# Challenges for Livecoding via Acoustic Pianos

**Steve Tanimoto**
University of Washington,
Seattle, USA
tanimoto@uw.edu

## ABSTRACT

The activity of "music performance livecoding" involves a human programmer writing or modifying a computer program that is creating music, typically in front of an audience. Livecoding input has typically employed computer keyboards. Sometimes, a MIDI keyboard has been used. Yet there are potential contexts in which non-MIDI piano keyboards make sense. In such a context, the only signals serving as input to the computer are acoustic. They mediate or mediate changes to a computer program that is being controlled and edited by a musician/coder. Reasons to consider this include performance venue constraints, musicians' preferences, potentially greater input bandwidth, and new music genres. This paper reviews some of the relevant literature and analyzes several problems related to the facilitation of livecoding via non-electronic pianos. It then addresses the challenges of designing and implementing a software toolkit in which to further study these problems. As a case study in design, an experimental software configuration called Piano Python is described. The paper touches on technical issues, human factors, aesthetic criteria, and artistic and educational possibilities of livecoding via acoustic pianos.

## INTRODUCTION

### Live Coding

Live programming for musical performance ("livecoding") has an established following (Blackwell and Collins, 2005) and an increasing variety of approaches have been tried (Collins, ed, 2015). In its most common format, a human programmer performs by writing and editing computer code in front of an audience, and the code, which is constantly being executed by the computer, creates music and/or other sensory experiences for the audience. The interface between programmer and computer in this context is conventional: a keyboard, mouse or trackpad, and display screen. In some cases, the livecoder is half of a duo, in which the other half plays a conventional musical instrument such as a cello (Chafe and Kermit-Canfield 2014). It is also possible for the computer to respond to the conventionally produced instrumental audio after sensing it with "machine listening" (Collins, 2015).

The role of the computer in livecoding is usually the central one in terms of both mediating the music's representation and rendering the audio. The music can be represented as a changing textual program in some language, such as Overtone (Aaron 2017a), Sonic Pi (Aaron 2017b), Impromptu (Sorensen 2017), or TidalCycles (McLean 2014), and the rendering may be accomplished with powerful synthesizers such as SuperCollider's synthesis server (McCartney 2004).

In some cases, livecoded music has been composed on the computer and rendered by an electro-acoustic piano such as the Yamaha Clavinet. Such performances can take advantage of the rich acoustical characteristics of the piano, however traditional that sound may be. Special effects are possible with such a setup beyond the traditional sounds; for example, the piano can be made to "talk" through a process of spectral approximation (Kirn 2009).

Piano keyboards are beginning to be used as input to computers for livecoding. The CodeKlavier system (Noriega and Veinberg 2017) uses a MIDI keyboard simultaneously as a musical instrument (electric piano) and a textual input device to a computer.

**Why Use an Acoustic Piano?**

Six reasons for livecoding with an acoustic piano are suggested:

R1. The piano is a high-bandwidth interface for human expression. The number of bits per second that can be transmitted from brain to device (assuming that information could be accurately encoded as bits) is at least comparable to, if not greater than, the rate for ordinary computer keyboards. It is often the case that piano music includes chords, which are groups of notes that are sounded simultaneously. This corresponds to a degree of parallelism in the communication channel that goes beyond what a computer keyboard offers, where normally one character key is pressed at a time, and when more than one are pressed at a time it is to use the shift key or control key together with the character key. The 88-key acoustic piano keyboard has roughly as many keys as the typical computer keyboard, but they are designed so that, in principle, any subset of them can be played at the same time. The limitation here is only the physiology of the hands and arms. It is common in piano music to have chords of 5 or more notes; some chords may even involve more than 10 notes, say when the thumbs (or any other fingers) are used to hit two keys each, as sometimes happens in both jazz classical music. It is not only the sequence of note subsets in time that communicates information in piano performance; it is also the timing of those note subsets that communicates information. While rhythmic communication through computer keyboards is also possible, there is no real tradition or convention for doing so. Finally, there are additional dimensions of expression in the piano that do not occur with computer keyboards: dynamics (points along the loud-soft continuum), and note sustaining (staccato, legato, and points in between).

The rate at which information can be expressed through a piano is partially dependent upon the speed with which notes can be played in sequence. These rates have been studied from the standpoint of virtuosity and what a composer might reasonably (or unreasonably) demand from a pianist. Classical pieces containing notably rapid note sequences include the Chopin “Winter Wind” Etude (Opus 25, No. 11) with up to 13.8 notes per second, and the Prokofiev First Piano Concerto first cadenza with up to 14.67 notes per second (Collins 2002). Since the note sequences in these passages have regular structure, the fact that they can be played this quickly does not generally mean that an arbitrary sequence of notes can be played so fast. Nonetheless, it suggests that the limit at which a pianist can enter a sequence of notes is under 20Hz, unless special cases such as glissandi are considered. Another estimate of the limiting speed for piano note sequences is 10Hz: “The lower limit for meter, that is, the shortest interval that we can hear or perform as an element of rhythmic figure, is about 100 milliseconds (ms).” (London 2004, Ch.2).

The rates of text entry, in words per minute, using QWERTY keyboards vs piano keyboards (using a special PianoText method) appear to be comparable at 80 wpm, and the piano keyboard offers the possibility of further gains through additional innovations with the mappings from keyboard events to text (Feit and Oulasvirta 2014).

R2. The piano, being already a musical instrument, can participate in the production of sound as part of the performance. Not only is it an instrument, but it uses a highly evolved technology of piano-based sound production and piano making (Morris 2006; Barron 2006). Music involving an acoustic piano and livecoding together (with pianist alternating piano-keyboard and computer-keyboard input) has already been composed and performed (Collins and Veinberg 2015). Now the question is whether the computer keyboard can be eliminated.

R3. The piano keyboards and actions of traditional acoustic pianos are ergonomically as well as musically advanced. A “real piano action” feels better to accomplished pianists than even fairly expensive MIDI keyboards. It is usually better adapted to technically demanding music such as rapid note sequences and passages in which high-contrast dynamics are required. The feel of a high-quality real piano action is a mechanically (and perhaps potentially also semantically) richer and deeper one than the feel of even an expensive, ergonomically optimized computer keyboard. If the keystrokes of such piano keyboards could be sensed accurately enough (as mentioned in R1), it might be possible to rival the coding bandwidth of conventional keyboards, at least perhaps for specialized computer languages.

R4. Many musicians who might wish to incorporate computation into their performances already know how to play a piano, and many are fluent in the use of piano keyboards. They are ready to use their fingers, hands, and even full forearms for special effects, and a traditional piano also typically has a damper pedal (for sustaining all currently played notes), and an *una-corda* (one-string, or soft) pedal which can affect the perceived tuning and timber of sound, as well as a third pedal – the *sostenuto* pedal which although less commonly used than the other two, offers an additional means of sustaining subsets of notes while others are played without sustaining, e.g., staccato. A reasonably accomplished pianist (and there are many of them) is prepared to engage through this interface in a high-bandwidth fashion.

R5. Many of the venues where musical performances take place already have acoustic pianos. This is especially true for concert halls and other large indoor venues.

R6. Pianos offer an interface that is accessible to many blind people. The combination of kinesthetic and acoustical feedback has helped blind musicians master the interface. A computer keyboard, on the other hand, is dependent upon additional technology, such as screen readers, to make it accessible to a blind person. (This reason is not an argument for avoiding MIDI keyboards, but simply for using piano keyboards.)

With these reasons as the first motivation, we make a set of assumptions for a new genre of live coding.

#### Acoustic Piano-Based Livecoding

Here are our grounding assumptions for the design discussion that follows.

A1: An acoustic piano is the only input device that the livecoder touches during the performance.

A2: The computer's information about what keys and pedals are being played arrives by micro-phone, and not via a MIDI stream. (That's not to say no MIDI stream can exist, but if there is one, that stream is itself derived from the acoustic information coming from the computer's microphone. Some pianos have both, but using their MIDI capabilities violates this assumption.)

These two assumptions raise the following questions.

Q1. When does the performance begin and end? A voice command could be used, or the performer might need to touch something other than the piano when preparing for the performance and when cleaning up after the performance. (Alternatively, an extra switch or sensor could be placed on the music stand or on a pedal.) The computer system that is listening and responding to the piano must be set up and turned on. Afterwards, it should be turned off.

Q2. Although in an ideal world, the computer system will be able to “hear” exactly what keys are being played and at what intensity and for how long, this can be a technically challenging issue, especially if the piano is out of tune or has other acoustic particularities. If it has bad keys or bad strings, then this can add to the technical challenge of interpreting the performer’s intentions. How should the system be designed in order to be robust in the face of such challenges? One answer is that the system should be adaptable to the situation. The language itself should include provisions for facilitating effective communication even when the channel is impaired to various degrees and in various ways. We explore the challenges further in the next section.

## CHALLENGES

Whereas a MIDI keyboard may, relatively easily, transmit accurate representations of key presses, including note timing and velocity, an acoustic piano keyboard has not been outfitted with sensors for such information. This information, in order to be obtained, requires an audio analysis process. The success of analysis depends upon multiple factors, and these factors may vary widely from one physical setting to another.

C1. The analysis results depend on the quality of the piano and the extent to which it is in tune. If the piano is out of tune, pitch tracking software may find the pitch assignments to note events ambiguous.

For example, except in the lower pitch range, a piano key normally activates a hammer that strikes three separately tuned sections of piano wire. If these are out of tune with each other, the assignment of a unique pitch value to the ensemble of the three pitches is an over-constrained problem. If piano strings are broken, unevenly voiced, or unbalanced in any other way, the pitch-tracking challenge may be substantial.

C2. This leads to a second challenge: the quality of the note-detection and description software may not be sufficient to get correct pitches consistently and efficiently across much of the range of the piano scale, particularly in noisy environments. Polyphony presents additional challenges for this computation, not only due to the additional computational burden of detecting multiple simultaneous pitches, but the overlapping harmonics of piano-string sounds make it difficult to sort out what combination of piano keys have been pressed to produce a particular spectral composite, even if the frequency analysis of the sound is completely accurate (Benetos et al 2013).

The state of the art in pitch tracking, onset detection, and note identification have advanced significantly in recent years. For example, the pitch component of note identification can be performed with polyphonic music with reasonable accuracy on a modern-day laptop computer in real time using spectral features inferred using machine learning (Thickstun et al 2017a,b). However, in a livecoding context, there may be background noise and/or instruments in addition to the piano (e.g., synthesized sound) that may complicate the machine listening process.

C3. The next challenge is that this kind of audio analysis requires a significant level of computational power. Handling the wide range of pitches of a standard 88-key piano (from 27.5Hz of A0 to 4186.01Hz of C8) means not only having a high enough sampling rate (theoretically at least 8372Hz), but processing those samples with filter banks and classifiers in real time. Getting temporal accuracy with respect to note onsets also consumes cycles. The presence of noise can mean that computationally simpler methods will not work. However, as processors get faster, and GPUs become more practical for livecoding in terms of energy use and cost, the computational demand may be met.

C4. Finally, the fact that several of these conditions will vary from one physical setting to another (different pianos, different venues' acoustical responses, different computers, and perhaps different skill levels of performers) suggests that a livecoding system that supports acoustic pianos as sole input devices needs to offer means to adapt to varying effective bandwidths of input.

## PIANO PYTHON

#### Components Needed

In order to explore some of the challenges, an experimental configuration of software has been set up. This can be regarded as a first step in a design that will allow the greatest flexibility within the constraints of the (a) acoustic piano that might happen to be available at a random venue, (b) the music recognition technology that is readily available, and (c) whatever processor is running the software.

Most of Piano Python is not yet implemented, but these parts are discussed here in order to present a more coherent view of the possible system. Here are some pieces. They are intended to provide offerings to the performer/livecoder at multiple levels of abstraction, and to allow for different levels of success with the technical challenges of the audio recognition.

An "Emacs-complete" code editor for arbitrary text (i.e., any possible Python program with or without bugs). "Emacs-complete" means that file operation or buffer operation that can be performed with emacs can also be performed with this editor. A simple way of achieving this is to actually run emacs, but drive it via a differently produced input stream than via a QWERTY keyboard. The piano note-pattern to Unicode character mappings can vary depending on the technical capabilities in a particular situation.

A Python program-inference module. When the technical constraints reduce the editing bandwidth for livecoding, one of the possible adaptations of the system is to infer, from the current text buffer

contents, the “closest legal” Python program. In a livecoding context, a program is typically running continuously, as it is being edited. There are two parts to this process – inferring a dynamically changing program (a sequence of programs) and managing a continuing execution that corresponds to the sequence of programs. The program-inference module deals with the first part, but with the second part under consideration. The notion of textual edit distance is well known and there are efficient algorithms for computing the edit distance between a pair of text strings or text files using the dynamic-programming algorithm paradigm. It is not necessary to use the common edit distance for program inference, because such distances can easily be redefined to better take into consideration the important properties of texts that represent programs. Provided that the new edit distance can still be computed using dynamic programming, the inference strategy can be made tractable computationally. The word “legal” for Python programs could possibly refer to standard Python syntax; however, it is more likely to be useful for us if it refers to a narrow subset of Python programs that are relevant to the livecoding context, such as those programs that specify the behavior of a music sequencer or an audio effect.

A Python continuous-execution engine. This is perhaps the most ambitious technical component of the proposed full Piano Python system. It is a Python execution engine (compiler-interpreter) that is capable of performing a coherent execution that corresponds to a possible long sequence of different Python programs. The key technical assumption is that the sequence of programs is “continuous”; this means that from one program in the sequence to the next, the difference consists of either (a) a statement dropped, (b) a statement added, or (c) a statement replaced. When a new program is appended to the sequence, the execution of the old version of the program either continues normally, if execution is outside of the changed statement, or it is interrupted and restarted at the closest relatively safe place if execution was within the deleted or changed statement.

The continuous-execution engine is a component that need not be particular to Piano Python, but might be useful in any Python live-programming context (Tanimoto 2012).

**Implementation of Piano Python**

This section describes the implementation of the prototype version of Piano Python as of the ICLC 2017 submission date.

The acoustic recognition technology consists of a simple encapsulation of the Pure Data “sigmund~” object (Puckette et al 1998) within a patch that recognizes specific pitches and sends messages via sockets to a running Python program called pianoPython.py. That program translates these note events into specific actions that may be taken in order to edit and run additional Python code. This code, in turn, can generate music that accompanies, or is accompanied by the piano. That music is actually rendered by Pure Data in response to another channel of messages sent from the running Python code back to Pure Data. The acoustic analysis portion of this Pure Data patch is illustrated in Figure 1. In practice, the performance of this analysis has been unreliable and sensitive not only to the values of the sigmund~ object creation parameters, and the peculiarities of the particular acoustic piano, but also something about the timing and execution of the system. Particular notes played are recognized in some runs of the program and not in others, even when using recorded test data as the piano input. This illustrates challenge C2 quite clearly. A more sophisticated analysis of the piano sound might employ a filter bank and a neural network that could accurately translate a polyphonic input sample window into a weighted sum of piano notes pressed – i.e., MIDI note sets with velocities.

There are essentially three parts to the Python side of the implementation: (a) the framework itself, implemented in a file called pianoPython.py, (b) an editor that maps acoustical patterns to commands and text, and (c) a set of library routines that support the generation of music during live coding. The editor in part b includes a text object that contains the Piano Python source code being edited by the live coder.

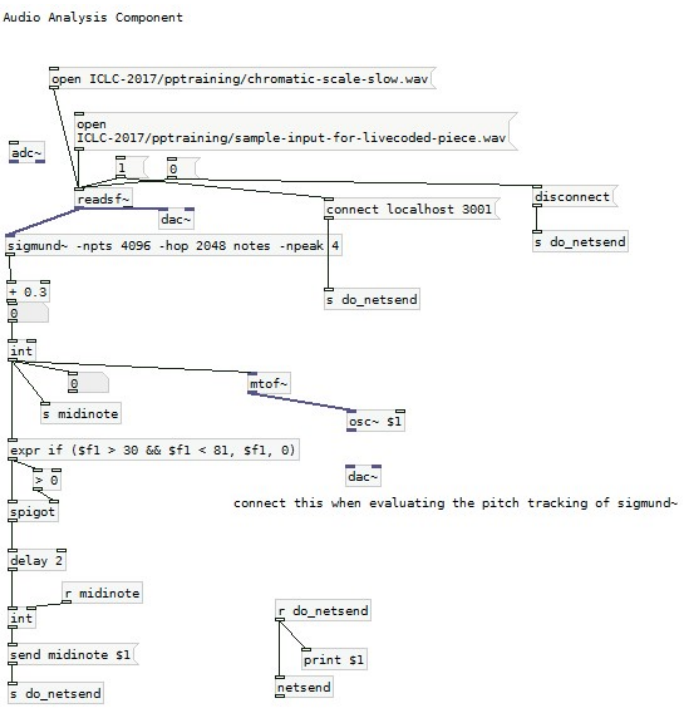


**Figure 1**. The audio-analysis portion of Piano Python, implemented as a Pure Data patch.

## EMACS-COMPLETE EDITING VIA ACOUSTIC PIANO

In this section, the challenge of text-editing from a piano is discussed, and several methods suggested, each with its own advantages and disadvantages. If a MIDI keyboard is used instead of machine listening, and if the musical byproduct of keyboarding is not of interest, then systems such as PianoText (Feit and Oulasvirta 2014) can possibly achieve high text-entry efficiency (up to 80 words per minute). However, without the MIDI capability, of prime importance is the quality of acoustic recognition. We distinguish four levels: monophonic without pitch or volume variation (simply sound on and sound off), monophonic with pitch but not volume, polyphonic with pitch but not volume, and finally polyphonic with pitch and volume. There are four more combinations of these three attributes that we'll consider briefly after our consideration of these four principal cases. But let us first suggest means of using the piano for each of these four principal situations.

#### Monophonic without Pitch or Volume

The suggestion here is to accomplish editing by using the Piano to send Morse-Code messages to the computer. This is a standard encoding, which although unfamiliar to most pianists, has a rich history of usage and methods for learning it. In a live-coding context in which the performer is playing piano music and live coding at the same time, one particular piano note might be reserved for use as the Morse Code tone. Part of the challenge of performing would be not only playing and coding simultaneously, or in some interleaved fashion, but making the coding note somehow sound all right together with the rest of the performance.

#### Monophonic with Pitch but not Volume

Characters are indicated by note patterns, separated by occurrences of the special note: e.g., middle C. Each note pattern is a subset of the Midi notes 49-65 (piano keys C# to G, just above middle C). When played serially, the notes are played in the sequence from highest pitch to lowest. Since there are 7 notes in this subset (G, Gb, F, E, Eb, D, C#), there are 2^7 different characters that can be represented this way. The representation gives shorter patterns to more frequently occurring letters, such as vowels. Also, lower pitches (within the 7-note range) are preferred for the vowels, because the end of the pattern can be signified by either an occurrence of the reference note Middle C or by

the playing of a pitch higher than the last pitch in the currently-finishing pattern. Such a pitch is the first note of the next pattern.

A restriction to a range such as this can prevent the need for the right hand to jump to other sections of the keyboard. Thus it may lead to faster input than schemes requiring jumps. It may also simplify the machine listening by reducing the number of filters in a filterbank needed for the task. Finally, it frees up other parts of the keyboard to take on other roles, such as control, “macros” that translate into larger text units, etc., though of course the machine listening must then listen for those notes, too. The PianoText system allows a note pattern to be played in any octave and mean the same thing, which is another approach to avoiding the need for hand jumps (Feit and Oulasvirta 2014).

#### Polyphonic with Pitch but not Volume

When the audio analysis software is capable of discerning multiple notes at a time, each character is indicated by one chord, using the same subsets as in the monophonic method above. Note that this scheme would assign to one of the 128 characters a 7-note chord, and it would assign to seven of them 6-note chords. These may be problematic both for both the performer and the acoustic recognizer. The performer might have difficulty with either a 7-note chord or a 6-note chord if only one hand is used to play the chord (the other hand being used for direct musical purposes). Thus these patterns should be avoided and handled as special characters in a sort of "shift" subset.

#### Polyphonic with Pitch and Volume

It is an open question whether the performer’s modulation of volume can be employed to beneficial effect in the expression of Unicode characters through a method that extends the above-described polyphonic method without volume modulation. There are technical challenges for the performer in controlling the volume of multiple notes independently and simultaneously. Easier is to control the overall chord volume, with each note within the chord receiving the same force and velocity during keypresses. While volume modulation might permit the transmission of another bit or two when using chords to express Unicode characters, the need to use volume for that purpose would make volume no longer available as a means of direct musical expression during a performance that combines live coding and conventional piano performance.

#### Special Patterns for Piano Python Reserved Words and Special Identifiers

The previous method can be extended to one in which a special shift-key note is used to indicate that the next chord represents a Python token. There is a token for each of the more commonly used reserved words of the language, including the following:

> def, return, for, if, in, try, except, class, while, True, False, break, pass, import, global, not, continue

By using single chords to represent complete tokens, the effective bandwidth for the coding may be approximately doubled.

#### Four Additional Cases

Because we are considering three binary variables (monophony/polyphony, pitch variation/no pitch variation, and volume sensitivity/no volume sensitivity), and we have considered four of the possible combinations already, there remain four more. Now we briefly consider those. The monophony+no pitch+volume combination simply adds volume sensitivity to the first case considered above. Volume, measured at or shortly after the attack of a piano note may be used to code for a text attribute (e.g., upper vs. lower case) or in combination with a rhythmic pattern to identify a letter, morpheme, word, or phrase. The monophony+pitch+volume combination simply adds the use of volume to the second principal case above. There is some danger that volume, if given too much significance, could become a source of communication errors, as it can be difficult to control volume accurately in the context of a rapid sequence of different notes. The case of polyphony without pitch variation or volume sensitivity can be interpreted as having two or more monotonic voices. They have different pitches,

but none of them can vary their own pitch. Their volume variations are only detected as on or off. The remaining combination is polyphony without pitch variation but volume variation. This simply adds volume sensitivity to the previous case. There may be design justifications for any of these, in a particular instance, if only to support an etude that explores the constraints of these choices.

## DISCUSSION

### Applications

A variety of possible "uses" for a Piano-Python-like system can be imagined. Here are a few. Piano music offers an opportunity for a form of musical steganography. Steganography is typically illustrated with textual or visual messages, where there is an easily observed carrier message or pattern, but there is also a hidden message, of which an uninformed observer is unaware. By use of special analysis, the hidden message can be revealed. For example, in the text, "Greetings on this occasion being a special event," taking the first letter of each word, we get "Gotobase" or "Go to base." A pianist/coder may play a piece of music that is on one level agreeable music, but on another level, code that a computer is using and deciphering, possibly for a purpose such as revealing a secret message, but possibly also for the purpose of enhancing the performance though control of additional sound or of visual stimuli. Such methods are related to ciphers, which have been used in music for many years, sometimes by well-known composers such as Robert Schumann (Sams 1966).

Musicians sometimes need to conduct from the keyboard (Schonberg 1987). For example, Youtube hosts a video of Karl Richter conducting the Brandenburg Concerto #5 from the harpsichord (Richter 2010). The conducting often serves to synchronize the ensemble or to communicate intended dynamic changes. If the instrument (harpsichord or piano) can be used in new ways, with the help of a computer, perhaps needs like this can be satisfied in new ways. Certain cues may be codes for synchronization acts, or acts to transmit intended changes of dynamics.

The most obvious application of a Piano Python sort of system is livecoding for accompanied piano. This means the production of piano-plus-computer music, which is produced by a single performer. While most commercial electric pianos today offer automatic accompaniment in the form of drum tracks or bass lines, the ability to use live coding at the same time could enrich the range of expressiveness. If the keyboard is already electronic, then the challenges of acoustic analysis could be bypassed, and main challenges might be for the performer/coder to learn how to code and play at the same time. Anne Veinberg has led the way with her piano and livecoding performance using the CodeKlavier system (Noriega and Veinberg 2017).

Another reason to explore livecoding with acoustic pianos is to develop a new niche of machine musicianship in the sense of Rowe. Creating systems for this can further human understanding: "Designing computer programs that will recognize and reason about human musical concepts enables the creation of applications for performance, education and production that resonate with and reinforce the basic nature of human musicianship." (Rowe 2001).

Live programming of video effects is another possible application. This has been done by others in the context of livecoded musical performance to which the video is added, but with the acoustic piano as input, and if the challenges discussed earlier can be overcome, the possibilities become richer.

Programming with the piano as input device, disregarding the music itself, is another possible application. We can imagine a situation when a MIDI keyboard is not available, and the programmer prefers a music keyboard to a QWERTY one.

A rather specific type of performance is a takeoff on the art of playing piano music for silent movies. Whereas music typically had to be added to silent films as an afterthought (either onto a soundtrack later, or live in a movie theater), the advent of livecoding from a piano could put the pianist in the controlling role. The pianist/livecoder could be either cueing movie clips, or more generally coding the unfolding of the story, perhaps by using piano gestures and livecoded expressions to play a game or a role in a multiplayer game.

Finally, we can consider one more possible application: cueing (or selecting or adapting) of lecture/tutorial messages from the piano keyboard. This might be appropriate for music appreciation lectures.

There undoubtedly are others. Any application in which traditional piano playing is to be combined with computer-based activity is a potential situation where some of the techniques described in this paper could be relevant.

**Implications for Livecoding**

The Piano Python design, apart from its implementation, raises two notable issues for livecoders. One issue is whether a display monitor is required or not and if not, what to do about editing feedback. Editing a program without being able to see the text will change the nature of a livecoder's experience, whether or not the audience can see the text (which is another issue). Critics of the approach may rightly ask that if a visual monitor is required, then why not have the whole laptop there, including it conventional keyboard, to facilitate the livecoding, much as was used in the performance of "Type A Personality" (Collins and Veinberg 2015)? A possible way to provide editing feedback to the livecoder without a monitor is to use audio editing feedback. This could be done either using headphones, separating this audio from the audience, or it could be included in the audio mix that makes up the performance. Then, whether it enhances or detracts from the music will depend on how it's designed, and how well it blends in with the acoustic piano and the music from the execution of the program being edited. Designing editor audio thus poses the double challenge of achieving an effective editing communication means plus being tolerable or even beautiful as part of a music performance.

The second issue concerns the problem of making tolerable or beautiful music on the piano under the constraints of having to express programming changes simultaneously with the same instrument. Two approaches are the following: (a) let the code come first, and all music gets produced as a byproduct of the coding scheme; this may be fine for a while, but listeners may become bored when they find that the piano music's role in the performance is secondary to what the computer is doing; and (b) let the music come first, and embed the code in various ways such that the music is not impaired harmonically or rhythmically. The embedding need not be steganographic; the audience can be in on it. The code's music is part of the music, but it doesn't clash. One way to achieve this with, say, Morse-code style text input is to assign separate notes to the dit and the dah of the code. By embedding the dit- and dah-notes in a musical motif, the letter is expressed, but the motif can fit the music, much as a jazz musician improvised a melody line that must fit the chord progression. There is large design space for such schemes, and it could be considered as "meta composing" to work up such a scheme.

**Cognitive Considerations**

The cognitive processes involved in livecoding have been contrasted with those used in traditional musical performance (Sayer 2015; Magnusson and Sicchio 2016); livecoding places more importance on conscious planning and action, whereas traditional performance emphasizes perceptual-motor accuracy and the recall of memorized sequences. Piano Python, as presented above, offers a performer the opportunity to exploit both forms of cognition and activity. These can be mixed in different proportions. However, in the current system, the acoustical piano cannot be entirely eliminated; its influence could be further reduced if a MIDI keyboard is substituted thus giving the option to turn off the production of sound directly from key presses.

In addition to human cognition, let us mention the issue of machine cognition. A possible scenario is an ensemble in which a mixed group of humans and computer agents performs music. The humans may play various kinds of instruments or sing. The same machine listening techniques that listen to the piano may be adapted to listen to other instruments in an ensemble. This may lead to collaborative livecoding (of the computer agents) by the human acoustic instrument players and/or singers. Enabling coding from musical instruments can also be seen as a first step towards deeper machine participation mixed ensembles using automated musical cognition.

## CONCLUSIONS

The prospect of using a piano keyboard as both a vehicle for programming a computer and for computer-enhanced piano performance is both a means to develop a new genre of live coding performance and a means to study fundamental issues in live coding such as Unicode-complete editing, program inference, and what it means to continually execute a changing program. Performance in this genre helps exploit the highly evolved technology of the acoustic piano while incorporating cutting-edge ideas in livecoding. The question of how best to exploit traditional musical-instrument interfaces in the new role of computer interface has many variations, with not only keyboard instruments in center position, but woodwinds, horns, strings, percussion, and even the singing human voice as potential computer input interfaces.

**Acknowledgments**

Thanks to Sam Aaron, Alan Blackwell, and Thor Magnusson for encouraging my explorations into combining live programming with music performance, and to John Thickstun and Sham Kakade for assisting me with experiments in pitch detection. I would also like to thank the anonymous referees for their helpful suggestions.

## REFERENCES

Aaron, Samuel et al. 2017a. Overtone: Collaborative Programmable Music. http://overtone.github.io/ (accessed July 2017).

Aaron, Samuel. 2017b. Sonic Pi: The Live Coding Music Synth for Everyone. http://sonic-pi.net/. (accessed July 2017).

Barron, James. 2006. *Piano: The Making of a Steinway Concert Grand*. New York: Henry Holt.

Benetos, Emmanouil; Dixon, Simon; Giannoulis, Dimitrios; Kirchhoff, Holger; and Klapuri, Anssi. 2013. "Automatic Music Transcription: Challenges and Future Directions." *J. Intelligent Information Systems*, DOI 10.1007/s10844-013-0258-3, Springer Publications.

Blackwell, Alan, and Nick Collins. 2005. "The Programming Language as a Musical Instrument." *Proceedings of PPIG05* (Psychology of Programming Interest Group).

Brown, Andrew R, and Andrew Sorensen. 2009. "Interacting with Generative Music Through Live Coding." *Contemporary Music Review 28* (1): pp.17–29.

Chafe, Chris; and Kermit-Canfield, Elliot. 2014. "Koan for Cello and Live Electronics." Performance at https://www.youtube.com/watch?v=NUTx4u5k6J0&index=2&list=PLMJca0djxjP2R92ZF1s1baffWV78G8QVI, uploaded Mar. 20, 2014.

Collins, Nick. 2015. "Live Coding and Machine Listening," *Proc. of the International Conference on Live Coding, 2015*.

Collins, Nick; and Veinberg, Anne. 2015. "Type A Personality." A performance at ICLC 2015. https://www.youtube.com/watch?v=0fX0AymCtgA

Feit, Anna Maria; and Oulasvirta, Antti. 2014. "PianoText: Redesigning the Piano Keyboard for Text Entry." In *Proceedings of the 2014 Companion Publication on Designing Interactive Systems* (DIS Companion '14). ACM, New York, NY, USA, pp.129-132. DOI=http://dx.doi.org/10.1145/2598784.2602800

Kirn, Peter. 2009. "The Speaking Piano, and Transforming Audio to MIDI" CDM, an online creative magazine. http://cdm.link/2009/10/the-speaking-piano-and-transforming-audio-to-midi/.

Magnusson, Thor; and Sicchio, Kate. 2016. "Writing with Shaky Hands," *International Journal of Performance Arts and Digital Media*, Vol. 12, pp.99-101.

McCartney, James. 2004. "Rethinking the Computer Music Language." *Computer Music Journal,* Vol. 26, No. 4, pp.61-68.

McLean, Alex. 2014. “Making Programming Languages to Dance to: Live Coding with Tidal.” In proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modelling and Design.

McLean, Alex. (ed). 2015. *Proceedings of the International Conference on Live Coding*. Held at Leeds, U.K.

London, Justin. 2004. *Hearing in Time: Psychological Aspects of Musical Meter*. New. York: Oxford University Press.

Morris, Edmund. 2006. “Sound Factory.” New York Times, Oct. 1. http://www.nytimes.com/2006/10/01/books/review/Morris.t.html

Noriega, Felipe Ignacio; and Veinberg, Anne. 2017. “CodeKlavier Hello World.” https://www.youtube.com/watch?v=ytpB8FB6VTU, uploaded to YouTube.com on April 30, 2017.

Puckette, Miller; Apel, Ted; and Zicarelli, David. 1998. “Real-time Audio Analysis Tools for Pd and MSP,” *Proc. International Computer Music Conf*., pp.109-112.

Richter, Karl. 2010. Bach Brandenburg Concerto 5. https://www.youtube.com/watch?v=vMSwVf_69Hc, uploaded to YouTube.com on Jan. 20, 2010.

Rowe, Robert. 2001. *Machine Musicianship*. Cambridge MA: MIT Press.

Sams, Eric. 1966. “The Schumann Ciphers.” *The Musical Times*, May 1966, pp.392-399.

Sayer, Tim. 2015. “Cognition and Improvisation: Some Implications for Live Coding.” *Proc. of the International Conference on Live Coding, 2015*.

Schonberg, Harold C. 1987. “Music; Pianists at The Podium- Old Tradition, New Interest.” *The New York Times*, April 26, 1987.

Sorensen, Andrew. “What is Impromptu?” http://impromptu.moso.com.au/. (accessed July 2017).

Tanimoto, Steve. 2012. “A Perspective on the Evolution of Live Programming.” Proc. of the Workshop on Live *Programming*, San Francisco, CA.

Thickstun, John; Harchaoui, Zaid; and Kakade, Sham M. 2017. “Learning Features of Music from Scratch,” in *Proc. International Conference on Learning Representations* (ICLR), 2017.

Thickstun, John; Harchaoui, Zaid; Foster, Dean; and Kakade, Sham M. 2017. “Invariances and Data Augmentation for Supervised Music Transcription,” (to appear).

Wszeborowska, Anna. 2016. “Processing Music on the Fly with Python.” YouTube.com video, at https://www.youtube.com/watch?v=at2NppqlZok, uploaded to YouTube.com on Nov. 18, 2016.