Playing Fast and Loose with Music Recognition

Chris Greenhalgh, Steve Benford, Adrian Hazzard, Alan Chamberlain
School of Computer Science
The University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
{chris.greenhalgh, steve.benford, adrian.hazzard, alan.chamberlain}@nottingham.ac.uk

ABSTRACT

We report lessons from iteratively developing a music recognition system to enable a wide range of musicians to embed musical codes into their typical performance practice. The musician composes fragments of music that can be played back with varying levels of embellishment, disguise and looseness to trigger digital interactions. We collaborated with twenty-three musicians, spanning professionals to amateurs and working with a variety of instruments. We chart the rapid evolution of the system to meet their needs as they strove to integrate music recognition technology into their performance practice, introducing multiple features to enable them to trade-off reliability with musical expression. Collectively, these support the idea of deliberately introducing 'looseness' into interactive systems by addressing the three key challenges of control, feedback and attunement, and highlight the potential role for written notations in other recognition-based systems.

Author Keywords

Music recognition; notation; sensing systems; looseness; performance; H-metaphor; casual interactions; attunement

ACM Classification Keywords

H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION

The combination of sensors and machine intelligence to create various kinds of 'recognition technology' is enabling new modalities in human computer interaction such as speech, gesture and – the focus of this paper – playing music. These bring the potential for more natural and expressive interaction while also raising new challenges for HCI in terms of how people understand and control complex and sometimes unpredictable systems.

Our particular interest here lies in how system designers can empower musicians to incorporate music recognition technologies into live performance. We present a case study of iteratively developing a music recognition system that supports the composition and performance of musical codes. These are fragments of music that can be played by a musician during a performance with varying levels of expression and disguise, so as to trigger digital interactions such as the system playing additional parts, controlling audio effects, triggering visual media, or communicating with other musicians or even the audience. Specialist software exists that will allow technically proficient expert users to do this (e.g. in Max/MSP with suitably crafted patches). Our goal is to create a general tool to support a far broader range of musicians possessing more everyday musical competencies, from amateurs to professionals.

Our overall approach is one of evolutionary prototyping of a complete functional system, developed in partnership with users, as a way to jointly explore and reveal challenges and principles for the system itself, its application and its interaction. In our case, we collaborated with a diverse group of twenty-three musicians in four workshops to iteratively develop, explore and reflect upon two major iterations of our system, for composing and recognizing musical codes.

Rigorous reflection on this process reveals how the system was progressively engineered to support musicians in negotiating varying degrees of "looseness" with regard to how they played with the system. We reveal how this notion of looseness involved striking a balance between on the one hand, expressive and improvised playing in the face of less precise recognition and on the other, more accurate playing against more precise recognition. Adopting a systems perspective, we reveal how looseness can be deeply embedded into recognition technologies through multiple complementary mechanisms and feedback loops so that humans can gradually attune these to their individual needs and practices.

While our notion of looseness is clearly grounded in the practice of music – where looseness and tightness are recognised terms to describe how musicians play together – we argue that it has wider purchase within HCI. In particular, we relate our notion of looseness to other modalities, in particular gesture recognition, and to recent discussion of the H-Metaphor [10] for controlling autonomous systems, noting the distinctive potential of human-writeable notation within such a system which is exemplified by the example of music.

CHI 2017, May 06-11, 2017, Denver, CO, USA
© 2017 ACM. ISBN 978-1-4503-4655-9/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3025453.3025900

RELATED WORK

We begin by positioning our research within two related fields of work: music recognition technologies and interacting with recognition technologies.

Music recognition technologies

Music recognition draws on techniques from the field of *music information retrieval* (MIR) where software tools such as the VAMP plugins [6] and MatLab MIR Toolbox [16] enable the extraction of audio and musical features including pitch, rhythm, timbre, tonality, note onsets, segmentation, chord progressions and loudness.

These underpin diverse *applications*. Automatic music transcription converts audio files to symbolic representations such as sheet music [3]; audio fingerprinting recognises specific audio recordings, for example in Shazam [32] or in query-by-humming [27]. Turning to live performance, automatic score following (audio-to-score) automatically synchronises live audio or MIDI input to a pre-composed score [15] so as to control computer-generated accompaniment, digital effects or trigger extra-musical events such as lighting and visuals [14]. More flexibly, Cypher [26] analyses MIDI data in real-time, extracting key, chord, beat and phrase group features so as to generate musical accompaniment while the Analyser plugin [29] for Digital Audio Workstations extracts real-time audio features and maps them to Open Sound Control (OSC) to control live visuals.

Music recognition technologies may be *embodied* in various ways so as to establish varying relationships with musicians and audiences. They may be seen as extensions of musical instruments. Hyper-instruments [17] for example, augment conventional instruments with digital capabilities. Alternatively, they may be embodied as intelligent or even autonomous players in their own right. The Continuator, for example, learns to improvise stylistically appropriate responses to musical phrases [23] while robot musicians can now improvise alongside humans [5].

Our focus is on supporting performing with 'ordinary' instruments by enabling musicians to compose musical codes that can then be played back during a performance to trigger various interactions. Ideally, these codes can still be recognized even when played back with varying degrees of expression, improvisation and disguise. This notion of musical codes builds on a longstanding tradition of composers playing with musical cryptography (J.S. Bach, Shostakovich and Elgar all hid messages in their music [28]), of mathematical codes providing a compositional framework (e.g., in Serial Composition [24]) and more generally the use of musical themes and Leitmotifs (e.g., by Wagner). More specifically, we build on a proposed format for musical codes that can be recognized by computers [11]. This previous research demonstrated the feasibility of the format and also noted the challenge of looseness. Our contribution here is to extend, apply and reflect upon this approach in greater depth through further iterations of a fully functional prototype.

Interacting with recognition technologies

Widening our perspective, HCI has a longstanding interest in how people interact with all manner of recognition technologies. While we lack space for a comprehensive account, we briefly highlight a few examples that inform subsequent discussions. It can be difficult for humans to understand how to engage invisible sensing systems when first encountering such systems [2]. Compared to direct manipulation, how do we/they know what they are attending to, what they expect, and how to recover from errors? In discussing 'natural' user interfaces, Norman highlights the challenges of needing to tune the system to balance between false positives and false negatives [21]. Others have argued for a systematic analysis of partial overlaps between expected human actions and those that can actually be sensed, arguing that the partial overlaps between these can be a source of both problems and opportunities [4]. While Dix considered temporal aspects of human-human collaboration over networks, his analyses of task and system pace [7] and the need for timely 'local' feedback [8] are equally relevant to human-system collaboration. Finally, various researchers have argued that systems need to provide feedforward to enable users to anticipate their likely actions as well as feedback [9,31].

Parallel to work on music recognition is a strand of research on interactive gesture recognition. The Wekinator [22] applies example-based machine learning algorithms to performance gesture recognition, emphasising the iterative process of training, testing and refinement. Gesture Interaction Designer (GIDE) [33] performs continuous online recognition and tracking of gestures in progress. GIDE allows gestures to be defined by example, and accuracy of matching can be controlled by a user-specified "tolerance" (error distribution). A similar iterative process is fundamental to the work presented here, but the specific case of musical rather than gestural interaction brings complementary challenges and opportunities, in particular the distinctive role of written notations in music.

A further important aspect of interaction that warrants consideration concerns the degree of autonomy of the system. The H-metaphor has been proposed as a concept for reasoning about negotiation between humans and autonomous systems such as autonomous aircraft [10]. The metaphor is to think of the system as being like a horse that is controlled via reins. Sometimes the reins can be loosened to allow the horse greater autonomy while the rider devotes their attention to other tasks, but sometimes tightened so that they can take direct control, for example when negotiating tricky terrain. The H-metaphor has inspired new gestural interfaces for mobile devices that distinguish between 'casual' and 'focused' interactions, where tightly pressing the screen invokes focused (tighter) control whereas gesturing over or towards it invokes casual (looser) control and invites system autonomy [25].

APPROACH

We followed an exploratory and evolutionary system prototyping approach [1]. By this we mean the rapid and iterative engineering of a richly functional and reliable system in collaboration with users in order to learn deeper principles for systems design and interaction. The approach requires the capability to rapidly reengineer a system according to emerging requirements while also abstracting the wider principles that underlie these. This approach shares important features with iterative prototyping, user-centred design [20,13] and agile development [18], but places greater emphasis on research outcomes beyond individual products. Its exploratory character also reflects the three goals of technology probes to inspire reflection on new technologies, understand users' needs and desires, and field test prototypes [12], but places greater emphasis on delivering general purpose tools while also generalizing system design principles beyond field testing. In short, it draws on elements of all of these approaches to embody a deep integration of technology, application and interaction perspectives in and through a robust, evolving system prototype.

Process

Our particular inquiry involved developing two major releases of a musical code recognition system, punctuated by several smaller interim releases. The first version considered here corresponds to "iteration 3" in [11]. The system could be downloaded and installed on any moderately specified desktop or laptop machine, operated through a browser-based interface, and the source code was also made available in a public Github repository[1]. The accompanying video shows a walk-through of the system in operation. The development process was driven by four workshops held over a ten-month period so as to involve musicians in learning and trying out the system and providing feedback. Workshops 1 and 3 were focused on testing major new releases and were hosted in our local Computer Science department. Workshops 2 and 4 focused on pushing the capabilities of each release and were staged in music departments (one local and the other elsewhere).

Workshops lasted between 3-5 hours and broadly followed a common structure which involved: (i) setting up instruments and installing our software onto participant's laptops; (ii) a brief introduction to the system and approach; (iii) a hands-on tutorial working through the system's functionality; (iv) individual explorations with facilitator support; (v) performing and/or reporting back to the wider group; and finally (vi) a round-table discussion. The workshop facilitators captured video documentation including observations of participants' interactions with the system. Periodic interviews were conducted, where participants were invited to explain their processes as they went along. The facilitators interposed questions during recorded feedback discussions to capture specific detail. We also collected the authored "experience" files from 16 of the participants for subsequent analysis.

Participants

We recruited 23 musicians (5 female). In addition 2 members of the research team who were also musicians were participants in the final workshop (P22, P25). In terms of their musical level, 4 were or had been professional, 7 semi-pro and 14 amateur. Their academic backgrounds spanned Computer Science (CS), HCI, music and information science. Collectively, they brought along a diverse collection of instruments including 6 MIDI keyboards, 1 MIDI piano, 1 MIDI drum pad, a keyboard with audio output, a digitally augmented resonator piano [19], 5 electric guitars, 2 acoustic guitars, an electric fretted bass, two electric fretless basses, mandolin, whistle and a laptop running Ableton Live. Three of them (P1, P13 and P16) attended two workshops.

Participants proposed and explored various applications including triggering visuals during live shows (P1); triggering backing tracks (P2, P4, P17, P25); calling up a score (P5, P13. P25); notifying others of tunes being played in a jam session (P10); controlling audio effects (P17, P22); input to generative music (P11); and support for learning (P13, P18, P23); A professional pianist and composer (P16) explored an innovative game-like composition in which the pianist triggers codes to jump to other parts of the score and activate interactive MIDI accompaniments and effects (a full work is currently in development).

FINDINGS: REFINING THE SYSTEM WITH MUSICIANS

We now report the findings from this process, which also illustrate further the nature and operation of the system, and in particular the features that were added to the system to support looseness and tightness in response to musicians' requirements and experiences. For the sake of clarity, we present the system and its features in terms of how they support an overall workflow of composing and performing codes as shown in Figure 1. This involves six key stages:

- **Composing** and refining the codes and pre-conditions.
- **Setting-up** the recognition technology to respond optimally to a specific instrument in the hands of a given player. We adopt this term from the common musical sense of 'setting up' an instrument for a musician. Indeed, our setting-up might potentially involve tweaking the instrument as well as the system.
- **Rigging** the system for a given performance. Again, we draw on common musical parlance that refers to how instruments, PA system, lighting and other parts

[1] https://github.com/cgreenhalgh/musiccodes

of the musical 'rig' are connected to enable an ensemble of musicians to perform in a specific venue.

- **Performing** codes during testing, rehearsal or a show.
- **Capturing** and **Analysing** a performance, including treating captured recordings and logs as if they were live input that can be used to generate new codes or explore other system settings.

We now consider each of these stages in greater detail, with particular emphasis on composing and performing.

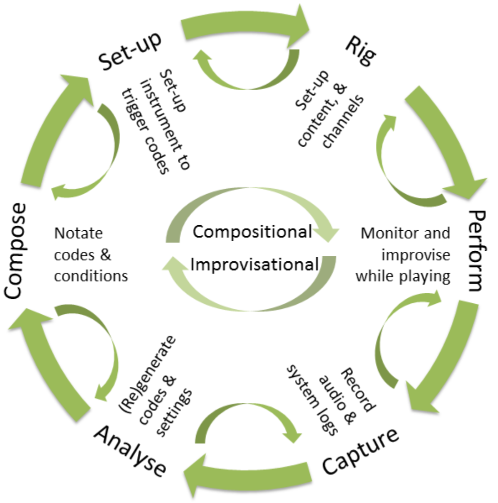

Set-up
Rig
Perform
Capture
Analyse
Compose
Set-up instrument to trigger codes
Set-up content, & channels
Monitor and improvise while playing
Record audio & system logs
(Re)generate codes & settings
Notate codes & conditions
Compositional
Improvisational

Figure 1. The workflow of composing and performing

Composing codes

Musical codes are fundamental to the nature and operation of the system: each code represents a musical phrase – a sequence of notes – that can be recognized or matched by the system in order to trigger a response. The system uses a base textual notation for codes which interleaves *pitch* and *timing* information. We consider these aspects of codes in turn. We then explain several further features of code composition in use.

Pitch

To notate pitch the system uses the International Pitch Notation which denotes middle C (261.63Hz) as "`C4`", i.e. scale note C in octave 4. The simplest musical code is a single note pitch or a short sequence of note pitches, and all of the participants except P19 (a drummer) started with codes of this kind. For example "`C3,Eb3,G3`" (P13) specifies the notes C3, E-flat-3 and G3 in that order. 22 of the 25 participants started by typing in their codes in this format, while the other 3 only entered codes by playing them on an instrument (a facility added in version 2, see below).

13 of the 15 participants in the first two workshops defined their codes to be octave-independent (each pitch could be played in any octave, for example diverse instruments). Five of these also created codes that were octave-specific, i.e. tighter. For example P10 (playing a whistle) changed their codes to specify the octave of the whistle: *"Without the octaves it was picking up talking and all the other stuff [noise] going round, but with the octaves you know what octave this [whistle and tune] is in" (P10).*

Length

Participants typically started by creating short codes comprising 3-5 pitches, as above. 9 participants created at least one code with 7 or more notes, and 4 of these created at least one code with 10 or more notes. These longer codes are more challenging to play and less likely to occur by chance in other music.

Rhythm and Timing

To notate timing and hence rhythm the textual notation can interleave delays. For example "`/1`" denotes a delay of one time unit (i.e., one beat). Only 6 participants incorporated timing into their codes. For example "`C2,/1,D2,/1,C2,/2,C2`" (P19) is a simple drum rhythm (the MIDI snare is C2 and the bass drum is D2). Including timing in the code makes it significantly tighter and more specific than the equivalent pitch-only code.

Version 1 (used in workshops 1 and 2) used the delay between the last two notes played as its time reference, e.g. "`G3/2,B3/2,D3/4`" (P12). However participants found these codes very hard to trigger due to the precision required and the lack of timing cues: *"it's really specific to your timing, you have to be 100%" (P6)* and *"if you had a metronome it would be a lot easier to do that, even if its just a flashing one" (P4)*

The second version of the system incorporated several changes to the handling of timing. By default, timing was made relative to a user-specified reference tempo, with a simple metronome view provided. The granularity used for timing was also made user-definable. At one extreme a granularity or tempo of "`0`" was used by most participants to ignore timing altogether. At the other extreme P16 used very fine granularity ("`100`") but only in conjunction with inexact matching and entering codes by playing (which are described below). P19 (the drummer) explored a range of granularities for timing.

Wildcards and Regular Expressions

Regular expressions are familiar in Computer Science as a way to express patterns to look for within strings. The first version of the system treated code patterns as textual regular expressions, which were matched against the *text representation* of the notes being played. This was completely re-engineered in the second version so that regular expressions and matching were defined in terms of *notes* and *delays* as atoms, with the textual form just a concrete representation. Eight participants made some use of regular expression elements in their codes.

Seven participants created codes like "`C3,.*,D#3`" (P18), i.e. a specific note (C3), followed some time later by another specific note (D#3), but with any number of other notes in between ("`.`" denotes any note, and "`*`" denotes any number of repetitions of the thing it follows). P20 used this to cope with certain errors in note detection that were introducing extraneous notes: *"sometimes the tracking fails by duplicating the note or maybe an octave out… but using the .* notation I was able to deal with those things" (P20)*

Four participants also experimented with other regular expression features. For example, in "`(C|D),.,(C|D)`" (P13) the first and third notes can be C4 or D4, while in "`[D3-F3],.,[D3-F3]`" (P13) the first and third notes can be any notes between D3 and F3. P12 and P15 also created codes with single-note variations, e.g. "`EF[CB]`" (P12, version 1 syntax). All of these made the corresponding code looser, i.e. potentially matched by a bigger range of specific musical phrases.

It was notable that those participants who used regular expression elements were either computer scientists, stated a prior familiarity with regular expressions, or were given support to integrate it into their code composition during the workshop, while those unfamiliar with the concept struggled to incorporate it into their codes. One potential solution for future work is to introduce selected wildcards (based on the ones participants found useful) as bespoke musical annotations.

Whole and Part-phrases

The system segments incoming notes into possible musical phrases based on a user-configurable silence between successive notes. So several notes played in quick succession form part of a single phrase, while a longer gap (by default two seconds) is assumed to mark the start of a new phrase. The composer can specify for each code whether it must occur at the start and/or end of a phrase (tighter), or whether it can appear anywhere within a continuous phrase (looser). All participants created at least some codes that could appear anywhere, while 4 participants (P11, P13, P15, P25) created codes that matched a whole phrase. Playing to trigger these codes becomes quite a distinct activity, including a pause before and after, for example playing a short introductory phrase before launching into a melody.

Logical pre-conditions

By default, the system continuously monitors its input and concurrently checks all of the codes in the current performance to see if they have been matched. It soon became apparent that different codes and actions might be relevant at different points in a performance, and support was added for controlling when codes can trigger via preconditions. To date this has been used by two participants. P16, a professional composer, is working on a composition that incorporates game-like elements, including different "routes" through the composition and musical challenges built into the performance. With support from a facilitator, she introduced a "`matched`" variable to represent whether a particular challenge had been completed yet or not, with a precondition on the challenge of "`!matched`", i.e. not yet done, and an update when the action was triggered of "`matched=true`", i.e. announcing to this and other codes that the first challenge had been met.

P24 was working on a simpler musical scenario but found:*"the one issue I'm facing is that the phrase repeats in the songs.. so right now it's triggering twice" (P24).* With support from the facilitator he then started to explore the use of a "`count`" variable and preconditions/updates to resolve this issue. This system of states and logical pre-conditions provides a powerful and expressive framework in which many different kinds of codes (with different tightness and looseness) can be combined. The current prototype allows more experienced users to selectively reveal and enable this functionality, however this complexity invites more tailored support.

Setting-up an instrument

The challenge of setting up the system to work with a specific instrument involves *connecting the instrument* to the system and adjusting or *'tuning' the system*'s response.

Connecting the Instrument

The system takes two types of input: an audio 'line input' as taken from an electric instrument output or a microphone pre-amp, or a MIDI input. An audio input is processed through the Silvet VAMP plugin [3] which extracts pitch, velocity (i.e. loudness) and timing data for each note onset (doing this for polyphonic music, which is a well-known and challenging problem). In contrast a MIDI input bypasses the feature extraction plugin as the system extracts note (pitch), velocity and timing data directly from MIDI 'note on' messages.

Out of the 25 participants across the 4 workshops, 14 used instruments which used an audio input into the system (6 via a microphone and 8 via a direct line input), 2 (P20, P25) used a third-party audio-to-midi convertor (designed for guitar) and 9 using a MIDI instrument. The majority of these using audio inputs were observed to encounter additional noise and artefacts beyond the note events they were aware of playing. Figure 2 (top) shows an example of a MIDI input stream with time increasing left to right, which shows a 'clean' sequence of note events with no other artefacts appearing. In contrast, figure 2 (bottom) illustrates an audio input from a microphone on an acoustic guitar playing a similar sequence. (These images are part of the system's visual feedback of notes played, which is described further in the section on performing codes.) In particular, some participants playing stringed instruments speculated that the system was 'hearing' a number of additional audio events, such as overtones (harmonics), or un-played strings resonating: *"So its not just picking up the fundamental it's picking up so many more harmonics [P17 plays a note and points to the resultant note cluster appearing on the Muzicode stream]". (P17)*

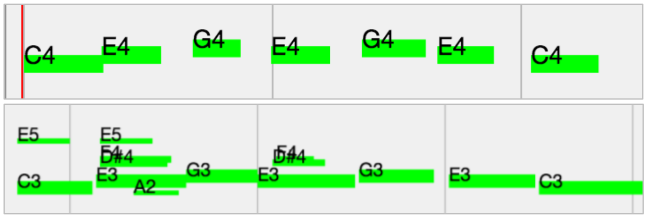


Figure 2: MIDI (top) and Audio (bottom) input examples

These additional 'notes' significantly affected the recognition and triggering of codes. The imperfections of the note recognition process appear to the system as looser playing by the musician. As a result, those participants using audio inputs spent a great deal of their time exploring code recognition, including 'tuning' the system and their own playing.

Tuning the System

In the second workshop, P14, who had extensive experience of note recognition, suggested using the extracted velocity data as a means to filter out unintended or undesirable note events, i.e. to ignore quiet notes (the thickness of the green notes' glyphs in Figure 2 reflects the velocity, e.g. the first E5 overtone is relatively quiet and thin). This suggestion was incorporated into the system and used by four participants. Options were also added to specify a pitch range within which the system would look for codes, which was used by seven participants, including three using MIDI instruments. For example, P17, who was playing an electro-acoustic bass guitar set a minimum velocity value of 25 to remove accidental notes that arise from handling noise in addition to overtones, and tightly constrained the frequency range to a little over an octave corresponding to the lower register of their instrument, to filter out overtones or accidentals that might fall outside of the pitches used in the code. By restricting the attention of the system these types of filters allow the other aspects of the performance to be looser.

Rigging the stage

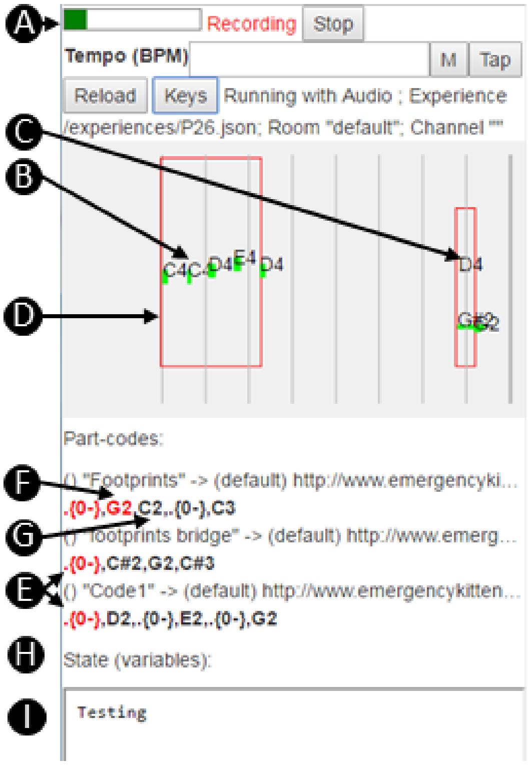
The system forms just one component within a much larger performance environment, which may include other musicians, lighting and visuals, sequencers and effects as well as the performer and their instrument. While not a focus of our workshops to date we note several ways that participants worked towards integrating the system into a broader performance ecology. The system's simplest form of output is to load and display a specified URL. All participants used this during initial prototyping, for example loading videos of musical accompaniments, images of scores and related web-pages. Two participants (P16 and P17) used the MIDI output capabilities that were added in version 2 to control external music applications (PureData and Ableton Live, respectively).

Performing codes

We now consider the actual performance of codes, within the broader performance setting. Two key aspects of the system are prominent here: the *performance monitoring interface* and support for *inexact matching*. We also reflect on *musicians' tactics* to adapt their playing to the system. Performance Monitoring Interface.

Performance Monitoring Interface

Figure 3 shows version 2's main performance interface, in this case for P20. This allows the musician to see: (A) audio input signal level (if using audio); (B) each detected note (green rectangle); (C) each filtered out note (grey rectangle); (D) each musical phrase (red box); (E) all available codes for this performance; (F) which notes of a code have already been matched (red text); (G) which notes are still to play (black text); (H) any current state or preconditions (there are none in this performance); and (I) the current default output channel.



Running with Audio ; Experience /experiences/P26.json; Room "default"; Channel ""

() "Footprints" -> (default) http://www.emergencyki...
.{0-},G2,C2,.{0-},C3
() "footprints bridge" -> (default) http://www.emerg...
.{0-},C#2,G2,C#3
() "Code1" -> (default) http://www.emergencykitten...
.{0-},D2,.{0-},E2,.{0-},G2

```
Testing
```

Figure 3: the performance monitoring interface

Throughout all the workshops we observed participants studying the note visual stream closely when performing codes in the 'performance mode'. In the following example P16 is demonstrating codes they have defined: *P16 starts to play the code while looking intently at the candidate code on the performance view, but notices she is playing it wrong from the (lack of) highlight, "Wait, I can't remember". She looks again at the string of notes in the candidate code and then plays it again, successfully -*

"yeah!". (P16) This behavior was typical across the participant group.

Inexact matching

For version 2 support was also added for *inexact* matching of codes based on a configurable edit-distance metric between the code and the played phrase. So, for example, the default (Levenshtein) distance (or error) between the phrase "`C4,D4,E4`" and the pattern "`C4,E4`" is 1, since it requires at least 1 change: delete (i.e. don't play) D4. The inexact matching algorithm also supports many features of regular expressions, including ranges, alternatives and repeats, all be it only for individual notes and delays.

P25 used inexact matching with relatively long codes and relatively large permitted errors, e.g. a 28 note phrase with up to 8 errors permitted. In this way they were able to cope with errors in note recognition and also introduce embellishments in how they played:*"because when you play tunes you never play them quite the same you embellish them a lot" (P25).* The error parameter thus directly controls one aspect of looseness.

With inexact matching it is also possible to adjust how similar pitches and delays need to be in order to 'match'. P16 used this with quite long pitch-and-rhythm phrases, intended to be challenging technical exercises. These codes used high-resolution timing (to 0.01 seconds) but with an allowed difference between delays of 0.1 second. This allowed P16 to match these codes with care. This is comparable to the tolerance parameter of [33].

Performance tactics

So far, we have focused on system features. However, it was notable that the musicians themselves adopted various tactics to adapt their playing to the system. Adjustment of playing technique to improve code recognition was a common behaviour amongst those using an audio input. For example, as P1 explained: *"So when I was just letting my guitar ring through it was picking up a lot more ghost notes whereas if I damped it I could make sure it would just pick up the note I was fretting and intending to play. You don't even hear those other notes or at least you don't recognise that you hear them." (P1).*

As well as damping or muting strings, participants also found that they could affect the system's performance by the way they plucked the strings, or by switching from using a pick to a fingers: *"I can get it if I finger it more the thumb and finger" (P4).* Some participants also noted that the system sometimes struggled with fast musical sequences: *"It's interesting that a rapid onset of notes it is not coping with. ...it doesn't like that da, da, da, dadada, da" (P9).* P9 achieved more consistent code recognition when the code was performed at a slower tempo.

Capturing and Analysing Performances

Many of the participants in the first two workshops called for alternative methods of code composition other than by textual means, for example traditional notation, or piano-roll input (as is ubiquitous in digital audio workstations), or most commonly, *"to enter a code by playing it" (P4)*

The second version of the system was therefore extended to allow the musician to record "examples" of musical phrases directly into the editor, which shows both the note/phrase view (as in figure 2) and the corresponding textual notation for the phrase, according to the current granularity of timing and pitch. All but one participant who had access to this feature used it to create initial codes at some point. For example, P1 (workshop 3), who had used version 1 in the first workshop, noted how this helped them: *"I mean I play the guitar, but I'm not classically trained ... you don't always know what the notes are. So I could just play something, hear it and take notes from that." (P1).* This functionality was also deemed beneficial by P16, a professional composer and pianist, who explains: *"If you want to write something idiomatic for piano or some other instrument it's good that you try it out yourself first instead of just feeding in some note into the system, that's why I didn't want to type them in, but to play them"(P16).*

In version 2 notes can also be copied from the performance interface back into the editor. For example this was used with P16 during testing: when they tried and failed to trigger a code that phrase was copied as a new variant example. The editor interface shows which examples are being matched by each code, allowing the code and/or filter settings to be adjusted until it matches all of the intended examples, a process reminiscent of Test-Driven Development of software.

Participants also used the recorded examples to explore and refine the instrument and filter settings described above (see *tuning the* system, above). For example P1 recorded multiple examples while they tried variations in their playing technique, and once they had a reasonable note stream: *"...I then went to the frequency [settings] and changed the box [range] and I just kind of played with it to see what was happening ... then I could see that I was just catching the range of notes that were in the phrase..." (P1)* (The visualisation of each example shows how the current filter settings would affect it.)

DISCUSSION: DESIGNING FOR LOOSENESS

Our experience of progressively refining the system in collaboration with musicians reveals how they set about incorporating music recognition technology into their performance practice, especially how they balance the need for musical expressivity with system reliability. The following discussion of exactly how they achieved this foregrounds the concept of "looseness" as a key characteristic of the relationship between human performer and system. We highlight three key aspects of looseness: negotiation of control; feedback and feedforward; and the process of gradual attunement, supported by notation.

	"Tight"		**"Loose"**
Human musician	Precise, slow playing. Muting, no sustain. Fixed score. Strict tempo. Same player & instrument. Repeated phrases.	→ *More expression* ← *Less expression*	Embellishments, Improvisation, grace notes. Feel, swing, rubato. Different player or instrument.
	↑ *Strict* ↓		↑ *Improvised* ↓
System	Long code. Whole phrase. Exact matching. Rhythm & pitch. Fine quantisation. No pre-filter. More preconditions.	→ *Fewer false –ves* ← *Fewer false +ves*	Short code. Part-phrase. Inexact matching, wildcards. Rhythm *or* pitch. Course quantisation. Tight pre-filter Few preconditions.

Figure 4. Tightness-looseness relationships and trade-offs

When used in relation to music, looseness is a rich and ambiguous concept, which can encompass elements of personal style, deliberate and accidental variations of tempo and rhythm (e.g. "swing" or "feel"), elements of improvisation, embellishment and variations in synchronization among musicians. In contrast, "tightness" implies close coordination, precision and synchronization.

We suggest that these terms can also be applied to the relationship between a performer (such as a musician) and a performance technology. Tightness strives to ensure a reliable system response to accurate performance, encouraging the precise rendition of codes with reduced recognition errors. In contrast, a degree of looseness introduces flexibility into the relationship between performer and system, enabling the performer to vary how codes are performed and enabling the system to interpret them with a degree of latitude. While our focus has been on music, the situation is similar with gestural control.

Negotiating control of looseness

Many of the technical innovations described above are responses to the need to express tighter or looser codes, and different specific mechanisms are needed and used depending on the musical context and the skills and experience of the user. Figure 4 shows how tightness and looseness can be *negotiated* through both the musician's playing and the configuration of the system. As we have seen, the system may be made tighter with the musician (bottom-left) by using longer codes, by matching on rhythm as well as pitch or by matching only entire phrases. But each choice of mechanism also imposes constraints on how such codes can be used and heard within the performance as a whole. For example, a longer code is necessarily a longer musical phrase, and a code that matches a whole phrase will not be triggered by the same musical motif within another phrase; inevitably codes are not "just" codes, but are an intrinsic part of the whole musical performance. Alternatively, the system can be made looser with the musician (bottom-right) by refining a simple code to use wildcards, inexact matching or pre-filtering. Each of these mechanisms also has distinct musical implications, for example inviting controlled repetition or variation in the case of wildcards, or the creation of distinct "trigger zones" within the expressive range of the instrument in the case of pre-filtering.

The addition of logical pre-conditions provides a powerful framework to control which codes can be triggered at which stages of the performance, for example providing a way for the musician to temporarily "conceal" loose codes when they should not be triggered. Note that almost all of these mechanisms have analogues in other modalities. For example, gestures can vary in length, the use of timing, whether they are embedded within longer gestural "phrases", how inexactly they are matched and whether and how sensor inputs are filtered. As with music, each specific mechanism will have distinctive performance implications.

At the human level, we reported earlier how the musician may adapt their playing to get tighter with the system (top-left) by playing slowly, simply and precisely, following a fixed score in tempo, using a muted style to avoid confusing overtones and sticking to known and recognisable instruments. Or they may choose to get looser with the system (top-right) by varying, relaxing, improvising and embellishing their playing.

Figure 4 also shows key relationships between the tightness/looseness of the musician and that of the system. There are two vertical relationships in which these come into positive alignment. At the tight end of the spectrum is a 'strict' musical relationship in which either the musician is able to play strictly as required by the system (e.g., to a composer's score and timing) or where the system is finely tuned to respond to the nuances of an individual's playing. At the loose end is an 'improvised' relationship where either the system is able to accommodate variations in playing and/or the musician is happy to respond flexibly.

Diagonal misalignments may be especially problematic. Workshop participants' initial exact (i.e. relatively tight) codes and their initial (relatively loose) playing led to many false negatives from the system, i.e. missed codes, especially when using audio inputs. Faced with such a situation, players might make their playing tighter, moving themselves towards the top-left of the figure, or might make the system looser, moving it towards the bottom-right of the figure. However, each choice has likely consequences, and each mechanism has distinct characteristics and challenges. A loose system will work to some extent with loose playing but will generate more false positives and require the musician to improvise to accommodate these. On the other hand, tighter playing will make the system more reliable but at the cost of a lack musical expression as the musician conforms to the specific expectations embodied in the codes.

Feedback and Feedforward

Table 2 summarises how *feedback* is provided from *all* stages of the music recognition pipeline: connecting a working instrument before ensuring the system is receiving a signal; detecting notes correctly; ignoring overtones; recognizing gaps between phrases; matching notes to available codes; and finally triggering the desired actions. The table also shows how the monitoring interface provides *feedforward* [9,31] to inform the musician what codes are available and what they need to do next to complete a chosen code, specifically which codes are close to triggering and which notes still need to be played to complete them.

These various forms of feedback and feedforward can be related to Bellotti et al's questions for the design of system systems [2] reviewed earlier. Musicians know that the system is attending through the VU meter and visible displays of recognised notes. They can see how it understands their command through the display of partially recognised codes. Feedforward helps them anticipate its likely actions and adjust their playing, for example to avoid mistakes such as triggering the wrong code.

Considering collaboration over networks (i.e. with delays) Dix [7] highlights the need to consider the 'pace' of interaction. This is how long a 'turn' or a complete action cycle takes (including feedback). In musical interaction some actions are short and fast paced, for example striking a string and hearing a sound; others are much longer and slower paced, for example playing a phrase or an entire piece. If the pace of the system is slow, for example due to communication delays [8] or in our case note and phrase recognition delays, then it is important to provide timely 'local' feedback that an action is in progress, in addition to the final (delayed) feedback that the action has completed successfully or otherwise. Applying Dix's insights, the system provides multiple feedback loops at different timescales in order to provide timely feedback on *all* stages of the phrase recognition process. [33] provides a similar example of fine-grained interactive feedback applied to gesture recognition rather than music.

Dataflow	**Feedback mechanism**	**[Delay] Utility**
Audio signal[1]	VU meter[1]	[<100ms[1]] Is my instrument working? [1]
(MIDI) notes	Note glyph (pitch/vol.)	[Up to 1s[1] / <50ms[2]] Is my instrument working? Is it detecting notes? [1] Which?
Filter	Note highlight	Is it ignoring overtones, etc.?
Phrase	Phrase box	[+2s[3]] Is it the same or a new phrase?
Code matching	Code order & part-highlight	Is it being matched by a code? Which code(s)?
Codes	Rest of code & other codes	What note(s) can I play next? What other code(s) are there?
Action	Final output	[Length of trigger phrase] Did the code work?

[1]Audio input. [2]MIDI input. [3]Between phrases or at end of a phrase (configurable).

Table 2: Summary of feedback/feedforward mechanisms

While the monitoring interface is useful for providing a rich array of feedback and feedforward, we note the key challenge of providing similar facilities without requiring the musician to stare at this screen, for example through a dynamic digital score, fine-grained audio feedback such as dynamic effects reflecting partial match, and feedback through other channels such as lighting or custom displays.

Attunement and Notation

Our experience suggests that it is also necessary to step-back and adopt a longer-term view of how looseness is negotiated over time. Specifically, we identify a complex and iterative process by which one or more musicians (composers and/or performers), specific musical instruments and the system gradually become aligned so that codes can be played reliability and with expression.

Inspired by Turner et al [30], we consider this to be a process of *attunement*, by which we mean the gradual, iterative and detailed refinement and configuration of an interactive system for use by a highly skilled practitioner. Their grounded theory studies of digital arts projects captured how computer programmers attune technology so that it is malleable to artists, revealing how this required intensive, iterative and fine-grained 'intimate iterations' to understand the 'language' and 'character' of the system and so produce both technical and aesthetic meaning.

Referring back to the workflow shown in Figure 1, we suggest that in our case, attunement is a highly iterative process at multiple levels of scale. At the macro level, musicians may pass around many iterations of the overall cycle as they gradually attune the technology to their needs and themselves to the technology. This is of course, standard practice for musicians who repeatedly practice, rehearse and perform in order to develop their craft. Indeed, becoming attuned to an instrument can be a lifelong process. At the micro level, each stage may involve iterations of exploration, testing and tweaking.

We observed two broad attunement strategies that begin on opposite sides of the cycle. The *compositional strategy* starts with composing codes that are then mapped onto specific instruments, players and settings so that they can be performed, with capture and analysis then enabling further refinement. In contrast, the *improvisational strategy* begins with playing music (typically providing examples) that is then captured and analysed so that codes, settings and configurations can be derived. The latter approach dominates in gesture-based systems such as [22,33], where there is no equivalent of a musical notation. This raises the challenge of supporting complementary compositional strategies in gesture and other modalities, perhaps based on existing notations used for dance, such as Labanotation.

While attunement may be entirely carried out by one musician who composes and performs their own codes, it might also be split among roles. Composers may be separate from performers, and there may be other specialists such as engineers and roadies all of whom might be involved at different stages. A common notation has a useful role within such collaborative processes, as we see from the everyday practices of music publishing.

Tools vs intelligent systems and the H-metaphor

By focusing closely on one specific music performance technology, we have been able to reveal the complex notion of looseness that arises when humans perform with digital technologies. We conclude our discussion by widening our perspective to consider the wider relevance of this to HCI in general. As noted earlier, the H-metaphor likens the idea of negotiating tightness and looseness between human and system to using reins to control a horse [10] which in turn has inspired the notions of 'casual' and 'focused' interactions with mobile devices [25].

Our framework for negotiating tightness and looseness for musical codes can be viewed as an example of applying the H-Metaphor to the domain of interactive performance, showing one way in which the metaphor can be deeply embedded into the design of an interactive system. We suggest that our experience offers three lessons for others who may wish to implement the H-Metaphor in other contexts:

- The negotiation of tightness and looseness can be subtle and context-dependent and requires multiple complementary mechanisms at different scales, not just a single mechanism or strategy (such as "tolerance").
- Interactive performance systems require detailed and timely feedback and feedforward about past and future system perception and behaviour to support productive interaction, especially during the attunement process. Typically this will require multiple feedback loops operating at different timescales.
- The use of a human-readable and writeable notation for the internal states can provide many benefits, including support for aspects of looseness (e.g. regular expressions), compositional as well as example-based strategies, feedforward and collaborative use.

An important difference between our work here and previous uses of the H-metaphor concerns the apparent intelligence and/or autonomy of the system. Previous uses have focused on systems that appear to be intelligent or independent. In contrast, our work positions the system as being a tool, albeit a complex one that can be difficult to control, much like a traditional musical instrument. For example our current prototype does not adapt *itself* dynamically, however the pre-condition system allows alternative codes with varying looseness to be configured within a single performance. However, we note a wealth of research into intelligent and autonomous music systems that engage with musicians on a more human level, including improvising with them, for example an emerging generation of robotic musicians that can play in ensembles with humans [5]. Future work might explore whether these kinds of interactive music technology will also require complex processes of attunement – perhaps more mutual ones as we might find between human musicians – with their own interpretations of tightness and looseness.

CONCLUSIONS

Through a process of exploratory and evolutionary system prototyping, we have revealed how a music recognition system can be extended to enable a range of musicians who are not technical experts to attune it to their needs. This involves negotiating an appropriate degree of looseness between their playing and the system's capabilities through diverse complementary mechanisms, both to deal with recognition errors and to support musical expression. The same mechanisms and processes can be applied in other modalities such as gesture, although we note that the use of a written notation – common in music – raises additional challenges in other modalities.

ACKNOWLEDGEMENTS

This work was supported by the UK Engineering and Physical Sciences Research Council [grant numbers EP/L019981/1, EP/M000877/1]. Data underlying this publication: http://dx.doi.org/10.17639/nott.70

REFERENCES

1. Michel Beaudouin-Lafon and Wendy Mackay. 2002. Prototyping tools and techniques. In *The human-computer interaction handbook*, Julie A. Jacko and Andrew Sears (Eds.). L. Erlbaum Associates Inc., Hillsdale, NJ, USA 1006-1031.
2. Victoria Bellotti, Maribeth Back, W. Keith Edwards, Rebecca E. Grinter, Austin Henderson, and Cristina Lopes. 2002. Making sense of sensing systems: five questions for designers and researchers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '02). ACM, New York, NY, USA, 415-422. DOI=http://dx.doi.org/10.1145/503376.503450
3. Emmanouil Benetos and Simon Dixon. 2012. A Shift-Invariant Latent Variable Model for Automatic Music Transcription. *Comput. Music J.* 36, 4 (Dec.2012), 81–94.
4. Steve Benford, Holger Schnädelbach, Boriana Koleva, Rob Anastasi, Chris Greenhalgh, Tom Rodden, Jonathan Green, Ahmed Ghali, Tony Pridmore, Bill Gaver, Andy Boucher, Brendan Walker, Sarah Pennington, Albrecht Schmidt, Hans Gellersen, and Anthony Steed. 2005. Expected, sensed, and desired: A framework for designing sensing-based interaction. *ACM Trans. Comput.-Hum. Interact*. 12, 1 (March 2005), 3-30. DOI=http://dx.doi.org/10.1145/1057237.1057239
5. Mason Bretan and Gil Weinberg. 2016. A survey of robotic musicianship. *Commun. ACM* 59, 5 (April 2016), 100-109.
6. Chris Cannam, Matthias Mauch, Matthew E. P. Davies, Simon Dixon, Christian Landone, Katy C. Noland, Mark Levy, Massimiliano Zanoni, Dan Stowell, and Luís A. Figueira, 2013. MIREX 2013 entry: Vamp plugins from the centre for digital music. MIREX.
7. Alan J. Dix. 1992. Pace and interaction. In *Proceedings of HCI'92: People and Computers VII*, Eds. A. Monk, D. Diaper and M. Harrison. Cambridge University Press. 193-207.
8. Alan J. Dix. 1995. Cooperation without (reliable) Communication: Interfaces for Mobile Applications. *Distributed Systems Engineering*, 2(3): pp. 171-181.
9. Tom Djajadiningrat, Kees Overbeeke, and Stephan Wensveen. 2002. But how, Donald, tell us how?: on the creation of meaning in interaction design through feedforward and inherent feedback. In *Proceedings of the 4th conference on Designing interactive systems*: processes, practices, methods, and techniques (DIS '02). ACM, New York, NY, USA, 285-291. DOI=http://dx.doi.org/10.1145/778712.778752
10. Kenneth H. Goodrich, Paul C. Schutte, Frank O. Flemisch and Ralph A. Williams. 2006. Application of the H-Mode, A Design and Interaction Concept for Highly Automated Vehicles, to Aircraft. 2006 ieee/aiaa 25TH Digital Avionics Systems Conference, Portland, OR, 1-13. doi: 10.1109/DASC.2006.313781
11. Chris Greenhalgh, Steve Benford, Adrian Hazzard. 2016. ^muzicode$: Composing and Performing Musical Codes. In *Proceedings of AM '16*, October 04 – 06. Norrköping, Sweden. http://dx.doi.org/10.1145/2986416.2986444
12. Hilary Hutchinson, Wendy Mackay, Bo Westerlund, Benjamin B. Bederson, Allison Druin, Catherine Plaisant, Michel Beaudouin-Lafon, Stéphane Conversy, Helen Evans, Heiko Hansen, Nicolas Roussel, and Björn Eiderbäck. 2003. Technology probes: inspiring design for and with families. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '03). ACM, New York, NY, USA, 17-24. DOI=http://dx.doi.org/10.1145/642611.642616
13. ISO/IEC. 1999. 13407 Human-Centred Design Processes for Interactive Systems, *ISO/IEC 13407: 1999* (E).
14. C. Joder, S. Essid, and G. Richard. 2011. A Conditional Random Field Framework for Robust and Scalable Audio-to-Score Matching. *Trans. Audio, Speech and Lang. Proc*. 19, 8 (November 2011), 2385-2397. DOI=http://dx.doi.org/10.1109/TASL.2011.2134092
15. Anna Jordanous and Alan Smaill. 2009. Investigating the Role of Score Following in Automatic Musical Accompaniment. *Journal of New Music Research*, 38 (2). 197 - 209.
16. Olivier Lartillot and Petri Toiviainen. 2007. A Matlab toolbox for musical feature extraction from audio, In *International Conference on Digital Audio Effects*, 237–244.
17. Tod Machover and Joe Chung. 1989. Hyperinstruments: Musically Intelligent and Interactive Performance and Creativity Systems. In *ICMC 1989*, 186-190.
18. Robert Cecil Martin. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA.
19. Andrew McPherson. 2010. The Magnetic Resonator Piano: Electronic Augmentation of an Acoustic Grand Piano. *Journal of New Music Research*, 39(3), 189-202. DOI=10.1080/09298211003695587
20. Donald A. Norman and Stephen W. Draper. 1986. *User Centered System Design; New*

Perspectives on Human-Computer Interaction. L. Erlbaum Assoc. Inc., Hillsdale, NJ, USA.

21. Donald A. Norman. 2010. Natural user interfaces are not natural. *interactions* 17, 3 (May 2010), 6-10. DOI=http://dx.doi.org/10.1145/1744161.1744163

22. Rebecca Fiebrink, Perry R. Cook, and Dan Trueman. 2011. Human model evaluation in interactive supervised learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11). ACM, New York, NY, USA, 147-156. DOI=http://dx.doi.org/10.1145/1978942.1978965.

23. François Pachet. 2004. On the design of a musical flow machine. In *A Learning Zone of One's Own,* Tokoro and Steels (Eds), IOS Press, 111-134.

24. George Perle. 1972. *Serial composition and atonality: an introduction to the music of Schoenberg, Berg, and Webern*. Univ of California Press.

25. Henning Pohl and Roderick Murray-Smith. 2013. Focused and casual interactions: allowing users to vary their level of engagement. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2223-2232. DOI: http://dx.doi.org/10.1145/2470654.2481307

26. R. Rowe. 1992. Machine Listening and Composing with Cypher. *Comput. Music J.* 16, 1, 43.

27. Justin Salamon, Emilia Gomez, Daniel P. W. Ellis, and Gael Richard. 2014. Melody extraction from polyphonic music signals: Approaches, applications, and challenges. *Signal Process. Mag*. IEEE 31, 2–134.

28. E. Sams. 1970. Elgar's Cipher Letter to Dorabella. *Music. Times*, 151–154.

29. A. M. Stark. 2014. Sound Analyser: A Plug-In For Real-Time Audio Analysis In Live Performances And Installations. In *Proceedings of New Interfaces for Musical Expression* (NIME), London, 2014

30. Greg Turner, Alastair Weakley, Yun Zhang, Ernest Edmonds. 2005. Attuning: A Social and Technical Study of Artist–Programmer Collaborations. In 17th *Workshop of the Psychology of Programming Interest Group*, Sussex University, P. Romero, J. Good, E. Acosta Chaparro & S. Bryant (Eds). Proc. PPIG 17, 106-109**,** June 2005

31. Jo Vermeulen, Kris Luyten, Elise van den Hoven, and Karin Coninx. 2013. Crossing the bridge over Norman's Gulf of Execution: revealing feedforward's true identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13). ACM, New York, NY, USA, 1931-1940. DOI: http://dx.doi.org/10.1145/2470654.2466255

32. Avery Wang. 2006. The Shazam music recognition service. *Commun. ACM* 49, 8 (August 2006), 44-48. DOI=http://dx.doi.org/10.1145/1145287.1145312

33. Bruno Zamborlin, Frederic Bevilacqua, Marco Gillies, and Mark D'inverno. 2014. Fluid gesture interaction design: Applications of continuous recognition for the design of modern gestural interfaces. *ACM Trans. Interact. Intell. Syst.* 3, 4, Article 22 (January 2014), 30 pages. DOI=http://dx.doi.org/10.1145/2543921.