A platform for low-latency continuous keyboard sensing and sound generation

Giulio Moro
Queen Mary University of London
E1 4NS, Mile End Road
London, England
g.moro@qmul.ac.uk

Andrew P. McPherson
Queen Mary University of London
E1 4NS, Mile End Road
London, England
a.mcpherson@qmul.ac.uk

ABSTRACT

On several acoustic and electromechanical keyboard instruments, the produced sound is not always strictly dependent exclusively on a discrete key velocity parameter, and minute gesture details can affect the final sonic result. By contrast, subtle variations in articulation have a relatively limited effect on the sound generation when the keyboard controller uses the MIDI standard, used in the vast majority of digital keyboards. In this paper we present an embedded platform that can generate sound in response to a controller capable of sensing the continuous position of keys on a keyboard. This platform enables the creation of keyboard-based DMIs which allow for a richer set of interaction gestures than would be possible through a MIDI keyboard, which we demonstrate through two example instruments. First, in a Hammond organ emulator, the sensing device allows to recreate the nuances of the interaction with the original instrument in a way a velocity-based MIDI controller could not. Second, a nonlinear waveguide flute synthesizer is shown as an example of the expressive capabilities that a continuous-keyboard controller opens up in the creation of new keyboard-based DMIs.

Author Keywords

Augmented keyboard, continuous control, Hammond organ

CCS Concepts

•Computer systems organization → Embedded software; •Applied computing → Sound and music computing;

1. INTRODUCTION

Several keyboard instruments offer a more or less subtle position and/or gesture dependent control on the timbral and temporal characteristics of the sound of a note, as reviewed in [10, 15]. The Ondioline, an electronic synthesizer invented in 1941 by Georges Jenny, is a particularly outstanding demonstration of how the effect of continuous key position, combined with side-by-side vibrato can produce a remarkably expressive instrument, even by today's standards [2]. Regardless, for many years it has widely been accepted that the scalar parameter of onset velocity is enough to characterise the qualities of a note for the purposes of synthesising or analysing a performance [17, 14].

The complex gestural language of a DMI performer is reduced in dimensionality and bandwidth to fit through the affordances provided by the interface, projected down through a *bottleneck* and then expanded out again into the parameters that control the sound generation [5]. Any data not actively selected for "digitisation" will not reach the sound generator and will not affect the resulting sound, and consequently get lost in the process. When a keyboard DMI is designed to let through its bottleneck only the information relative to the note pitch and velocity, all those more or less subtle forms of control available on acoustic and electromechanical keyboard instruments which are not velocity-based disappear.

McPherson in [9] introduced a portable device to sense the continuous position of the key on the piano, designed for the purpose of performance analysis and augmentation. We use their keyboard scanner as a controller for DMIs, applying it to keyboards other than the piano. Its sensing capabilities allow us to widen the bottleneck of digital keyboards and to design instruments that respond to different touch gestures and key position in a way impossible with velocity-based controllers. The raw positional data can be used directly to control parameters of a sound generator, or processed through a gesture recognition algorithm. In our platform we connect the scanner to a Bela[1] computer for real-time sound synthesis, solving a number of associated engineering challenges, ultimately achieving action-to-sound latency shorter than 5 ms. In the remainder of this paper we describe our platform, as well as two instruments we designed with it.

2. THE PLATFORM

We aim to create a responsive platform, with an action to sound latency that is low enough not to be detrimental for performance. David Wessel recommended 10 ms with a jitter of ±1 ms as the recommended worst case latency for a responsive instrument [18]. Jack et al. confirm those findings, in particular showing that larger jitter values can be detrimental to the perceived quality of the instrument [4]. In order to meet these requirements, we started with a Bela board, which was designed to achieve action to sound latency figures below 1 ms when reacting to audio or sensor inputs [12, 11], and we connect McPherson's keyboard scanner to it.

The scanner's native interface to the host is via its USB device port, through which it streams Sysex MIDI data to the host. However, there is no way of accessing the USB bus in a real-time safe way from the Bela audio environment, as USB communication goes through the generic Linux driver, which cannot guarantee low-latency and low-jitter performance, especially under heavy CPU load. We therefore engineered a solution where one of the PRUs[2], acts as the

[1] `http://bela.io`
[2] PRU(Programmable-Realtime-Unit), is a microcontroller on-board the Texas Instruments AM3358 System on Chip.

NIME'20, July 21-25, 2020, Royal Birmingham Conservatoire, Birmingham City University, Birmingham, United Kingdom.

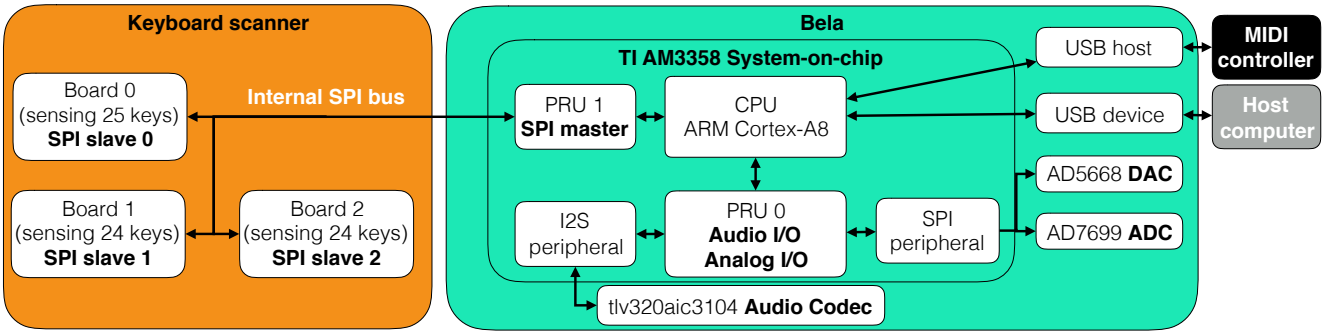

Figure 1: Block diagram of the system comprising the keyboard scanner and the Bela board

SPI[3] master on the internal communication bus used by the keyboard scanner boards. A block diagram of the system comprising the keyboard scanner and the Bela board and all the relevant peripherals and communication buses is detailed in Fig. 1.

The Scanner I/O thread, running on PRU1, handles the communication with the scanner boards. It gathers data from each board every 1ms (sampling rate is 1kHz), and it sends a synchronization message every 5ms, to ensure that the devices remain time-aligned. As soon as a frame of data is received, the PRU sends an interrupt request (IRQ) to a companion Scanner Processing real-time thread running on the ARM Cortex A8 single-core CPU. The thread wakes up and retrieves the data from the PRU memory, applies the linearisation algorithm and makes the linearised data available to the user code. This thread can also perform signal processing on the key scanner data, in order to compute key velocity, or extract touch features, from the raw data. On the other hand, when the instantaneous position of the key is used to directly control a parameter in the audio generator, then the audio thread can access the positional information directly through a shared buffer, and apply the necessary filtering to smooth the data, if needed.

When running alongside the Bela environment, the Bela audio thread will also run on the ARM CPU, while the Bela PRU code will run on PRU0. The Bela audio thread and the Scanner Processing thread will therefore be contending for CPU time. Given how we regard a glitch in the audio as more detrimental than a delay in retrieving a scanner reading, or even the loss of one full frame of scanner data, we give a higher priority to the Audio Thread than the Scanner Processing thread, so that the former can preempt the latter as needed. Some lower-priority threads run alongside the Audio and Scanner threads on the ARM CPU. These provide facilities such as MIDI I/O, for an external USB MIDI controller, and disk output, to write logging audio and key position data to disk for later analysis.

2.1 Action to sound latency

We performed a series of measurements to evaluate the action-to-sound latency of the platform, by measuring the delay between flashing a IR source in front of the sensor and the resulting sound output. The overall latency figure will include the scanning latency of the sensor boards, the audio processing block size on Bela and the built-in latency of Bela's audio DAC. The summary of our results are shown in Table 1. The mean latency is 3.33 ms for an audio block size of 16, and the jitter, that is the maximum variation between the mean and the worst case scenario, is below ±0.5 ms. At 64 samples per block, the mean latency is 4.15 ms, and the jitter is ±0.7 ms, so that the worst case scenario is always below 5 ms. CPU usage does not seem to have a considerable effect on the latency figures, however we would expect that when the CPU usage of the audio thread approaches 90%, the performance may degrade severely, as the scanner thread will start missing some of its deadlines. Our results confirm that the performance of our platform well exceeds the criterias set by [4] and can consistently achieve better results than most devices commonly used for DMIs [11].

[3]Serial Peripheral Interface, a 4-wire high-speed communication bus.

Block size	CPU usage	Latency(ms)			
		Mean	σ^2	Min jitter	Max Jitter
16	25%	3.33	0.19	-0.37	0.44
16	60%	3.33	0.19	-0.37	0.35
64	25%	4.15	0.28	-0.65	0.62
64	60%	4.17	0.28	-0.67	0.69

Table 1: Action to sound latency figures for our platform. We performed 1600 measurements for each condition.

2.2 Sensor linearisation

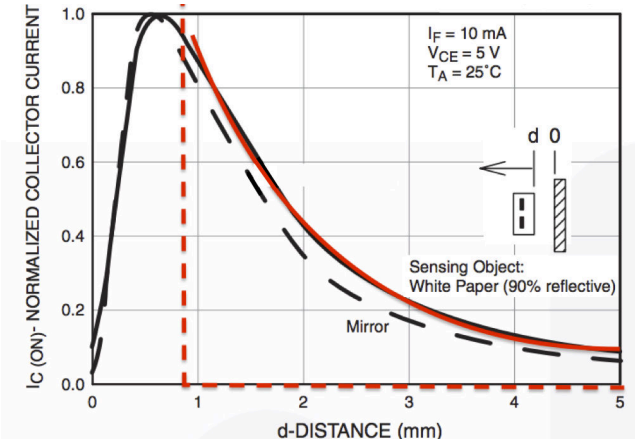

Figure 2: QRE1113: Normalized collector current vs Distance between device and reflector. This image is our overlay (red) on top of an image from the QRE1113 datasheet. The solid red line is the curve described by Eq. (1).

The keyboard scanner uses near-field optical reflectance sensors (Fairchild QRE1113). The collector current of the sensing phototransistor, which is sensed by the board's ADC after being converted to voltage, is a non-monotonic function of the distance of the reflecting surface (the key), as shown in Fig. 2.

In the monotonic region, between 0.6 mm and 5 mm, the relation between the sensor reading s and the key displacement x can be described by the following formula:

$$s = \frac{a}{(x+b)^2} + c \qquad (1)$$

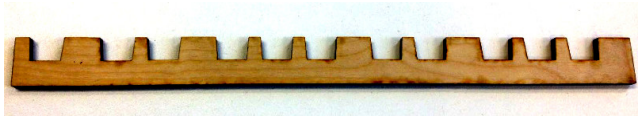

Figure 3: The "calibration comb"

Our semi-automated calibration procedure measures the sensor output at three known points: key top, key bottom, and an intermediate point, the latter of which is obtained using the calibration tool in Fig. 3. The three readings are used to infer parameters a, b and c through least-squares numerical curve fitting, so that x can be computed at runtime.

2.3 Percussiveness detection

A percussive key press occurs when the finger is already moving downwards before hitting the key, and we are interested in obtaining a discrete metric for each key onset, to quantify the amount of percussiveness. [1] obtain a percussive metric from the ratio of the key depression at half the attack duration to the maximum key depression and the average of the key depression curve. Besides introducing a latency in the detection, by postponing the computation of the metric until the key has reached the key bottom, this approach would not work in the presence of incomplete key presses. The approach in [9] considers the ballistic collision that causes the key to bounce off the finger shortly after the initial finger-key impact.

Fig. 4 shows the key and velocity profiles of a percussive key press played on a Yamaha CP-300 digital piano[4], sensed through the piano scanner. As the key is hit by the finger, kinetic energy is transferred from the finger to the key, and the key starts a fast downward motion while it temporarily loses contact with the finger, which is still moving downwards but more slowly. The key is moving freely downwards, and the kinetic energy progressively dissipates until the key stops and eventually starts moving upwards. Shortly after that moment, the finger, which has kept moving down all along, catches up and the key starts moving downwards again, this time under the direct pressure of the finger. This behaviour translates in the velocity profile exhibiting an initial spike due to the impact, and the key profile exhibiting a local maximum during the early part of the onset, corresponding to the point where the key starts the upwards motion.

The percussion detection algorithm starts by detecting a local maximum in the key position during the early part of the key onset, when a maximum is found, the program looks back at the recent history of the key position to find the maximum value of the velocity, and that value is then used as the percussiveness metric.

[4] https://uk.yamaha.com/en/products/music_production/stagekeyboards/cp300

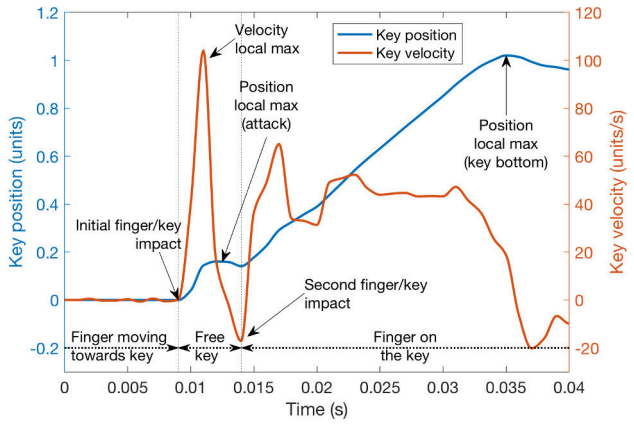

Figure 4: The position and velocity profile of a percussive key press.

2.4 Aftertouch

Aftertouch is the term used to indicate the extra pressure put into the key once it reaches key bottom. In [9], the aftertouch threshold is set dynamically for each key press when a local maximum is encountered in the temporal evolution of the key position. When playing pressing the key fully, this local maximum would correspond to the key bouncing back ever so slightly after hitting the keybed (see Fig. 4). However, when playing the keyboard in a continuous fashion, the player would often perform partial key presses, where they may move the key up and down in the key throw, thus generating several local maxima, without ever reaching the key bottom. Our solution is to record the key bottom position during calibration, and to also record the maximum amount of key displacement in response to an aftertouch gesture for each key. These two values are used to normalise the aftertouch data across the keys.

2.5 Monophonic key detection

Keyboard instruments with limited polyphony need to adopt a strategy to decide what keys emit sound when more keys than the polyphony limit are pressed at the same time. In the case of monophonic synthesizers, the most common strategies are lowest-key, highest-key or most-recent key priority. However, these priority schemes only make sense in the context of discrete key states, where a key can only be "pressed" or "not pressed". However, in the case of an instrument where the key position continuously shapes the sound, like ours, a more complicated model is needed in order for the interaction to be intuitive. We created a priority algorithm that can be defined as "most recent and deepest priority" to be used with our platform when in monophonic mode, which aims at being intuitive for the player, so that the latest key that has seen considerable action is the active one, unless it is being released, in which case another key can take priority more easily. This is achieved through dynamic movement thresholds informed by the state of the key, according to [9]'s state machine.

3. HAMMOND ORGAN EMULATOR

The Hammond organ, patented by Laurens Hammond in 1934, is an electromechanical keyboard music instrument, a polyphonic additive synthesizer, whose oscillator bank consists of 91 quasi-sinusoidal signals. Each contact is connected to one of the sinusoids from the tone generator, and each of these corresponds to the frequency of one of the harmonics or sub-harmonics of the note. Every time a key is pressed, this causes each contact to close against one metal bar ("busbars"), connecting each oscillator, via resistive wires that allow for passive scaling and summation, to the output busbar. The characteristic onset transient of notes on the Hammond is due to a combination of the non-synchronous triggering point for the nine contacts, and their bouncing pattern, which is dependent on the continuous key position and key velocity during the key press [16]. Because of manufacturing tolerances, each contact closes at a different point in the key-throw, so that the activation of each contact is dependent on the instantaneous position of the key. Additionally, the velocity measured around the point where key contacts close affects the duration of the of each contact's bouncing pattern. The Hammond is therefore touch-responsive, in such a way that cannot be captured with traditional velocity-based keyboard sensing. Most digital emulations of the Hammond organ do not allow the player to control the sound generator with the

continuous position of the key, as they tend to use regular MIDI keyboards as controllers, with the notable exceptions of Hammond-Suzuki's New B-3[5] (9 switches), and XK-5[6] (3 switches).

We created a Hammond emulator for the purpose of a performance study (not presented here) on the effect of mappings between key position and virtual contacts in the tone generator, thus maintaining the playing interface (keyboard and expression pedal) as close to the original instrument as possible was of primary importance. We used the keyboard of an actual Hammond C-3 console as a controller, by installing the keyboard scanner on its lower manual and replacing the organ's sound generator with audio generated on Bela.

Figure 5: The keyboard scanner installed on the lower manual of a Hammond C-3

3.1 From discrete to continuous

setBfree[7], an open-source Hammond emulator which served as the sound generator, is a fully-featured Hammond emulator that, in its original implementation, receives MIDI note messages to activate each key, and implements static or random envelopes to emulate contact bouncing. In order to take advantage of the continuous keyboard controller, we implemented the following additional features:

- position-dependent activation of the individual contacts, adding hysteresis to avoid repeated triggering as a consequence of noise in the key position signal
- a dynamic bounce generator where the initial speed of the contact can be set for every contact for every onset
- instantaneous velocity detection at the moment of contact activation, in order to drive the bounce generator

3.2 Dynamic contact bounce model

We used an empirical approach to simulate the bouncing of contacts during the note onset, producing a physically-justified model that can dynamically respond to the initial velocity of the contact. We describe the motion of a contact bouncing on the busbar as the piece-wise concatenation of harmonic motions, where each time the contact touches the busbar, a new motion is started with reduced initial velocity, due to the energy dissipated in the impact [6]. To account for the higher frequency oscillations of the contact beam, we threshold the contact position obtained above so that if it is higher than a high threshold th_h, it is fully open, if it is lower than a low threshold th_l it is fully closed, and if it is in between the two, its behaviour is randomised, as observed in the measurements in [16, figure 5]. This behaviour is shown in Fig. 6, and can be controlled dynamically by the initial velocity of the contact.

[5] https://www.soundonsound.com/reviews/hammond-b3
[6] http://hammondorganco.com/products/portable-organs/xk-5/
[7] https://github.com/pantherb/setBfree

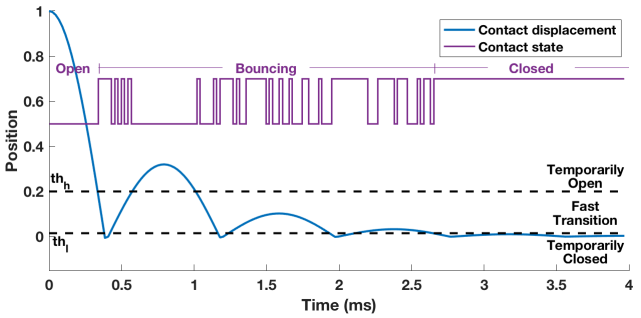

Figure 6: Synthesized contact bounce.

After a visual inspection of the audio recordings of the actual instrument (such as the ones in [16, Figure 2]), we noticed that the amount of amplitude modulation caused by the contact bounce was smaller than the full-range value it could be expected to have, possibly for a combined effect of the capacitive coupling between the contact and the busbar and of the inductance of the coupling transformer. We took this into account by introducing a tunable parameter to control the actual amplitude excursion of the bounces.

3.3 Touch responsiveness

We implemented three different mappings between key position and contact triggering. For each of them, the triggering points are adjustable.

- **M1, Single triggering point, no velocity**: the onsets of all the contacts are triggered when the key crosses a given threshold.
- **M2, Single triggering point, velocity**: the onsets of all the contacts are triggered after the key crosses a given threshold. The onset of each individual contact is delayed in time depending on the velocity of the onset, resulting in a temporal staggering effect where faster velocities will trigger all the contacts closer together, whereas slower velocities will spread them over a longer period of time.
- **M3, Individual triggering points**: each contact is set to trigger at a given threshold in the key-throw. This can be customized per-contact and per-key. The temporal staggering is due in this case to the different thresholds for each contact, which prevent them from closing synchronously, similarly to an actual Hammond organ.

For key presses that have a constant acceleration and go all the way to the bottom of the key, the behaviour of **M2** and **M3** can be virtually the same. However, for partial presses, or when the acceleration is distributed non-uniformly during the key press, then the difference between them becomes more obvious. The instantaneous velocity of the key at the triggering point, computed as the difference between the current and previous sensor reading, can be used as a proxy for the initial velocity of the contact in the contact bounce model. By combining the triggering point mappings and the instantaneous velocity we can reproduce the macroscopic behaviour of the Hammond, or of some Hammond emulators.

The Clavia Nord C2 features *"An ultra fast trigger-to-sound response time"*.[8] Our understanding is that this behaviour is achieved by triggering on the upper contact of the otherwise standard two-contact key-bed it is equipped with. As a consequence of triggering on the first contact, we expect that there will be no velocity control available to drive the sound engine. This behaviour can be approximated using mapping **M1** with no velocity control on the dynamic bounce model.

The sound engine of the Crumar Mojo keyboard[9] provides velocity-sensitive onsets, where the duration of the onset is affected by the velocity. This behaviour can be approximated using mapping **M2** and applying velocity control to the dynamic bounce model.

The behaviour of the C-3 organ measured in [16] can be approximated by using mapping **M3** with triggering points selected in the range illustrated in their Figure 6. When using **M3**, our instrument allows to reproduce some specific

[8] https://www.nordkeyboards.com/products/nord-c2
[9] https://www.crumar.it/?a=showproduct&b=4

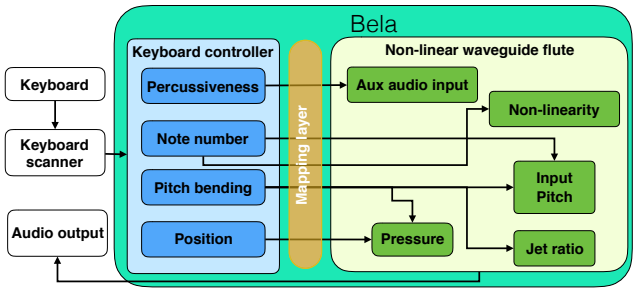

Figure 7: Block diagram of the physical modelling flute controlled with continuous key position.

Figure 8: The keyboard scanner installed on the Yamaha CP-300.

Hammond behaviour such as partial key presses (i.e.: a key press that does not go all the way down and does not trigger all of the key contacts), and very slow presses (i.e.: a key is depressed very slowly so that the harmonics start playing one at a time).

4. NONLINEAR WAVEGUIDE FLUTE SYNTHESIZER

This instrument was designed as a probe for studying the generalization of keyboard playing skills to changes in the mapping of the keyboard interface (study not presented here). It is a monophonic synthesizer based on a physical model of a flute and it associates several continuous gestures on the key with a clear sonic effect. We use an off-the-shelf weighted keyboard controller without any mechanical modifications but, by using it as a continuous controller, we attempt to extend the concept of keyboard beyond its common understanding, and we challenge some of the basic assumptions underpinning most keyboard instruments:

- **continuous control**: the key is a continuous controller, and the key position affects the sound throughout the duration of a note.
- **onsets**: onset velocity has no effect, but percussive onsets are detected and a percussive sound is produced in response.
- **interaction between keys**: pressing two neighbouring keys at the same time results in an interaction between them, producing a pitch bending gesture with the second key acting as a continuous controller on the pitch of the first.

The high-level block diagram of our instrument is displayed in Fig. 7. We fitted two boards of the piano scanner on a Yamaha CP-300 digital keyboard, covering the range from $B\flat_3$ to B_6 (38 notes). None of the sounds or electronics from the Yamaha were used, only its weighted keyboard. A picture of the instrument is shown in Fig. 8.

The starting point for the sound engine is a nonlinear waveguide physical model of a flute developed in the FAUST [10] programming language [13]. We modified the model to provide control over the length of the delay of the air jet between the mouth and the mouthpiece, so that it allows to generate overblown tones and interesting turbulent and multi-phonic timbres when set to non-integer fractions of the bore delay [8].

[10] http://faust.grame.fr/

The FAUST compiler produces a C++ file that contains the DSP code as well as wrapper code for the platform it will run on, which we modified in order to integrate it with the keyboard scanner library. Our full code is available online and implementation details can be found in [15].[11]

4.1 From discrete to continuous

The original FAUST implementation of this synthesizer would, upon receiving a MIDI note input, trigger envelopes applied to the air pressure, to provide smooth fade in and fade out of the note, and to introduce a 5Hz modulation to produce a delayed vibrato effect.

When using a continuous keyboard controller all the automations can be replaced by the player's action on the key itself, and the parameters from the physical model can be controlled by the performer's gestures:

- **air pressure** (intensity of breath): controlled by the vertical position of the current key.
- **pitch** (length of the bore): controlled by the current active key, and during bending gestures by the vertical position of the bending key.
- **jet ratio** (angle between lips and mouthpiece): changed during a pitch bending alongside the pitch parameter.
- **auxiliary audio input**: if a key is struck percussively, a percussive sound is injected into the resonant bore.

4.2 Gestures and sounds

The mapping of key position to air pressure will make so that when the player presses the key with a swift, decisive gesture, similar to a *forte* on a piano, the corresponding sound will attack immediately. Conversely, to fade in a note, they can press the key slower, as if they were playing *pianissimo*, the tone will then transition from an airy, inharmonic breathy sound to a fuller tone, richer and richer in harmonics as the air pressure increases. The intensity and timbre of the note once the key has reached the bottom will be the same in the two cases, what changes is uniquely the shape of the onset transient. To play a quieter note, the key will have to be partially pressed and held half-way through the key-throw. There is no need for the key to reach a particular point for the sound to begin, in fact the onset gesture can be infinitely long, or never reach the bottom: sound will be produced throughout. At any point in the key throw, vertical oscillating motions on the key will naturally translate into a tremolo effect. Pressing into the keybed gives access to an extended range of pressure which yields a growling sound.

A pitch bending is generated when holding one key down and progressively pressing one of the keys within a major third interval. The vertical position of the bending key then controls the pitch of the tone. Bending a note on a transverse flute is done in practice by changing the distance between the upper lip and the mouthpiece, therefore resulting in a timbral change during the bending. As the sound of a pitch change obtained by adjusting the length of the bore is pretty flat and uninteresting, we change both bore length and jet ratio, which produces a more turbulent and unstable sound, akin to the one obtained when gliding between notes on a transverse flute. A state machine is implemented in software so that, if the player lingers in the pitch bending space, the sound can become unstable and break into a multi-phonic or overblown sound, which can be used for creative purposes.

When a percussive key press is detected, a percussive sound, a pre-recorded sample of a person producing a "T" sound with their mouth into a microphone, is injected into

[11] https://github.com/giuliomoro/flute-key/

the resonant bore of the physical model through the auxiliary audio input. This is not strictly equivalent to the effect of a flutist pronouncing a "T" sound into the mouthpiece, however, the resulting "chuff" is reminiscent of the sound of a sharp attack on a flute.

Video and audio recordings of example gestures and sounds can be found online.[12]

5. IMPLICATIONS

The instruments we developed provide richer interactions with the key than most keyboard DMIs. In terms of [5]'s concept of bottleneck, we made the bottleneck of our instruments wider. Our hybrid Hammond instrument allows some gestures to produce an audible difference that would not otherwise do so on a velocity-based keyboard. The behaviour of the instrument with **M3** is somewhere between discrete and continuous: a key press no longer amounts to a single instantaneous measurement, but at the same time the continuous behaviour unfolds over a timescale that is too short for conscious control of all the detail during normal playing. However, it brings in a certain sonic richness associated with different key trajectories, which could lead performers to skilfully manipulate the character of the onset, consciously or unconsciously. Where [5] observed a progressive impoverishment of the gestures used in an instrument with a narrow bottleneck, we expect a progressive enrichment as a consequence of a wider one.

Our flute synthesizer is a more radically "continuous" instrument, where each key acts as a continuous controller and percussive gestures have a specific sonic result. One of the most important advantages of MIDI is that of *generality*: as long as a sound generator can understand note and velocity information, then it can be played with a keyboard (or other MIDI controllers). Our instrument requires the player to move the fingers on the keyboard in some new and unusual ways, due to the specific characteristics of the mapping between gesture and sound. Therefore, if we were to try and replace the sound generator, it would be an arduous task to maintain the exact meaning of gestures, and the performer would have to make an effort to adapt to the new mappings. By making the bottleneck wider, we have gained in the amount of control available, and in the *character* of the instrument, but we lost in the generality of the control data.

Some previous attempts to overcome the discrete characteristics of the keyboard interface, such as the Seaboard [7] and the Continuum [3], did so by completely transforming the mechanics of the instrument and its haptic and tactile response, eventually preserving only the spatial location of the notes. Continuous key sensing allows to control sound generators through a conventional keyboard controller with a degree of detail in certain respects similar to that of the Haken or the Seaboard, with the advantage of preserving a familiar interface for trained keyboard players, and the platform we presented can serve as the basis for new instruments of this kind. Expressive E's recent announcement of the Osmose[13] keyboard with continuous sensing and MIDI 2.0's upcoming support for high resolution per-key controllers[14] pave the way for an exciting future for continuous keyboards.

6. ACKNOWLEDGMENTS

This work was funded by EPSRC grants EP/L019981/1 and EP/N005112/1.

[12] https://vimeo.com/364675614
[13] https://www.expressivee.com/discover-osmose
[14] https://www.midi.org/midi-2

References

[1] M. Bernays and C. Traube. Piano touch analysis: A matlab toolbox for extracting performance descriptors from high-resolution keyboard and pedalling data. *Proc. JIM*, 2012.

[2] L. Fourier, C. Roads, and J.-J. Perrey. Jean-jacques perrey and the ondioline. *Computer Music Journal*, 18(4):19–25, 1994.

[3] L. Haken, E. Tellman, and P. Wolfe. An indiscrete music keyboard. *Computer Music Journal*, 22(1):30–48, 1998.

[4] R. H. Jack, T. Stockman, and A. McPherson. Effect of latency on performer interaction and subjective quality assessment of a digital musical instrument. In *Proceedings of the Audio Mostly 2016*, pages 116–123. ACM, 2016.

[5] R. H. Jack, T. Stockman, and A. McPherson. Rich gesture, reduced control: the influence of constrained mappings on performance technique. In *Proceedings of the 4th International Conference on Movement Computing*, page 15. ACM, 2017.

[6] X. Jun, H. Jun-jia, and Z. Chun-yan. A dynamic model of electromagnetic relay including contact bounce. In *2008 International Conference on Electrical Machines and Systems*, pages 4144–4149. IEEE, 2008.

[7] R. Lamb and A. Robertson. Seaboard : a new piano keyboard-related interface combining discrete and continuous control. In *Proc. NIME*, pages 503–506, Oslo, Norway, 2011.

[8] M. E. McIntyre, R. T. Schumacher, and J. Woodhouse. On the oscillations of musical instruments. *The Journal of the Acoustical Society of America*, 74(5), 1983.

[9] A. McPherson. Portable measurement and mapping of continuous piano gesture. In *Proc. NIME*, pages 152–157, Daejeon, Republic of Korea, May 2013.

[10] A. McPherson. Buttons, handles, and keys: Advances in continuous-control keyboard instruments. *Computer Music Journal*, 39:28–46, 2015.

[11] A. McPherson, R. Jack, and G. Moro. Action-sound latency: Are our tools fast enough? In *Proc. NIME*, volume 16 of *2220-4806*, pages 20–25, Brisbane, Australia, 2016. Queensland Conservatorium Griffith University.

[12] A. McPherson and V. Zappi. An environment for submillisecond-latency audio and sensor processing on beaglebone black. In *Audio Engineering Society Convention 138*, 2015.

[13] R. Michon and J. O. Smith. FAUST-STK: a set of linear and nonlinear physical models for the FAUST programming language. In *Proceedings DAFx*, Paris, 2011. IRCAM.

[14] F. R. Moore. The dysfunctions of MIDI. *Computer music journal*, 12(1):19–28, 1988.

[15] G. Moro. *Beyond key velocity: continuous sensing for expressive control on the Hammond organ and digital keyboards*. PhD thesis, School of EECS, Queen Mary, University of London, 2020.

[16] G. Moro, A. P. McPherson, and M. B. Sandler. Dynamic temporal behaviour of the keyboard action on the hammond organ and its perceptual significance. *The Journal of the Acoustical Society of America*, 142(5):2808–2822, 2017.

[17] O. Ortmann. *The Physical Basis of Piano Touch and Tone*. Kegan Paul, Trenc, Trubner & Co., 1925.

[18] D. Wessel and M. Wright. Problems and prospects for intimate musical control of computers. *Computer music journal*, 26(3):11–22, 2002.