see this poster online

THE ELECTRIC GUITAR AS A LIVE CODING INTERFACE

Martin Dupras | Simon Holland | Paul Mulholland
The Open University (UK)

read the paper

PROBLEM TO SOLVE

How can electric guitarists live code guitar processing algorithms without removing their hands from the gu[illegible]r?

Main design challenges:
(1) translating gestures into recognisable abstract data
(2) mapping strategies for variety of players and styles?

APPROACH

A PureData patch receives an audio signal for each guitar string (hexaphonic pickup) and MIDI from a Softstep2 foot controller.

The guitarist live codes by using foot controller switches to select *marshalling modes* and playing short note sequences converted in actions that navigate an n-tree of programming nouns and verbs that can be turned into PureData objects insertable into the audio processing graph.

The foot controller manages *marshalling modes* that guide listening agents to navigate, assemble and dispatch command queues, and to collect playing data such as pitch sequences, pitch sets, chords, rhythms and patterns. Programming nouns and verbs, abstract data and musical data are created, entered, and manipulated through normal guitar playing actions.

EXISTING WORK

Piano keyboard interfaces and other interfaces isomorphic to an array of buttons have been used as live coding interfaces, but not, to our knowledge, stringed instruments.

The CodeKlavier (Veinberg and Noriega (2022) uses a MIDI piano as a live coding interface. This approach exploits the symbolic (MIDI) representation of the playing to send commands during performance. The sound of the piano is always audible, and therefore the musical playing that creates the code is intrinsic to the musical performance. The piano is heard all the time and both as an audible musical performance and possible act of programming.

Because the programming actions are audible, they need to also be musically appropriate. The solution we propose would offer the choice to the player to make the programming audible or silent appropriately to musical context.

The Stenophone (Armitage and McPherson (2017) is "a live coding keyboard which is also a digital instrument." The implementation converts chord key presses into "words" and uses continuous gestural data mapped to synthesis parameters.

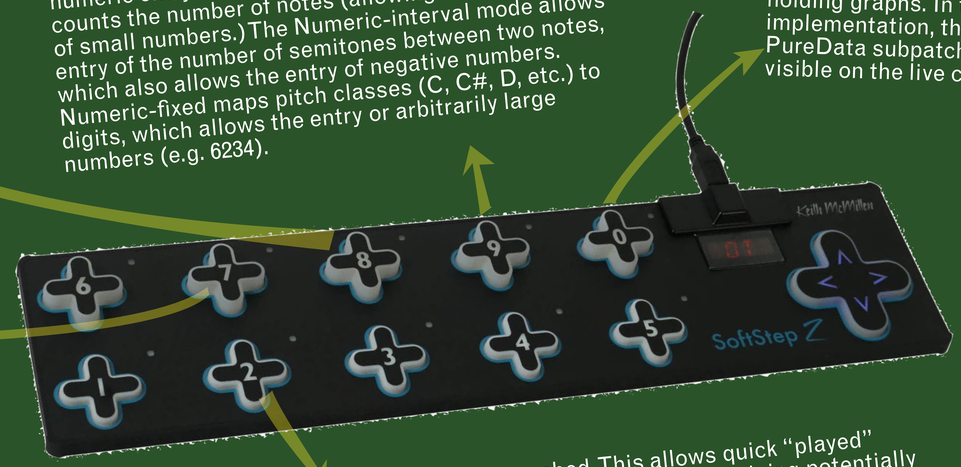
The Stenophone does not make use of learned instrumental playing skills, since it is based on the stenotype, a device used by stenographers.

PROTOTYPE

Record mode allows the recording of audio data into arrays, which can be chosen by using the Select switch and navigating a tree pointing to the available arrays using guitar playing.

Numeric mode enables entry of numeric data. Pressing the Numeric switch enters this mode if coming from another mode, or cycles through three numeric entry methods. The Numeric-repetition mode counts the number of notes (allowing the quick entry of small numbers.) The Numeric-interval mode allows entry of the number of semitones between two notes, which also allows the entry of negative numbers. Numeric-fixed maps pitch classes (C, C#, D, etc.) to digits, which allows the entry or arbitrarily large numbers (e.g. 6234).

Play mode allows the signal from the guitar to be heard by the audience, and for it to be processed by any active processing graph. Signal processing graphs are created in Codeboards, which are empty containers capable of holding graphs. In the current implementation, the containers are PureData subpatches that are visible on the live coding screen.



Capture mode allows the capture of musical "symbolic" structures such as pitch sets or grid rhythms to be captured and stored to the last selected capture container. Different types of containers (e.g. value, value range, pitch set, pitch sequence, chord, quantized rhythm) are available, visible to the player and audience, and can be chosen by pressing the Select foot switch and navigating the selection tree using the guitar playing in the same way as Prog mode. The selected container listens to the mode appropriate for its data type (e.g. a pitch-set container would listen to a pitch listener.)

Prog mode allows the creation of processing graphs. Entering Prog mode initiates a command selection mechanism which navigates a hierarchical tree of programming command elements. In this mode, fret n of the lowest guitar string selects the nth branch of the lowest level of the tree; fret n of the next string selects the nth subbranch, until a leaf is reached. This allows quick "played" navigation of a tree containing potentially hundreds of command elements. Once the desired command element ("leaf") is found, it can be queued using the Queue switch on the foot controller. The Enter switch then executes the assembled command, typically adding or modifying a node in some processing graph.

CONCLUSIONS & FUTURE WORK

Real-time programming during guitar performance raises these questions:
(1) what factors will govern how well coding and conventional playing can be interleaved artistically and effectively;
(2) what methods are fast enough to allow programming constructs to be entered by instrumental playing
(3) how flexible must the system be to accommodate a variety of playing techniques.

System needs to balance programming flexibility and expressive guitar playing.

Future work:
(1) user study to map the conceptual programming space for performers;
(2) iterative design ofsystem for live performance;
(3) user study of skilled performers learning to use the system .