the required accuracy of the system in order to detect all musical notes in range would be 20Hz, although towards the top end of the range the highest frequency interval is 171Hz. This means that 4 divisions of accuracy could be defined at its top end but only 1 division of accuracy at its bottom end.

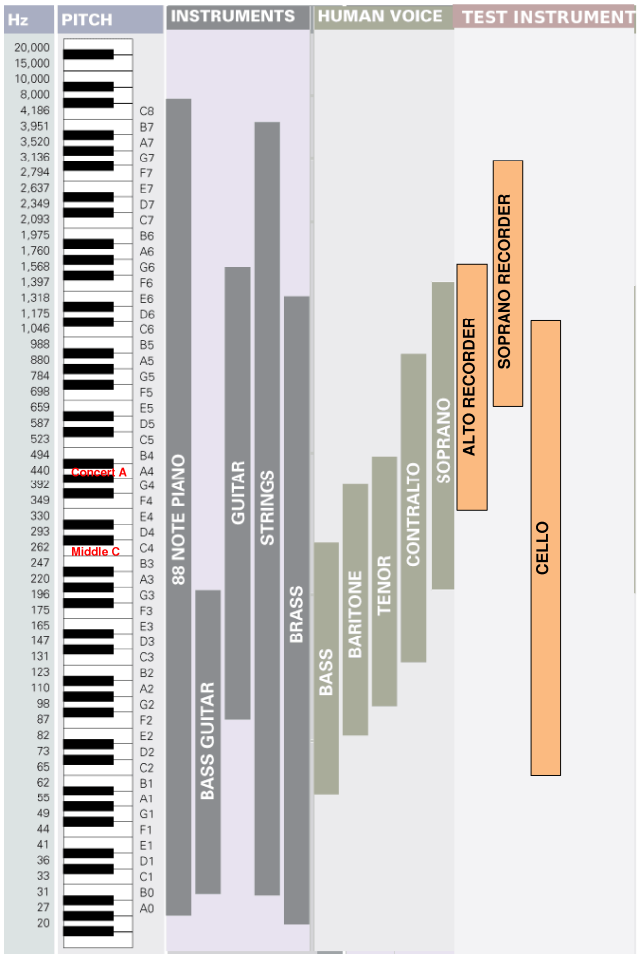

Figure 3.1. Frequency Map for Instruments. The frequencies of all the white notes on a keyboard are shown in this map along with the frequency range of some example instruments.

if a set of 3 discrete zones are required for pitch analysis, which specifies if the pitch is sharp (above the note), on the note, or flat (below the note), then a sampling window should be chosen to produce a frequency resolution of 20/3 ~ 6.5Hz for the alto recorder, for example. Even though further divisions are possible at the higher end of its range, the system will be designed around the finest grained feedback on the bottom end of the scale for the sake of uniformity.

3.3 Pitch Sample Rate

Pitch sample rate specifies how often the pitch of the audio signal is sampled. Feedback frequency about pitch accuracy cannot be faster than the sample rate as the system needs to first analyze the current set of samples to determine the frequency. Requirements for the sample rate are determined by the shortest note duration expected in the performance as well as the lowest expected frequency. The relationship between sample rate and frequency will be explained in following sections. The Music Coach application uses approximately 20Hz as its sample rate.

3.4 Processing Time

Processing time is determined by the computational overhead of the application and the capability of the device. To facilitate real-time feedback to the user the processing time to carry out pitch, timing, and tempo detection should be made as short as possible. The majority of the computational load is introduced by the pitch detection system; thus the FFT thread that carries out this task was assigned the highest priority. The Nokia N900 was able to carry out an FFT on a 100ms sample of data in 3ms. In this case feedback will be delivered to the user 103ms after the note started playing, which is an acceptable and reasonable delay.

3.5 Tempo Detection

The fact that motion of the phone is reflected in accelerometer measurements leads to our proposal of using the phone as a tempo detector. When the user shakes or rocks the phone periodically, the application detects the period of such movements and translates it into tempo.

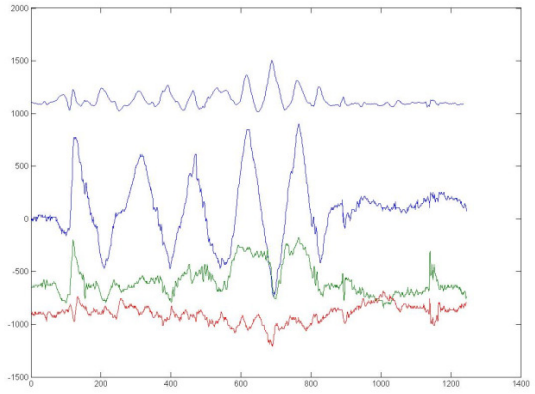

Figure 3.2. Accelerometer readings when the user is moving the phone back and forth. The top line is the plot of the vector amplitude of x, y, z readings. The next three lines are x, y, z direction readings respectively.

Figure 3.2 shows the typical accelerometer readings when the user is moving the phone in a rhythmic manner. We can see clearly from the plot that the period of repetitive motion is reflected in the accelerometer readings as the time difference between two peak readings. Since the phone can be shaken or rocked at any direction, the vector amplitude would be a reasonable measure to use. In this way, the tempo detection problem can be translated into a problem of finding peaks in a digital signal.

Many peak detection methods and algorithms have been developed and proposed in signal processing. However, most of such algorithms are unsuitable for our application because of the requirements of real-time and minimal lag. The ideal solution would be an algorithm that does not need a large buffer to build statistical models but processes data on the fly; the algorithm should be robust and reasonably accurate, while the computational cost should be minimized.

4. SYSTEM ARCHITECTURE

The software design for the system follows a threading model in which components that needed to run concurrently are executed in separated threads. These included (a) recording sound to a buffer, (b) analyzing the sound with an FFT, and (c) carrying out analysis of accelerometer input. There are also timer modules, such as (a) note progress timer and (b) metronome timer, which control the progress of notes and beats in the Music Coach system.

Figure 4.1 shows the interaction of all the components. The software was built using the Qt application framework (Qt version 4.6). The signal and slot mechanism in the Qt framework is used for inter-component communications. For example, when the FFT thread detects a new note it will 'emit' a 'signal' to the Music Coach object at a pre-configured 'slot'. The master object will then take appropriate actions to handle the displaying of the detected note. Similarly, the accelerometer thread sends a tempo update signal when it detects a change in tempo.

The MIC thread made use of the pulseaudio sound server. This server allows the user to create full-duplex audio applications, which was required for this application because the metronome object produced sound at the same time as the microphone thread recording sound.

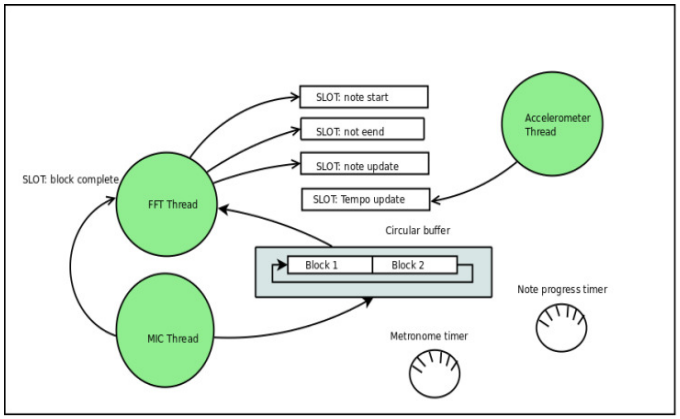

Figure 4.1. System Architecture.

5. DESIGN AND IMPLEMENTATION

5.1 Pitch Detection

One of the best known techniques for pitch recognition uses the Fast Fourier Transform (FFT). The FFT transforms a set of audio samples in the time domain to a set of samples in the frequency domain for frequency analysis.

Figure 5.1 illustrates how an FFT analyzes a monophonic musical source. A continuous sequence of sound samples $x(n)$ is fed to a windowing function, restricting the set of points in the waveform to a short segment of time. The FFT algorithm is then performed on the windowed samples $y(n)$, producing a vector $Y(k)$ of frequency domain coefficients. The pitch of the sound source can then be determined by scanning the $Y(k)$ values to determine a local maximum in this time window.

Figure 5.1. Using FFT to Analyze a Monophonic Musical Source.

Although the FFT algorithm designed by J.W. Cooley in 1965 improved the general Discrete Fourier Transform (DFT) by reducing the computational complexity from $O(N^2)$ to $O(N \log N)$, it was still insufficient for real-time purposes on personal computers in early 1990's with a processing load of approximately 184000 multiplications and additions per second at a sample rate of 20Hz and a window size of 512 samples. Other techniques such as autocorrelation in the time domain and building a large filter bank to determine pitch were explored before the dawn of high speed personal computers in the 1990's with a certain degree of accuracy, however, specialized hardware was required in that case [8].

Nowadays, the time to compute an FFT on a sample window of 2048 samples on a modern computer capable of billions of instructions per second is less than a millisecond. The Nokia N900 phone has an ARM Cortex-A8 600 MHz processor capable of 3.33 MIPS/MHz or 2000 MIPS at 600MHz.

As long as the time for the FFT is shorter than the sample window size, real-time pitch analysis is possible. Measurements on the N900 phone showed that for a sample window of 100ms with 4096 sample points the FFT took approximately 3ms to compute with the full overhead of the operating system and simultaneously recording the next window while computing. This measurement was done using the QTime component in the Qt library with millisecond accuracy.

A circular buffer shown in Figure 5.2 is used to record sound to facilitate the mechanism of analyzing a sample window while simultaneously recording the next window. Note that each buffer is reused after 2 cycles.

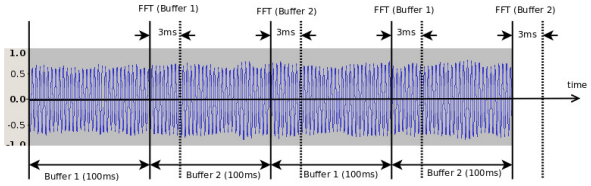

Figure 5.2. Using a Circular Buffer to Allow FFT Pipeline Processing.

One drawback of using an FFT is that the frequencies at which the coefficients $Y(k)$ are computed are evenly spaced rather than logarithmically spaced as with a linear sequence of musical notes as shown in Figure 3.1.

Analyzing live musical performance involves continuously processing a moving time window of audio data and obtaining the frequency spectrum from it. The choice of time window size depends on the expected smallest duration of the performed notes as well as the frequency range expected from the performance.

The following definitions and formula will help to gain insight into the expected accuracy of the real time analysis of a musical performance.

R = sample rate (Hz)

N = number of samples in time window

T = N/R (period of time window)

F = R/N (frequency resolution of spectrum analysis)

For example, if you have audio data sampled at 44100 Hz and you choose a sample window which contains 2048 samples, this will result in a time window of 46ms and a frequency resolution of 21Hz. If the frequency spacing of the notes to be analyzed is far less than 21Hz, these settings will not be sufficient to meet the accuracy requirements and thus the sample window size will need to be increased. However, increasing the sample window size will increase the delay between the time a note is played and the time feedback is given. This means there will always be a trade off between accuracy and real-time delay.

The FFT library being used for this project is FFTW developed at MIT [9]. Using a series of experiments, it was proved that it was approximately 50% fast than 40 competing algorithms during 1998. Recent scans of the literature show that FFTW still contains the fastest FFT open source implementation available today.

5.2 Tempo Detection using Accelerometer Readings

As stated in 3.5, the main challenge in tempo detection is to design a light-weight peak detection algorithm that does not involve much computational power but still yields reasonable accuracy.

A naïve approach would be the zero derivative point approach. However, this algorithm is extremely sensitive to noise. As we can see in Figure 5.3 where the amplitude stream is shown, data collected from the accelerometer are naturally noisy and there are many local maximums and minimums that will significantly confuse the naïve zero derivative approach.

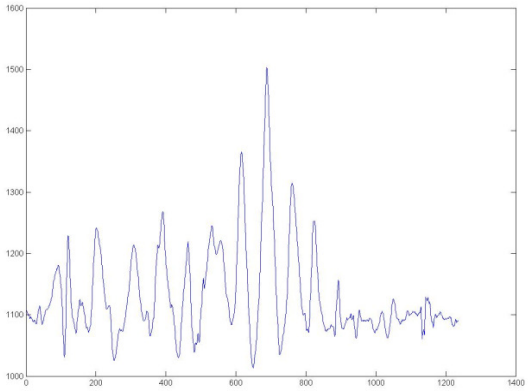

Figure 5.3. The vector amplitudes of a accelerometer reading stream when the user is moving the phone back and forth.

In order to overcome the noise issue, we designed two simple algorithms to detect 'significant' peaks in the signal. They are described in pseudo code below.

Algorithm 1. Detect Significant Drop or Rise

```
Smooth data by calculating a 5-reading average;
IF current_reading > max_value
        max_value = current_reading;
        max_time = current_time;
END
IF current_reading < min_value
        min_value = current_reading;
        min_time = current_time;
END
IF detecting_max
        IF current_reading < (max_value – DELTA) AND
        max_value > NOISE_THRESHOLD
                REPORT (peak, time)
                SET detecting_max to FALSE
        END
ELSE
        IF current_reading > (min_value + DELTA) AND
        max_value > NOISE_THRESHOLD
                REPORT (peak, time)
                SET detecting_max to FALSE
        END
END
```

Algorithm 2. Detect Significant Change

```
Smooth data by calculating a 5-reading average;
IF |current_average – previous_average| > THRESHOLD
        IF no peak was detected X time ago
                RECORD (peak, time)
        END
END
```

The two algorithms are both tested on the N900. Both of them are robust and efficient detecting fast movements. When the movement is slow, algorithm 1 yields better results than algorithm 2, which is easily explained by the fact that slow movements do not generate a significant change in acceleration.

5.3 Audio Output

Music Coach has the option of using audio output to provide tempo indication to the user. Tick sounds are played according to the current tempo at each beat.

Two kinds of beat sounds are created at frequency 6000Hz and 4500Hz for the application to play to the user as tempo indicators. The beat sounds are generated by simply sampling a sine wave:

$a_i = \frac{255 \times \left(\sin\left(\frac{2\pi f_k i}{f_s} \right) + 1 \right)}{2}, i = 0..N$, where f_k is the frequency of the sine wave and f_s is the sampling frequency. In our application, f_s is set to 44100Hz.

Sound management in Maemo 5 is done through PulseAudio. In order to play a sound, the application passes a sound buffer to PulseAudio specifying sampling frequency, data format, and channels, and PulseAudio will automatically schedule the task and interact the low-level drivers to produce the sound output.

5.4 GUI Design

Besides the limitation of processing power, applications on mobile phones will also have to deal with small displays. The Nokia N900 phone we use to implement our application has a 3.5inch LCD touch screen. In order to provide a good user experience on a limited size display, much consideration has been given on the GUI design.

In the current design, Music Coach provides the user with real-time feedback on their musical performance mainly through the visual display. The idea is to translate detected pitch and timing inaccuracies into easily recognizable measures on a screen. There are many open-source applications available. Taking these example applications as reference, we designed our GUI shown in Figure 5.4.

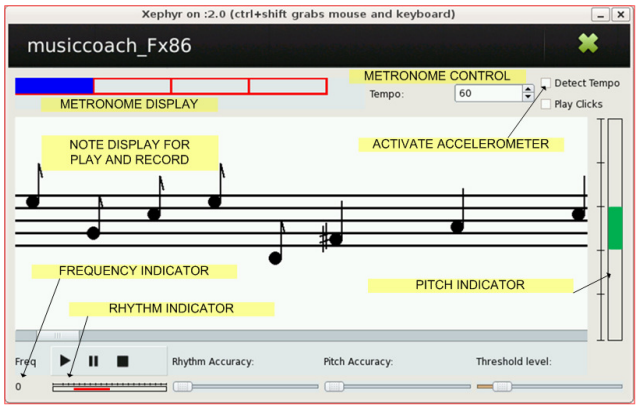

Figure 5.4. The Music Coach Application GUI.

We used the Qt framework to develop our GUI. However, due to the fact that Qt 4.x releases are not stable in the Maemo 5 environment, we decided to launch it in classic Windows style.

In the main window the user will see notes detected or recorded in a standard music score style. The application maintains a note history played by the user and can be viewed later after the user finishes their performance. In this way, Music Coach can also be used to generate a music transcript.

On the top of the screen is the metronome and related tempo control options. The user can choose to enable or disable tempo detection using the accelerometer, as well as to enable or disable the audio output generated by the metronome.

On the bottom of the screen are the rhythm indicator and the pitch detection control options. The user can configure the pitch detector to adapt to a noisy environments by setting the threshold level.

On the right of the screen is the pitch indicator that provides visual feedback to the user about their pitch accuracy. The purpose would be to keep the bar in the green zone; if the note is too high or too low, the bar will slide up or down and change to orange. If the note is totally off, the bar will slide to the top or bottom and be displayed in red.

6. RESULTS

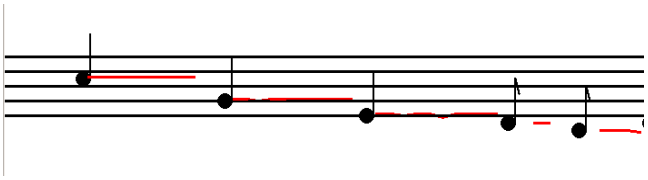

Figure 6.1. Real time music performance analysis, black notes show pre-loaded music score, red line shows actual notes played and duration.

Evaluating the real time performance of music is done by using a line drawn on the music staves. For each note this line can move between 3 discrete points on the y-axis (pitch axis). It can be "in tune" which would represent a centre point of a normal discrete note. It can be "sharp" or too high which would be a point 1/6 of the stave spacing above the "in tune" point. It can be "flat" or too low which would be a point 1/6 of the stave spacing below the "in tune" point.

On the x-axis (time axis) the length of a continuous line segment represents the length of time the note was played. The resolution of the line segment is equal to the size of the sampling window. For this particular example this was 100ms.

Analyzing the line segments after a performance will give a musician a good idea of how accurately a note was pitched over the complete duration of the note. This includes "wavering" during the note performance, where the performer does not maintain constant pitch. This can clearly be seen for the third note in Figure 6.1. The rhythmic accuracy can be extracted by looking at the start points of each line segment, relative to the note positions. In this example, the first 3 notes were accurately played whereas the performer played a little late on the 4th note. The length of the note performance gives an idea of the playing style. "Legato" playing, which is required for certain sections of music, is a style in which the performer holds the notes as long as possible before playing the following note. "Stacatto" playing occurs when the performer plays the notes with very short duration. In this example, "legato" style playing was used for the first three notes and "staccato" style playing was used on the 4th note.

Statistics can be calculated from this data to give the performer a final rating in terms of percentage of notes that were on pitch and average number of milliseconds of early or late note attacks together with their corresponding variance.

7. SUMMARY AND FUTURE WORK

In this paper we present our mobile phone application Music Coach that has been implemented on a Nokia N900 smartphone. It utilizes the audio input/output as well as accelerometer to provide the user real-time feedback on their musical performances. Our result shows that the application is easy to use and convenient for musical learners, especially beginners. In addition to the real-time feedback it is possible for a performer to review their performance and see exactly how accurate their pitch or rhythm was for every single note using a line graph display on a musical stave.

Although our current implementation of Music Coach demonstrates the fundamental idea of the application and shows

the potential of coaching/learning software on mobile phones, we can still foresee following improvements in the future.

- GUI Design: Better GUI design can be achieved by releasing prototypes and collecting user feedback on the overall experience.
- Improved Audio Isolation: A Bluetooth headset/mic can be used and attached to the instrument to reduce noise.
- Threading: Separate threads can be used to detect note timing and note pitch. Note timing can be done using threshold in the time domain, which will also allow more accurate evaluation of rhythm.
- Professional music typesetting Library Support: guido[7] note library can be used for typesetting music on the screen. This is "latex" like music library for professional typeset music.
- Display-Free Feedback: Buzzer can be sued to provide feedback about note performance instead of visual feedback.

8. REFERENCES

[1] GuitarToolkit, http://appshopper.com/music/guitartoolkit

[2] TyroTuner, http://www.appstoreapps.com/2008/07/30/tyrotuner/

[3] ZOOZBeat: http://www.zoozbeat.com/

[4] PeakDet, http://billauer.co.il/peakdet.html

[5] GTick, http://www.antcom.de/gtick/

[6] MuseScore, http://www.musescore.org/

[7] GUIDOLib qt music notation library, http://guidolib.sourceforge.net/

[8] Kuhn, W.B., Gupta, P. and Kumar, P.R., *A real-time pitch recognition algorithm for music applications*, Computer Music Journal, pp. 60-71, 1990

[9] Frigo, M., and Johnson, S.G., *FFTW: An adaptive software architecture for the FFT*, IEEE International Conference on Acoustics Speech and Signal Processing, volume 3, 1998