MusicScore: Mobile Music Composition for Practice and Fun

Zimu Liu, Yuan Feng, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

ABSTRACT

In this paper, we present our design of *MusicScore*, a professional grade application on the iOS platform for music composition and live performance tracking, used by composers and amateurs alike. As its foundation, we have designed and implemented a high-quality music engraver, capable of real-time interactive rendering on mobile devices, as well as an intuitive user interface based on multi-touch, both built from scratch using Objective-C and Cocoa Touch. To make *MusicScore* appealing to the general population for their practice and fun, we have introduced a unique auditory capability to *MusicScore*, so that it can "listen" to and analyze live instrument performance in real time. In order to compensate for the imperfect audio sensing system on mobile devices, we have proposed a collaborative sensing solution to better capture music signals in real time. To maximize the accuracy of live progress tracking and performance evaluation using a mobile device, we have designed a collection of note detection and tempo-based note matching algorithms, using a combination of microphone and accelerometer sensors. Based on our real-world implementation of MusicScore, extensive evaluation results show that *MusicScore* can achieve acceptably low error ratios, even for music pieces performed by highly inexperienced players.

Categories and Subject Descriptors

H.5 [**Information Interfaces and Presentation**]: User Interfaces and Presentation; H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Signal Analysis, Synthesis, and Processing*

Keywords

Music Composition, Mobile Application, Signal Processing

1. INTRODUCTION

Since the advent of the iPad, there has been a steady trend of new professional-grade applications (or "apps") released for multi-touch mobile devices with a large display, such as Adobe Photoshop Touch for image processing, as well as GarageBand for soundtrack editing. Such professional applications have made it possible for creative professionals to work on their projects effectively with a mobile device.

With a burgeoning repository of mobile apps for professionals, it may be a surprise that no mobile apps exist for interactive music composition, score editing, and live performance tracking. Table 1 lists a few representative mobile apps, related to music composition and practice. For music composition, Symphony is the only existing mobile app supporting music editing, in comparison to NoteStar and forScore, which only provide simple annotation in downloaded images of music sheets. However, we notice that Symphony suffers from a poorly designed music typesetting system that violates basic score layout guidelines, an inferior user interface that is hard to learn and use, and an incomplete music symbol support. With respect to live performance tracking, Tonara claims that automatic page-turning is supported using sensed live piano performance. However, in reality, Tonara can only track simple music pieces at moderate tempo, and it fails to work as a slightly complicated piece is played or a piece is played in a fast manner. Such a lack of high-quality apps in this category reflects the inherent complexity of developing professional-grade music apps. Hence, we firmly believe that a well designed, interactive, high-performance and feature-complete music composition app should be designed and built. This paper presents design challenges and our proposed solutions in *MusicScore*, a professional-grade music app on iOS mobile devices, such as the iPad.

Table 1: Representative mobile apps related to music composition and practice.

Mobile App	Features
Symphony	Music composition; Real-time score rendering; Music synthesis
NoteStar	Music sheet display; Key transposition; Demo play with progress indication
forScore	Music sheet display; Simple annotation
Etude	Music sheet display; Demo play with progress indication
Tonara	Music sheet display; Auto page-turning

First, *MusicScore* is designed for professional composers. We developed a high-quality music engraver from scratch in Objective-C, achieving artistic and pleasing engraving results by strictly conforming to traditional engraving rules. Our music engraver is optimized for real-time interactive score editing on a resource-limited mobile device, based on intuitive multi-touch gestures. *Second*, *MusicScore* is also designed for the general population who love music, such as piano learners and amateur musicians. An iPad with *MusicScore* can be used on the music stand to display a music score digitally, and to keep track of the progress of a live performance by

MM'12, October 29–November 2, 2012, Nara, Japan.

automatically highlighting the notes and measures being played, as well as turning pages. Thanks to such live progress tracking, *MusicScore* is also able to evaluate the quality of live performance, and point out mistakes and imperfections along the way (*e.g.*, an inconsistent tempo and a missed note).

In order to make it possible for *MusicScore* to accurately "listen" to a live music performance and keep track of its progress, a number of important challenges needs to be addressed. *First*, due to limitations of built-in sensors in mobile devices and inherent constraints of audio signal processing, we discover that it is a non-trivial task to capture high-quality signals and conduct accurate performance tracking. In *MusicScore*, we take full advantage of available sensors on mobile device to sense signals from the instrument. After carefully integrating signals from the microphone and the accelerometer sensor, *MusicScore* uses a set of carefully designed algorithms to identify the notes played, and compare them with the music score for live performance tracking and evaluation. Due to the complexity of instrument signals, we present a note matching algorithm using tempo estimation to minimize errors when identifying notes, typically due to mistakes made by inexperienced players.

To objectively assess the quality of *MusicScore*, we present a thorough evaluation of the performance on our implementation. Individual functional components are carefully tested and measured, so that all design and implementation details are examined.

The remainder of this paper is organized as follows. In Sec. 2, we first introduce *MusicScore*'s music engraver and user interface, designed for professional composers using a mobile device. In Sec. 3, we discuss limitations and challenges on mobile devices as we are trying to support the auditory capability in *MusicScore*. Sec. 4 presents in detail how audio and vibration signals are processed in *MusicScore* to achieve accurate and efficient tracking of live instrument performance. In Sec. 5, we thoroughly evaluate *MusicScore* to understand its quality as a professional music composition application, and its accuracy as an automated performance evaluator. We conclude the paper with a discussion of related work and final remarks in Sec. 6 and Sec. 7, respectively.

2. *MUSICSCORE* FOR PROFESSIONAL COMPOSERS

The primary design objective of *MusicScore* is to provide professional composers with a well designed, interactive, high-performance and feature-complete mobile app for composing and editing music scores. *MusicScore* is designed and built from scratch to work on mobile devices with limited capabilities, rather than as a desktop application, where alternatives have been available for a long time. To the best of our knowledge, no such mobile apps exist to fulfill the need of professional composers to compose music, perhaps due to the inherent complexity of designing and building it.

2.1 A Professional Music Engraver

To make composing music on mobile devices possible, the most critical component is a typesetting engine that supports dynamic music notation layout and rendering, commonly referred to as an *engraver*. We believe that a professional-quality engraver on mobile devices must exhibit two important properties. *First*, correctness of engraving layout. As an app used by professional composers, the music sheet produced by the app must conform to strict artistic rules of music engraving (with its legacy spanning centuries), so that the result of engraving is intuitive and artistically pleasing to sightread. *Second*, efficiency. The engraver must respond swiftly to user input in real time, so that a smooth and satisfactory user experience is guaranteed. Unfortunately, after an extensive investigation on existing works, it is discovered that neither of the two top-of-the-line open-source engravers, libmscore (from the MuseScore project on the desktop) and LilyPond [10], satisfies our demanding needs, as shown in Table 2. Therefore, in *MusicScore*, we have designed and built a new mobile music engraver entirely from scratch, making design decisions that are tailored to offer the best user experience on mobile devices.

Table 2: Two open-source engravers: a comparison.

	libmscore	LilyPond
Correctness	○ Some major layout errors	● Almost perfect
Efficiency	● Suitable for interactive score editing	○ Complex layout decision making suitable for batch processing only
Suitability for mobile OS	◐ Based on Qt (in immature α-stage on iOS)	● Based on pure C++ and Scheme

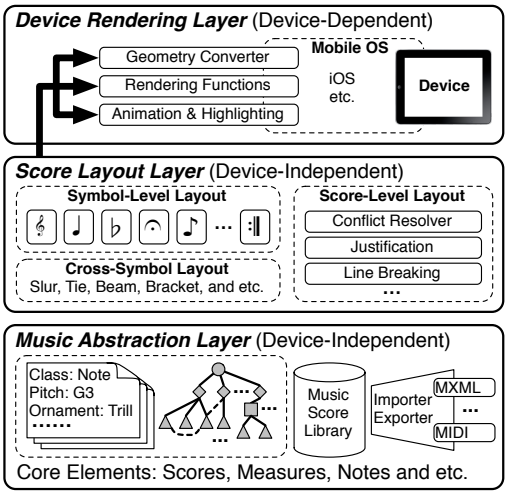

Figure 1: An architectural overview of *MusicScore* engraver.

To guarantee the engraving quality and to maximize the efficiency of our new engraver on mobile systems, we adopt a layering design pattern as illustrated in Fig. 1. The device rendering layer is closely tied to the mobile OS and device hardware, while the score layout layer and the music abstraction layer are *device independent*, in that the core data structures storing score elements and the core algorithms used to compute the score layout are not tied to a particular type of mobile OS or device hardware.

The Music Abstraction Layer contains the metadata of all musical elements, from the score at the highest level, to notes and rests at the lowest. These elements are organized using a tree-like hierarchical data structure, with some special links between nodes, *e.g.*, two tied notes. Such a core data structure essentially captures and presents the complex structure of a music score. Scores are indexed and stored in the `sqlite`-powered Music Score Library, and can be imported or exported through various public score-exchange formats, such as `MusicXML` and MIDI. To improve its flexibility, additional formats can be supported using external import/export plug-ins.

The Score Layout Layer is in charge of the layout placement of musical elements in a score, by strictly following guidelines of traditional music engraving [12]. Within this layer, we further partition layout components into three levels: symbol level, cross-

symbol level (such as the placement of a bracket over a group of notes), and score level, as shown in the figure. Each component is decoupled from others using the *delegate* mechanism. All engraving algorithms inside the score layout layer are fully optimized so that results can be computed swiftly on mobile devices. In addition, the produced layout results are expressed in an abstracted score coordination system, using the quarter note head as its unit of size, which can be easily converted to any device-specific coordination system at the rendering stage.

The Device Rendering Layer retrieves the detailed layout results from the score layout layer, and renders them in the actual device through the OS drawing API. Since every mobile device has its own geometric properties, conversions from device-independent coordinates to device-specific coordinates should be conducted accordingly. To make it possible for users to interact with musical symbols, each musical symbol is encapsulated by a graphic container in the OS, so that touches on that symbol can be easily detected.

2.2 A Score Composition User Interface

Taking full advantage of multi-touch capabilities on the iPad, we have designed an intuitive user interface to compose music scores using a combination of simple touch operations and multi-touch gestures. For example, the pitch of a note can be changed by dragging a note vertically, a group of musical symbols can be selected by dragging a selection box, and a chord can be selected using a two-finger touch. As demonstrated in Fig. 2, all properties of musical elements can be edited "in-place" using popover controls, with minimal finger movement. Users may also enter a score using a multi-touch virtual keyboard, as if they are composing on an actual piano keyboard.

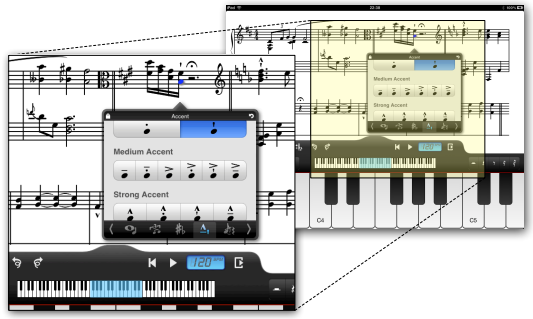

Figure 2: A screenshot of running *MusicScore* on the iPad.

When composing music, it is critical to provide users with audio feedback in real time. In order to achieve this goal, we have implemented a real-time audio engine with a 256-channel synthesizer, as shown in Fig. 3, supporting music score playback as well as interactive user operations, *e.g.*, when playing on the virtual piano keyboard and when modifying the pitch of a note. Using both the `vDSP` framework and the `AudioUnit` framework at the lowest level, the *MusicScore* audio engine is carefully tuned to achieve the highest possible performance on the iPad, with respect to both processing delays and the sound quality. To support sounds from multiple instruments as in an electronic keyboard, sound libraries can be dynamically loaded or unloaded in the audio engine, so that users may enjoy the same piece played back with different musical instruments.

3. *MUSICSCORE* FOR PRACTICE AND FUN

MusicScore is designed from the ground up to allow professionals to create music using a user-friendly composition interface and a powerful engraving backend. In addition, we are firm believers that a high-quality mobile app should be exciting and fun for everyone, and we strive to make *MusicScore* useful to piano learners and amateur musicians alike. A casual *MusicScore* user may not be interested in composing music, yet she may find *MusicScore* useful and fun if it is capable of performing the following tasks:

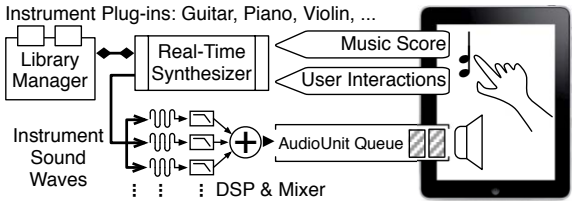

Figure 3: The real-time audio engine in *MusicScore*.

- ▷ Displaying music scores on the iPad placed on a music stand, and allowing her to change the score to suit her own tastes (*e.g.*, adding ornaments and fingering). Since *MusicScore* is designed to support complex engraving and interactive editing, this can be achieved without difficulty.
- ▷ "Listening" to her live piano performance, keeping track of the progress of her play in the music score, and automatically turn pages for her as needed.
- ▷ Dynamically evaluating her live piano performance, by identifying possible errors in real time as the performance progresses, including incorrect or missed notes, incorrect note timing, and inconsistent tempo. As errors are being highlighted in the score, she can focus more on her weaknesses. As such, *MusicScore* acts as a private music tutor when she is practicing.

Clearly, to achieve these goals, we must enable the auditory capability in *MusicScore*, *i.e.*, audio and other signals from a piano must be captured and analyzed on mobile devices in real time. In this section, we describe the characteristics of signals to be sensed by devices, and discuss the challenges to support high-quality music signal sensing on mobile devices in real time.

3.1 Characteristics of Sound from the Piano

To be able to "listen" to a live piano performance, the first step is to thoroughly understand signals from the piano. Different from the unpleasant pure tones defined by note pitches, which have sinusoidal and periodic waveforms with certain frequencies, the sound generated from a piano has a complicated waveform. Though musical sounds vary from a piano to another piano, they share a set of characteristics that can be utilized for real-time analysis.

Fundamental frequency. The fundamental frequency, often denoted as f_0, represents the lowest of frequency of a periodic waveform. In a modern Western music system with twelve-tone equal temperament, each note pitch is assigned to a unique fundamental frequency, so that all instruments can be tuned for a performance. By international standard, the `A4` is set at 440 Hz, and other notes' fundamental frequencies can be derived using the logarithmic rule—the frequency ratio between two adjacent notes is the twelfth root of two, $2^{1/12} \approx 1.0595$. Furthermore, notes are grouped by octaves, where an octave corresponds to a doubling in frequency, *i.e.*, $f_0^{\mathtt{A5}} = 2f_0^{\mathtt{A4}} = 880$ Hz. Naturally, to evaluate a player's performance, we should compare frequency characteristics of sound, against the expected fundamental frequencies of notes in the score. Note that a few notes of a piano might be slightly out of tune due to temperature or humidity changes. *MusicScore* can be

calibrated with the player's piano and learn its sound characteristics, so that notes slightly out of tune can be correctly recognized.

Harmonic series. Musical instruments are often based on acoustic resonators, such as strings in a piano. Due to the inherent spacing of the resonances, frequencies of waves are mostly at integer multiples, or *harmonics*, of the fundamental frequency f_0, and a harmonic series is hereby formed. Take note `A0` as an example, the power is ideally concentrated at frequency points around $f_0^{\texttt{A0}} = 28$ Hz, $2f_0^{\texttt{A0}} = 56$ Hz, $3f_0^{\texttt{A0}} = 84$ Hz, and etc., and the higher orders of harmonics have decreasing levels of energy. However, because of the stiffness of real strings in the piano, the frequencies are not exactly harmonic and can be modeled by $f_n = nf_0\sqrt{1+Bn^2}$, where B is the inharmonicity coefficient of an instrument [5]. Since the order of magnitude of B is usually between 10^{-4} and 10^{-5} for the piano, only lower harmonics ($n \leq 4$) can be used to accurately infer f_0.

Amplitude envelope. When an acoustic musical instrument produces sound, the loudness of the sound changes over time. To quantify the loudness, we can measure the instantaneous amplitude of the wave, as illustrated in Fig. 4. Though the changes of amplitude vary significantly across players and instruments, the sound of a single-note play can be, in general, partitioned into the following four stages [11]: (1) *Attack*: the initial rise of energy from zero to peak after the key is pressed; (2) *Decay*: the subsequent slight drop of energy from the attack level to an almost constant level; (3) *Sustain*: the period when the loudness is nearly constant after the decay, until the key is released; and (4) *Release*: the time taken for the sound to fade away after the key is released. Although there is only one global peak, we observe that there exist several local peaks in the attack, decay, and sustain stages, where energy level is higher in general. Clearly, when multiple notes are played sequentially or simultaneously, as instructed in a music score, the overlapping sound waves would produce a series of peaks: some represent real note attacks, but others do not. In order to keep track of the progress of live piano play and to identify the timing of each note, we would need to correctly detect true attacks of notes when they are mixed together.

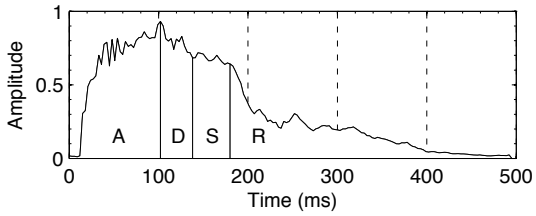

Figure 4: The instantaneous amplitude of note `G4`, played on a grand piano, with four stages labeled.

3.2 Imperfect Audio Sensors on Mobile Devices

Modern mobile devices have been equipped with various sensors, including the microphone, accelerometer, proximity sensor, ambient light sensor, and gyro, serving as "input" sources of mobile apps. Though mobile sensors have empowered mobile apps, they do have their limitations. In the application scenario of *MusicScore*, the built-in microphone is naturally the major signal source of *MusicScore*. In order to accurately and effectively track and analyze the player's live performance, we must identify constraints of the audio sensing system on mobile devices.

Frequency response. In contrast to professional grade audio recording systems, most audio recording systems on mobile devices have a limited frequency response, due to inferior microphone sensors, analog-to-digital converters, and software limiters. As shown in Fig. 5, it can be observed that there exists a clear drop of frequency response below 150 Hz on mobile devices. Therefore, to some extent, mobile devices would experience "listening" problems for notes below `D3`, whose fundamental frequencies are below 150 Hz.

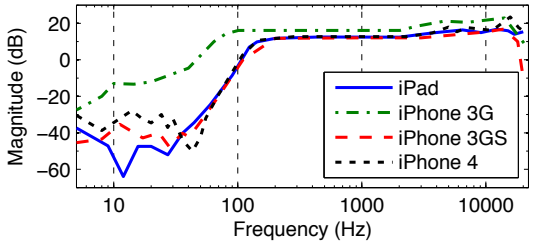

Figure 5: The frequency response of built-in microphones on iOS mobile devices.

To address the issue of the inferior frequency response below 150 Hz, we take advantage of the harmonic effect in a piano, and in particular lower orders of harmonics ($k \cdot f_0, k = 2, 3, 4$), to help us identify notes with low f_0s. For example, a weak peak of `B0` at $f_0 = 31$ Hz can be confirmed by a growing energy peaks at 62 Hz, 124 Hz, and 247 Hz. It should be noted that higher orders of harmonics are not used in *MusicScore* due to the aforementioned inharmonicity.

Directional characteristics. Although most microphones on mobile devices are omnidirectional, they are usually embedded into the body of the device for a better external look, and sound waves can only reach the internal microphone through a hole or cone carved on the body frame. Such a design leads to a unidirectional-like sound perception, *i.e.*, the microphone picks up sound mostly from the direction that the hole faces, and to a lesser extent, the sides as well. To evaluate the effect of directionality of built-in microphones on mobile devices, we have recorded a grand piano playing in a $4 \times 4 \times 2.5$ m^3 room, using an iPad is placed on the music stand with microphone facing the ceiling (portrait mode). As shown in Fig. 6, even though notes are played in the same key-press velocity, some notes cannot be clearly "heard."

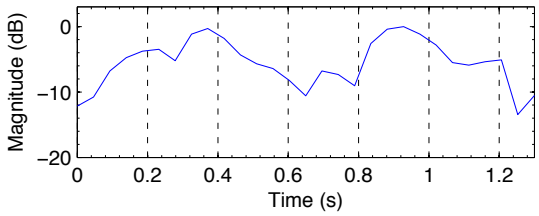

Figure 6: The normalized magnitude envelope of FFT on signals from the iPad on the music stand.

Although the note pitch can still be identified, the amplitude envelope cannot accurately reflect the exact timing of the note attack, due to the directionality of the built-in microphone. Such a deficiency of the audio sensor on mobile devices suggests that we may consider other sensing sources to evaluate the piano player's live performance.

3.3 Collaborative Sensing for Real-Time Music Capture

Since we need to judge whether a piano performer plays the *correct note pitches* at *righting times*, we need to conduct spectral analysis on the audio signal, after slicing the stream of audio samples

into "frames" using a sliding window. By examining the sound's frequency spectrum within a specific time frame, against the expected fundamental frequencies of one or more notes at the corresponding time in the score, the player's performance can then be evaluated in real time. In *MusicScore*, the zero-padded Quadratically Interpolated Fast Fourier Transform (QIFFT) is adopted, due to its simplicity and accuracy. However, as stated by the Heisenberg-Gabor limit, one cannot achieve a high temporal resolution (in the time domain) and a high frequency resolution (in the frequency domain after the Fourier transform) at the same time, *i.e.*, simply applying QIFFT on the audio signal cannot achieve satisfactory spectral and temporal accuracies, which are required in *MusicScore*. Hence, we must carefully design special techniques, taking into account the application scenario of real-time performance evaluation.

Intuitively, QIFFT is sufficient to identify a single note attack or notes played far apart. However, in music pieces, chords, which consist of a set of two or more "nearby" notes sounding simultaneously (*e.g.*, `C2-E2-G2` in `C` major), are often seen. In order to correctly identify a chord, we must be able to distinguish individual note pitches of the chord within a single frame. In the context of the spectral analysis for *MusicScore*, this implies that the minimum frequency separation of QIFFT should be no greater than than frequency difference between two notes in an arbitrary chord. As we have previously mentioned, the f_0 spacing between adjacent notes is logarithmic rather than linear, *i.e.*, the frequency difference between two adjacent notes of low pitches will be much smaller than that of high pitches. For piano, the required minimum frequency separation should be less than the frequency difference between its two lowest notes, *i.e.*, $\Delta f_{\min} = (2^{\frac{1}{12}} - 1) \cdot f_0^{\text{lowest}} = 1.6$ Hz. To achieve such a frequency separation, each frame must be longer than $\text{MAFS}/\Delta f_{\min}$ seconds long [1], where MAFS is the minimum allowable frequency separation coefficient given by a set of QIFFT parameters, and MAFS is greater than 1. For a zero-padding factor of 5 (*i.e.*, the QIFFT input size is 5 times the frame size with zeros padded) with the widely used Hann window function, MAFS is 2.28 in practice, implying that each frame would be at least 1.4 seconds long, which will clearly hurt the temporal resolution of our signal analysis. In addition, a one-second lag is noticeable to users, and intolerable for real-time interaction. Aiming to improve the temporal resolution and reducing the processing delay, we propose to take advantage of the effect of harmonics in the piano discussed in Sec. 3.1, and use the fourth-order harmonic to distinguish adjacent notes, so that $\Delta f_{\min}$ can be relaxed to 12.8 Hz and the each frame can thus be reduced to 178 ms long.

However, in the time domain, a 178-ms resolution is still not enough for live play progress tracking. For example, in a typical performance with *Allegro* tempo (a quarter note $=$ 140 beats per minute (BPM) $=$ 428 ms), each eighth note lasts for around 214 ms. To accurately identify a consecutive sequence of eighth notes at the same pitch, which are often seen in a score, we should achieve a temporal resolution of 100 ms, or even shorter, so that peaks and valleys of power can be observed. Note that by allowing two consecutive frames overlapping with each other 50%, we may reduce the interval between frames by a half. However, a shorter interval does not result in a better temporal resolution, since overlapping will also blur the changes of power in the time domain.

To overcome the shortcoming of audio signal processing with respect to the temporal resolution, in *MusicScore*, the built-in high-sensitivity accelerometer is used to work *in collaboration* with the audio recording system. When a mobile device is in contact with a piano, vibrations of the sounding board can be passed to the device body, and then be sensed by the internal high-sensitivity accelerometer (*e.g.*, iOS devices can achieve a sensitivity of 1 mg). Using the maximum accelerometer sampling rate of 100 Hz, we may be able to achieve a detection interval of 10 ms, which would help to significantly improve the accuracy of note attack identification and shorten its response delay.

To justify the feasibility of vibration sensing, we have conducted a few preliminary experiments using the built-in accelerometer. Fig. 7 shows that, when the device is placed on the music stand of the piano, all 15 note attacks can be identified from the accelerometer readings. Another advantage of such a collaborative sensing solution is that the directionality problem of internal microphones is resolved naturally, as the the hammer striking the string gives an abrupt beginning of a note.

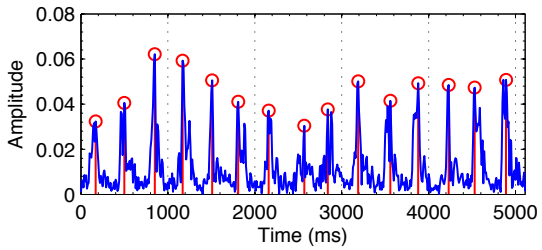

Figure 7: The instantaneous amplitude of accelerometer readings when a `C5` $\rightarrow$ `C6` $\rightarrow$ `C5` scale is practiced at a fast tempo of 180 BPM.

4. EVALUATING LIVE PERFORMANCE IN *MUSICSCORE*

In this section, we present our complete solution for evaluating live music performance, using innovative signal processing algorithms within our collaborative sensing solution.

4.1 Evaluating Live Performance with Progress Tracking: an Overview

In order to evaluate a live piano performance, both the sensed note pitches and the timing of the attack should be compared against the original music score. Hence, we first analyze the score to prepare the expected note attack sequence as *a priori* knowledge for real-time performance evaluation. Such a note attack sequence reflects the expected structures of input signals, including fundamental frequencies and timing information, as demonstrated in Table 3. This sequence will then be fed into signal processing modules running on mobile devices, as illustrated in Fig. 8, to conduct progress tracking and performance evaluation. Note that, though cloud-based signal processing is widely adopted in recent years, it does not satisfy the stringent delay requirements enforced by our real-time interactive music application. Our measurements on residential Internet connections have shown that the average round-trip time to popular cloud service providers is around 103 ms, with delay jitters up to $400 - 600$ ms. Clearly, this would lead to a very uncomfortable experience during real-time progress tracking, automated page turning, as well as music performance evaluation. Hence, all signal processing procedures are running on local mobile devices, rather than in the cloud.

4.2 Note Pitch-Attack Analyzer on Audio Signals

To sense the note pitch and timing, we first need to analyze the piano sound using QIFFT, as discussed in Sec. 3. In *MusicScore*, the captured audio stream is partitioned into overlapped frames (with frame size $n = 8192$ and overlapping coefficient $\alpha = 75\%$),

Table 3: An example of the expected note attack sequence produced in *MusicScore*.

#	Relative Duration Since Last Attack	Note	f_0 (Hz)	Relative Duration
1	-	C5 C4	523 262	1/8 1/4
2	1/8	D5	587	1/8
3	1/8	E5 G4	659 392	1/8 1/4
⋮	⋮	⋮	⋮	⋮

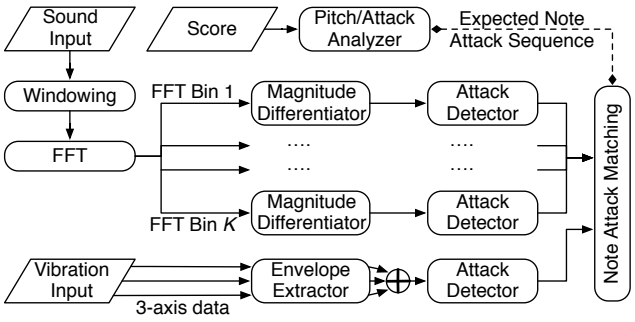

Figure 8: The signal processing flow chart in *MusicScore*.

and each frame of data is multiplied by a Hanning window function. Since the fundamental frequencies of notes in logarithmic scale while FFT bins are in linear scale, applying the Hanning window function can effectively minimize the effect of spectral leakage due to unaligned frequencies. After conducting FFT on the windowed data, we can easily derive the power in each FFT bin, *i.e.*, the magnitude of sinusoid with f_k^{bin}, as shown in the upper panel of Fig. 9. As we mainly focus on the frequency points covering Octave $0-8$, only FFT bins below 10 kHz are processed.

To identify note attacks from magnitude signals at any given f_0, we first use a high-pass filter to cut off the magnitude values below the human-audible note attack threshold, so that magnitude peaks of irrelevant background noise are excluded, and those representing potential note attacks are preserved. It must be noted that the computed magnitude values only represent the relative energy measured by the microphone, and we empirically choose 35 dB for the iOS audio system. Afterwards, the first-order different function is calculated and half-wave rectified, as demonstrated in the lower panel of Fig. 9, to unveil peaks in the processed magnitude curve after high-pass filtering. To efficiently extract peaks in real time, we choose the rectified first-order differentiation, since it can be conducted by simply storing magnitude values from previous FFT frame and subtracting current and previous values in a vectorized manner, which not only conserves memory usage on mobile devices but also reduces computational complexity.

Thanks to the smoothing effect of overlapped frames, magnitude series are smoothed over time and contain much fewer local sub-peaks around major peaks. Thus, the corresponding rectified difference series $\mathcal{D}(t)$ are able to reveal major peaks evidently. We identify the note attacks in audio signal by picking all local maxima in $\mathcal{D}(t)$ satisfying a threshold:

$$\text{Attack}(t) = (\mathcal{D}_t >= \mathcal{D}_{t-1}) \wedge (\mathcal{D}_t > \mathcal{D}_{t+1}) \wedge (\mathcal{D}_t >= \mathcal{D}_{\text{th}}).$$

By studying magnitude series of very "soft" piano play with the *una corda* pedal, we discover that $\mathcal{D}_{\text{th}}$ should be set to as low as

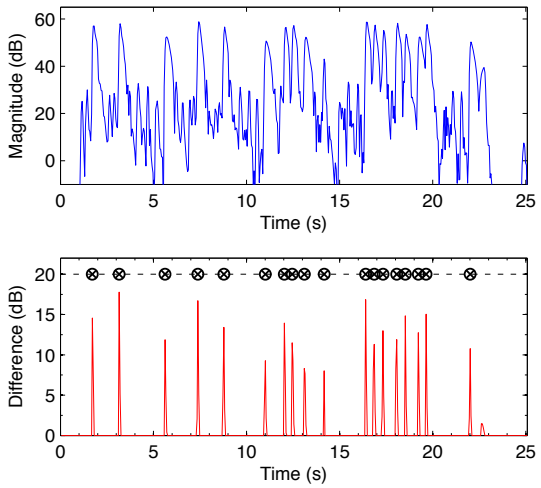

Figure 9: An example of audio signal processing for a performance with 16 measures. The upper panel shows the magnitude curve of FFT bin of 657 Hz (the bin closest to C5 = 659 Hz); and the lower one shows the half-wave rectified difference of magnitude signal after high-pass filtering, with identified note attacks marked on the top.

4.7 dB (*i.e.*, around three times of energy increase for a valid attack) to capture soft note attacks, which are often heard in a performance. Note that this threshold can be automatically adjusted by calibrating *MusicScore* with a user's own instrument.

4.3 High-Resolution Attack Detection using Vibration Signals

Due to the temporal resolution issue that we have previously discussed in Sec. 3, vibration signals from the accelerometer should also be properly processed to improve the accuracy of note attack analysis. For each signal sequence from 3 axes, the envelope of the vibration amplitude is first extracted by convolving with a 40-ms half Hanning window, which keeps the sharp attacks in the amplitude curve. After the envelope is extracted, envelope curves of all axes are summed for note attack detection. Fig. 7 has revealed that peaks in vibration signals generally rise much faster than they fall, which is very similar to an ADSR-shape curve of a note attack in audio signals. In order to extract note attacks from the vibration signal, we have designed a simple, yet sufficiently efficient, adaptive threshold algorithm for *MusicScore*, which is able to achieve a reasonable accuracy at its inherent temporal resolution.

To be specific, two criteria are used to detect a potential note attack: (1) The current amplitude envelope rises above the mean of the last few values, *i.e.*, $\mathcal{V}(t) \geq \beta \sum_{i=1}^{L} \mathcal{V}(t-i)/L$, where β represents the threshold coefficient and L is the number of historical values to be considered. This criterion is mainly based on our observations of the fast-rise and slow-drop phenomena in the signal. (2) No note attack has been identified yet, since the time t_v of the last valley in signal $\mathcal{V}$ received so far ($t_v = \sup\{t' : t' < t, \nabla\mathcal{V}(t') \leq 0\}$). This criterion effectively merges subsequent ("back-to-back") rises of amplitude into the first rise of a note attack.

In reality, for a sensitive note attack, we have chosen $\beta = 1.2$ to accommodate some slower rises of the amplitude envelope, which appear occasionally in our experiments due to "soft" notes; and, L is set to 4 (*i.e.*, 40 ms) to ensure a swift and accurate response to a passage of consecutive short note attacks. Fig. 10 demonstrates an

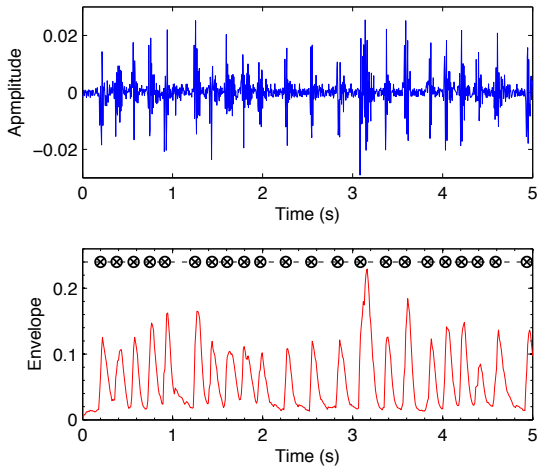

Figure 10: An example of vibration signal processing showing the first 5 measures in a performance. The upper panel shows the waveform of x-axis accelerometer data; the lower one shows the aggregated envelope with identified note attacks marked on the top.

example of vibration analysis using our proposed algorithm. Similar to $\mathcal{D}_{\text{th}}$, β can be learned by *MusicScore* during calibration. Note that, the attack detection criteria we have chosen can be easily implemented without logging the entire history of $\mathcal{V}$. Fortunately, with the `vDSP`-accelerated convolutional envelope extraction, the process of vibration signal analysis can be fully optimized for mobile devices.

Once both audio and vibration signals are processed, the potential attacks are merged to obtain accurate pitch and timing information, and then compared against notes in the expected play sequence from the score. As notes are matched progressively, *MusicScore* will automatically advance the play progress indicator, turning to the next page if necessary, and highlighting any detected mistakes (*e.g.*, pressing a wrong key) on the music score.

4.4 Intelligent Note Matching with Tempo Estimation

To achieve a better understanding of our proposed signal processing algorithms in real-world scenarios, we have invited two groups of piano players (5 people in each group), one with over 5 years of experience and another with less than one year of experience, to play two pieces of music with different tempos, and use *MusicScore* to track their performance. Although preliminary experiments show that our algorithm successfully handled the music piece performed by more experienced players at a moderate speed, the accuracy dropped significantly when notes were played in a very fast tempo, *e.g.*, a sequence of sixteenth notes at the tempo of *Vivace*, or when inexperienced players made some unexpected mistakes, *e.g.*, a key was accidentally pressed twice. As *MusicScore* is designed to serve the general population with a wide range of music background, it is a must to further improve its accuracy so that it is sufficiently robust to handle various performing speeds and human errors.

To achieve this goal, we need to identify possible cases that we may face in real-world performances. By analyzing collected sound tracks of musical performances, we believe that the accuracy is mainly affected by two categories of mismatching between sensed note attacks and actual notes in the score: (1) *Missed note attacks (false negatives)* usually happen when a very weak attack appears in-between a sequence of strong attacks of the same note or notes in different octaves, or in the worst case, when a player forgets to play a note. (2) *False note attacks (false positives)* occur as two or more adjacent peaks exist on a single note, or, some unexpected "attacks" from background noise. As demonstrated in Fig. 11, both cases will result in incorrect matching of note attacks.

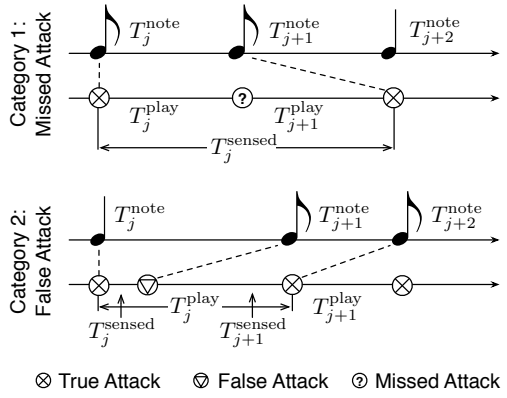

Figure 11: Two examples of tempo estimation when some note attacks are not identified accurately.

Since a piece of music is expected to be played in a near-steady tempo for a pleasant audition, we propose to use tempo tracking and prediction to minimize mismatching and further improve our algorithms. Before presenting our robust note matching algorithm, we first describe the mathematical terms used. Let T_i^{note} be the relative duration of the i-th note (*e.g.*, half note, quarter note, and etc.), T_i^{play} be the actual played duration for the i-th note (regardless whether or not the note is detected), and T_i^{sensed} be the sensed duration. In the ideal case, the corresponding true tempo is computed by $S_i = T_i^{\text{note}}/T_i^{\text{play}}$. As illustrated in Fig. 11, for the first category of mismatching, when p note attacks are missed after the j-th note, the system would compute the tempo as

$$\tilde{S}_j = \frac{T_j^{\text{note}}}{T_j^{\text{sensed}}} = \frac{T_j^{\text{note}}}{\sum_{i=j}^{j+p} T_i^{\text{play}}} = \frac{T_j^{\text{play}}}{\sum_{i=j}^{j+p} T_i^{\text{play}}} \cdot S_j.$$

For the second category, when q false attacks are detected after the j-th note, the system would calculate the tempo as

$$\begin{aligned}
\tilde{S}_j &= \frac{T_j^{\text{note}}}{T_j^{\text{sensed}}} = \frac{T_j^{\text{note}}}{\gamma_0 T_j^{\text{play}}} = \frac{1}{\gamma_0} \cdot S_j, \\
\tilde{S}_{j+1} &= \frac{T_{j+1}^{\text{note}}}{T_{j+1}^{\text{sensed}}} = \frac{T_{j+1}^{\text{note}}}{\gamma_1 T_j^{\text{play}}} = \frac{T_{j+1}^{\text{play}}}{\gamma_1 T_j^{\text{play}}} \cdot S_{j+1}, \\
\tilde{S}_{j+2} &= \ldots,
\end{aligned}$$

where $\sum_{k=0}^{q} \gamma_k = 1$. Note that it is safe to assume that γ_0 is not close 1, because when $\gamma_0 \to 1^-$ the case is equivalent to $q-1$ attacks are falsely detected after the $(j+1)$-th note. Since nearby T_i^{play} and T_i^{note} are mostly comparable due to the rhythmic structure of a performance, the relative error $\eta = |(\tilde{S}_j - S_j)/S_j|$ would be outstanding if either missed or false attacks occur. Observing that tempo tends to be relatively steady for a short passage, we could infer an expected tempo $\hat{S}_j$ from the history of played notes, and then compare it against $\tilde{S}_j$ to detect any significant error.

In order to derive the expected tempo from historical data, we need a valid prediction model for $\hat{S}$. Although many generic pre-

diction models, such as ARMA, have been widely used for prediction, the model parameter training usually involves complex computational procedures based on historical tempo data, and complicated data pre-processing has to be conducted to fulfill special requirements of these models [3]. Instead, we propose a simple but effective adaptive moving average model, which directly takes the sensed tempo history, and fully considers special characteristics of music performance. In a nutshell, our algorithm calculates $\hat{S}$ as the arithmetic mean of previous M values, where M is a varying window size determined by the structure of the music score. Since our empirical studies reveal that notes in adjacent beats are played with a more consistent tempo, M is automatically adjusted so that the averaging window always covers all the notes from the previous beat to the current one.

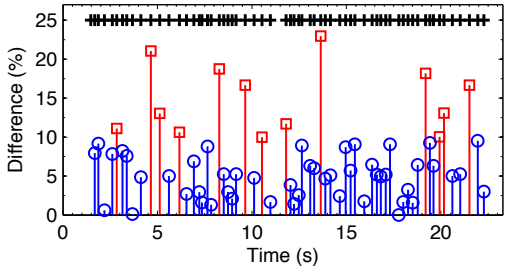

Figure 12: The normalized difference of true tempo series of a music piece performed by a typical inexperienced player, with note series marked on the top. Red stems with a square head indicate significant changes of tempo ($\geq 10\%$).

With the tempo $\hat{S}$ predicted, we also need to determine the relative error threshold η_{th} for comparing against $\tilde{S}$. Instead of using a constant for η_{th}, we have discovered that η_{th} would be best chosen according to the context of the current note being processed. As shown in Fig. 12, it is evident that the inexperienced player performs at a tempo that varies significantly as the note density changes, *i.e.*, when the relative note duration T_i^{note} of close-by notes changes notably. Such a observation implies that η_{th} should be a function of T_i^{note} to better deal with human errors. Consistent with the $\hat{S}$ prediction algorithm, we adopt the same adaptive windowing strategy. Then, within the window-covered range, the geometric mean is computed over the relative note duration ratios in logarithmic scale, since T_i^{note} is usually in the power of 2. Finally, using the Weber-Fechner logarithmic law [13], η_{th} is calculated as

$$\eta_{\text{base}} + \eta_{\text{flex}} \log_{10} \left(\prod_{i \in \text{Window}} \left| \log_2 \frac{T_i^{\text{note}}}{T_{i+1}^{\text{note}}} \right| \right)^{\frac{1}{M}},$$

where $\eta_{\text{base}} = 10\%$ and $\eta_{\text{flex}} = 20\%$, considering that the logarithmic ratio seldom exceeds 5 in music pieces. Note that the equation above can be further simplified to a form of arithmetic mean over $\log_{10} \left| \log_2 T_i^{\text{note}} - \log_2 T_{i+1}^{\text{note}} \right|$, which is easier to compute in a vectorized manner.

With carefully chosen models of $\hat{S}$ and η_{th}, we can then detect missed and false attacks using our intelligent note detection algorithm with tempo estimation. Our proposed algorithm, summarized in Algorithm 1, is executed every time when a newly sensed attack is going to be matched with the $(j+1)$-th note (*i.e.*, all previous j notes have matched with valid sensed attacks). Any false attacks can be detected if $\tilde{S}$ is alarmingly high as shown in Line $3-5$, while missed attacks can be detected iteratively when $\tilde{S}$ is significantly lower than $\hat{S}$ as shown in Line $6-14$.

Algorithm 1 *Intelligent Note Matching with Tempo Estimation*, triggered when a sensed note attack is going to be matched with the $(j+1)$-th note.

Input: The sensed note duration T_j^{sensed}, *i.e.*, the time since j-th note attack.
Output: If the sensed attack is valid

1: Obtain the expected tempo $\hat{S}_j$ using adaptive moving average prediction and derive the relative error threshold η_{th}.
2: Use the sensed note duration to compute the tempo $\tilde{S}_j = T_j^{\text{note}} / T_j^{\text{sensed}}$.
3: **if** $\tilde{S}_j > \hat{S}_j$ **and** $|(\tilde{S}_j - \hat{S}_j)/\hat{S}_j| \geq \eta_{\text{th}}$ **then**
4: **return** The sensed attack is invalid (a false attack).
5: **end if**
6: Set the counter of missed notes $p = 0$.
7: Set $T^{\text{left}} = T_j^{\text{sensed}}$ for iterative missed note detection.
8: **while** $\tilde{S}_{j+p} < \hat{S}_{j+p}$ **and** $|(\tilde{S}_{j+p} - \hat{S}_{j+p})/\hat{S}_{j+p}| \geq \eta_{\text{th}}$ **do**
9: Mark the $(j+p)$-th note as detected (missed note attack).
10: Estimate the play duration of the missed note $\hat{T}_{j+p}^{\text{play}} = T_{j+p}^{\text{note}} / \hat{S}_{j+p}$, and add $\hat{S}_{j+p}$ into the tempo series.
11: Update $T^{\text{left}} = T^{\text{left}} - \hat{T}_{j+p}^{\text{play}}$.
12: Increase p to consider the next note.
13: Compute the tempo $\tilde{S}_{j+p} = T_{j+p}^{\text{note}} / T^{\text{left}}$.
14: Predict the expected tempo $\hat{S}_{j+p}$, and update η_{th}.
15: **end while**
16: **return** The sensed attack, which matches with the $(j+p)$-th note, is valid.

5. *MUSICSCORE*: EXPERIMENTAL EVALUATION

In this section, we conduct a thorough evaluation of *MusicScore*, developed on the iOS platform in Objective-C.

5.1 Professional Music Composition

As a professional music composition app for mobile devices, we first evaluate the quality of our music engraver, which serves as the cornerstone of *MusicScore*. The *MusicScore* engraver is designed to produce artistically pleasing music sheets, by strictly following traditional guidelines of music engraving. To examine its engraving quality, we carefully compare engraving outputs of *MusicScore* with hand-tuned engraving results [12]. In addition, we conduct a set of regression tests from LilyPond [10], which include a large number of scenarios a professional engraver needs to solve. For comparisons, regression tests are also run in Symphony for iOS, LilyPond, and libmscore. We discover that the *MusicScore* engraver produces a satisfying engraving output on mobile devices. For example, in the accidental and beaming tests, as shown in Fig. 13, *MusicScore* produces almost identical output as compared to LilyPond, while both Symphony and libmscore have severely violated basic engraving rules (marked by red rectangles): (1) accidentals are too far away from notes; (2) the ledger line is unreadable; (3) the slope of the beam is incorrect; (4) the partial beam is in a wrong direction. Clearly, these engraving errors may lead to a lack of comfort as professional musicians are composing or playing.

Since *MusicScore* targets mobile devices with limited computational capabilities, the efficiency of the engraver is also critically important. Using powerful instrumentation tools from Xcode, we evaluate in detail the performance of the *MusicScore* engraver on-the-fly in terms of its CPU usage, in both static and dynamic engraving scenarios. As summarized in Table 4, when *MusicScore* is

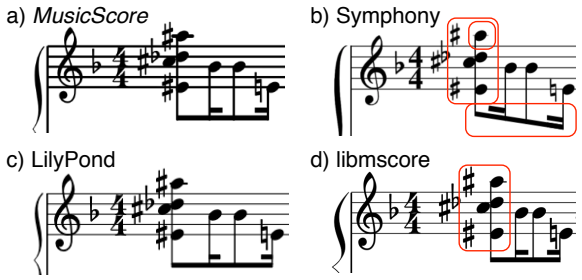

Figure 13: A comparison of engraver output among *MusicScore*, Symphony, LilyPond, and libmscore. Red rectangles indicate incorrect engraving results.

running on a less powerful first generation iPad, the average CPU load is no more than 25% for typical usage, such as digital music sheet display and music composition routines. The CPU load exceeds 50% only during engraving preparation (usually no more $2-3$ seconds at the loading stage), or when the score is being edited intensively. Considering the inherent complexity of professional engraving, we believe *MusicScore* has successfully achieved a satisfactory level of performance on mobile devices.

Table 4: Average CPU load of the *MusicScore* engraver.

Workload	Description	CPU (%)
Engraving preparation	One-time engraving preparation when loading a score	68.5 ± 10.3
Music sheet page turning	Rendering a full page of the music sheet	5.5 ± 3.9
Moderate interactive editing	Using multi-touch gestures to add notes, change properties, and organize the score	13.9 ± 7.1
Intensive interactive editing	Adding a large number of complicated chords with the virtual keyboard, and etc.	53.2 ± 9.3

5.2 Assessment of Play Progress Tracking and Music Performance Evaluation

In addition to features designed for professional users, we strive to provide casual users with the best possible features in *MusicScore*, such as play progress tracking, automated page turning, and live performance evaluation. To evaluate the accuracy of algorithms in *MusicScore*, we conduct our evaluations in a progressive manner, with aforementioned two groups of piano players.

First, we focus on our proposed note detection algorithms based on audio signals. Given digital music scores from the Canadian Royal Conservatory of Music Grade 3 syllabus, by using QIFFT based spectrum analysis to sense notes played in live performance, it is shown that *MusicScore* can achieve an overall detection accuracy of 98.6%, in terms of the note pitch. In Fig. 14, we further summarize the identification accuracy along with the 95% CI, in different octaves. The results reveal that notes in low octaves with low f_0 may be mistakenly identified as wrong notes, due to the limited input frequency response on iOS devices.

When note attack timing is considered, we measure the accuracy of our algorithm using two metrics: the ratio of missed attacks (R_{m}) and the ratio of false attacks (R_{f}), both over the total number of true note attacks. As shown in Table 5, the accuracy is reasonably high when the tempo of the music is slow, but R_{m} and R_{f} increase remarkably as the tempo becomes faster, particularly for

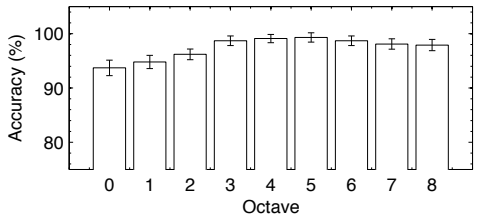

Figure 14: The accuracy of note pitch detection.

inexperienced players. This confirms the need for an intelligent note matching mechanism — to be later evaluated — to identify both false and missed attacks.

As the music sound analysis lacks sufficient temporal resolution for accurate performance evaluation, vibration signals are analyzed in *MusicScore*. We evaluate the improvement of temporal resolution by comparing the note attack timing detected by audio and vibration signals, respectively. As listed in Table 6, with the assistance of vibration signals, 46.4% of note attacks can be detected $100-200$ ms ahead of audio-based attack detection. Given that the temporal resolution of QIFFT in *MusicScore* is around 200 ms, the accuracy of note timing has been significantly improved.

Table 5: The accuracy of audio-based note detection, evaluated by a pair of metrics $(R_{\mathrm{m}}, R_{\mathrm{f}})$. Both experienced and inexperienced players are invited to play music pieces at different tempos.

Music Tempo	Players	
	Experienced	Inexperienced
Slow	(0.5%, 0.3%)	(2.8%, 3.2%)
Moderate	(3.2%, 5.6%)	(10.4%, 8.7%)
Fast	(12.3%, 8.9%)	(17.9%, 14.6%)

Table 6: Statistics of attack timing difference between a note detected from vibration signals and the same note from audio signals. A positive value implies that a note is first detected by vibration signals.

Time Diff. Range	%	Time Diff. Range	%
Note not detected by vibration signals	1.5	[100 ms, 150 ms)	24.8
< 0 ms	0.8	[150 ms, 200 ms)	17.8
[0 ms, 50 ms)	15.1	≥ 200 ms	10.7
[50 ms, 100 ms)	21.6	Note not detected by audio signals	7.7

By integrating audio and vibration processing modules with intelligent note matching using tempo estimation, we can then assess the accuracy of the performance evaluation feature in *MusicScore*. Compared to Table 5, we measure the accuracy using R_{m} and R_{f} after intelligent note matching. As shown in Table 7, a significant degree of accuracy improvement is observed after missed note attacks and false attacks have been carefully handled by our proposed algorithm based on tempo estimation. We observe that, for experienced players who have a better control of tempo, error ratios have no strong correlation with the music tempo; and the total error ratio is lower than 10% in general, even for inexperienced players.

Since all play progress tracking and performance evaluation algorithms are running in real time, it is critical to evaluate their per-

Table 7: The accuracy of performance evaluation, evaluated by a pair of metrics $(R_{\mathrm{m}}, R_{\mathrm{f}})$.

Music Tempo	Players	
	Experienced	Inexperienced
Slow	$(< 0.1\%, < 0.1\%)$	$(0.3\%, 0.7\%)$
Moderate	$(0.9\%, 1.1\%)$	$(2.4\%, 2.5\%)$
Fast	$(1.3\%, 1.0\%)$	$(4.2\%, 3.7\%)$

formance when running on mobile devices. All our proposed algorithms must meet the deadline for timely signal processing, such that users experience no uncomfortable lags when using *MusicScore* to track a live performance. As shown in Fig. 15, we measure the average CPU times consumed in every iteration of our algorithms. With respect to audio signals, one iteration takes a frame of the audio stream as input, while one iteration of vibration signal processing reads 3 values from the 3-axis accelerometer. Our measurement results show that both iPhone 4S with a faster CPU and iPad (first generation) with a slower CPU can finish our signal processing tasks within 1 ms, which can be attributed to the simplicity of our algorithms, as well as the fully optimized implementation using the iOS `vDSP` framework.

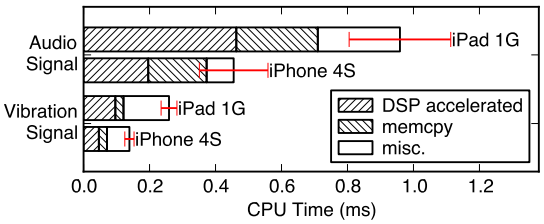

Figure 15: CPU times consumed for signal processing.

6. RELATED WORK

In the existing literature, pitch analysis and note attack detection algorithms have been widely studied. Harmonic product spectrum based algorithms proposed to extract the note pitch by finding the maximum coincidence of harmonics among Fourier transform outputs [7], while Master proposed to extract the pitch using Cepstrum along with the harmonic product spectrum [8]. In addition, an autocorrelation function is applied in the time domain to identify the fundamental frequency, which reveals a strong autocorrelation pattern [4]. With respect to note attack detection, Scheirer proposed a tempo analysis algorithm using bandpass filters and banks of parallel comb filters [13]. Goto used a simple peak-picking algorithm to roughly detect note attacks for music beat tracking [6]. Different from these algorithms which have no *a priori* knowledge about the signals to be analyzed, *MusicScore* is targeting a different scenario, where algorithms are carefully designed to analyze signals with the knowledge of the score being played. As we have made full use of available information, detection errors have been greatly reduced, so that users can enjoy a smooth experience with *MusicScore*. Another closely related research area is real-time score following, which aims to synchronize positions between the audio and the digital score. Arzt and Widmer proposed to use online dynamic time warping with simple tempo models to follow the music [2], while Montecchio and Cont modeled the system as a state tracking problem with various probabilities carefully trained [9]. Distinguished from these works, *MusicScore* can not only track the live performance, but also identify wrongly played notes or tempo. Furthermore, these existing score following algorithms require either time-consuming searching or complicated training, *i.e.*, they are not suitable for mobile devices with limited computational power.

7. CONCLUDING REMARKS

This paper presents the design and implementation of *MusicScore*, the first professional grade music composition application on the iOS platform. *MusicScore* is designed for both professional and casual users, as music scores can not only be composed and edited, but also be used for live performance tracking and evaluation, all taking place in real time. Our design in *MusicScore* features a number of salient highlights: a professional-grade interactive engraver, a collaborative sensor solution that mitigates the limitations of the audio recording system on mobile devices, new attack detection algorithms that use both audio and vibration signals, as well as a note matching algorithm with tempo estimation. All combined, *MusicScore* is shown to perform well as it tracks live performance of both experienced and inexperienced players, while staying comfortably within the limits of the computational power on mobile devices.

8. REFERENCES

[1] M. Abe and J. O. Smith, III. Design Criteria for Simple Sinusoidal Parameter Estimation Based on Quadratic Interpolation of FFT Magnitude Peaks. In *Audio Engineering Society Convention*, 2004.

[2] A. Arzt and G. Widmer. Simple Tempo Models for Real-Time Music Tracking. In *Proc. Sound and Music Computing Conference (SMC)*, 2010.

[3] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, 4 edition, 2008.

[4] A. de Cheveigne and H. Kawahara. YIN, A Fundamental Frequency Estimator for Speech and Music. *J. Acoust. Soc. Am.*, 111(4):1917–1930, 2002.

[5] H. Fletcher, E. D. Blackham, and R. Stratton. Quality of Piano Tones. *J. Acoust. Soc. Am.*, 34(6):749–761, 1962.

[6] M. Goto. An Audio-based Real-Time Beat Tracking System for Music with or without Drum-Sounds. *Journal of New Music Research*, 30(2):159–171, 2001.

[7] Y. Li and D. Wang. Pitch Detection in Polyphonic Music Using Instrument Tone Models. In *Proc. IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages 481–484, 2007.

[8] A. S. Master. Speech Spectrum Modelling from Multiple Sources. Master's thesis, University of Cambridge, 2000.

[9] N. Montecchio and A. Cont. A Unified Approach to Real Time Audio-To-Score and Audio-To-Audio Alignment using Sequential Montecarlo Inference Techniques. In *Proc. IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pages 193–196, 2011.

[10] H. W. Nienhuys and J. Nieuwenhuizen. LilyPond, A System for Automated Music Engraving. In *Proc. Colloquium on Musical Informatics*, 2003.

[11] T. Pinch and F. Trocco. *Analog Days: The Invention and Impact of the Moog Synthesizer*. Harvard University Press, 2004.

[12] T. Ross. *The Art of Music Engraving and Processing*. Hansen Books, 1970.

[13] E. D. Scheirer. Tempo and Beat Analysis of Acoustic Musical Signals. *J. Acoust. Soc. Am.*, 103(1):588–601, 1998.