2018

Tangible Virtual Patch Cords

Stefano Fasciani
University of Wollongong, fasciani@uow.edu.au

Habibur Rahman
University of Wollongong

Tangible Virtual Patch Cords

Stefano Fasciani
Faculty of Engineering and Information Sciences
University of Wollongong in Dubai
stefano@fasciani.xyz

Habibur Rahman
Faculty of Engineering and Information Sciences
University of Wollongong in Dubai
hr19392@gmail.com

ABSTRACT

This paper presents a system to tangibly manipulate the virtual patching cords in graphical programming environments, such as Max and Pure Data. The system includes a physical interface, a communication protocol, and a software library, providing physical extension of the graphical programming paradigm. The interface includes a patch bay with connectors representing signal inlet and outlets from the programming environment. When inlets and outlets are connected with patching cords, equivalent virtual connections are created at runtime. The system supports one-to-many and many-to-one connections with different signal combination schemes. The design of the hardware and software components of the current prototype is detailed in the paper, as well as possible use of the system for programming and live performances.

1. INTRODUCTION

Visual programming languages for audio and multimedia share similarities with modular synthesizers. Both include a set of basic modules for signal generation or processing, which can be patched together to implement complex and interactive sound synthesis. By default, they do not provide an instrument ready for use. In the early days, modular synthesizers were large, expensive, and used in studios by engineers or technically-minded musicians. With the advent of portable synthesizers, such as the Minimoog, switches and rotary selectors replaced patching cords for signal routing. Later, when digital synthesizers emerged, signals were routed electronically or via software. This enabled to store and recall synthesis preset using memory chips. Modular synthesizer regained popularity with the introduction of the Eurorack standard, based on the Doepfer A-100 released in 1995. In the last two decades, the number of manufacturer of Eurorack modules grew significantly, providing a wide range of small and affordable modules, including CPU-based units with esoteric functionalities and integration with computers [1]. Eurorack reignited the interest in modular systems and using patching cords is still the primary synthesis programming technique. Patching bay for signal routing are becoming popular also in standalone synthesizers, enabling creative interfacing with other machines.

Concurrently to the emerge of Eurorack modular systems, Cycling ’74 Max and Pd (Pure Data) [2] became popular for creating interactive computer music systems and multimedia works. Max and Pd are visual programming languages that include a large set of basic operators as well as complex modules from the community. Their programming paradigm resembles modular systems. Indeed, interactive or algorithmically controlled synthesis can be programmed by routing control or audio signals across modules and operators. Virtual patching cords in Pd and Max are abstractions akin to patch cables in modular synthesizers. In both domains, the physical and the virtual, the term *patch* is used to identify a specific program or signal routing configuration. In both Max and Pd, there are analogies with the vision of Don Buchla on the separation of audio signals from control signals. Virtual patching chord for messages and signals are graphically different and often incompatible, as in Buchla’s systems where control signals are routed using unscreened cables with banana plugs and audio signals using screened cables with 1/8" phono jacks. Instead Robert Moog, the other pioneer of analog synthesizers, made no distinction between the two type of signals, using screened cables with 1/8" phono jack for both [3].

The flexibility of modular system was immediately received and exploited by artists for creative purposes. Patching enables to quickly prototype novel interaction or synthesis approaches, and to explore the integration between different components. Programming by patching is visually intuitive and it requires no expertise in hardware or software development. Predicting the sonic output of complex patches is challenging also for expert users. Patching is an effective heuristic to explore of modular systems and discover appealing configurations.

Miller Puckette, a prominent developer and user of Pd and Max, recently stated that these tools “make possible for people to develop computer music applications not available in commercial software packages”, “they are programming environment that make easy for musicians to program their own applications” [4]. Matthew Davidson commented that modular systems are “non-specific, open ended, they refresh possibilities and arouses new interests”, “playing with Lego never gets old, Max is an infinite box of audio Lego”, and “musicians inherently understand patch chords” [5]. Robert Henke argued that Max enables “writing structures that make music”, “Max has the potential to change the way we think about music”, “it freed me from this linear idea that something has a start and an end”, and Max “is a tool for sculpturing music” [6]. Indeed as observed by Butler [7], the object-oriented appearance of Max and Pd patches are reminiscent of the lateral, processual and distributed objects [8].

Besides analogies in modularity and in programming paradigm, there are still intrinsic differences between the physical and virtual patching environments, especially when used in live performances. The system we present in this paper extends the virtual patching affordance of Max and Pd to the physical world, including the capability of editing the signal routing at runtime, such as in performance contexts. In the proposed design, advantages of digital programmable system are used to provide additional patching features not possible in analog domain.

2. LIVE PATCHING

Magnusson observed that the knowledge required to make and to play with an electronic musical instrument are significantly different [9] because users are not required to be familiar with sound synthesis and music theory. However, he argues that this holds only when dedicated physical interfaces are provided and embodied practices are possible. When using instruments providing only graphical interfaces, the symbolic channel includes only elements such as menus, cables, icons, or other abstract representation of the instrument's variables. In this case the digital instrumentalist must be also a luthier familiar with the system he is playing with. We can extend this observation also to modular systems that, despite having a tangible interface, expose users only with technical settings and parameters strictly related to the sound synthesis process.

Live electronics performances featuring modular synthesizer are increasingly popular due to the success of the Eurorack format. Performers often showcase complex patches that require minimum interaction due to the use of on-board sequencers. The physical gesture is limited to simple actions such as adding or moving patch cords, pushing buttons, and turning rotary knobs. These actions require little mastery, but the performer is required to be deeply familiar with his patch and his modular system [10]. Auricchio and Borg describe musicians performing with modular synthesizers as ringmasters of a sonic-circus, because they are detached from obvious sonic changes, exhibiting a control over the instrument at a high level [11]. This forces the audience to a reduced mode of listening because they ignore how sound is being produced [12]. Software emulation of modular synthesizer has been introduced as well, such as the open-source Eurorack-like VCV Rack [13], or the g200kg WebModular[1] running in the browser. The VCV Rack provides replicas of existing modules or novel units developed by the community, enabling users to experience a similar workflow and interaction paradigm of modular system.

Max and Pd have been extensively used to program interactive computer music application for live performances. These can be easily integrated with control interfaces supporting MIDI or OSC protocols. Generally, such interfaces include elements such as buttons, faders, rotary knobs, encoders, and pressure sensitive pads. These can be mapped to continuous- or discrete-value variable of the synthesis algorithm. Max and Pd have also been used to develop standalone performance systems such as P2Live [14] and Music_SDP [15]. Virtual cords in a Max or Pd patch can be easily added or re-routed by using mouse and keyboard, or by scripting software that parse and modify the .maxpat or .pd file. However, changes in the DSP chain require a rebuilt process and the interruption of the audio output for a few milliseconds. Patching in the virtual environments is possible only at programming stage and not while performing. This represents a major difference with hardware modular systems.

Objects providing signal routing functionalities are available, such as gate~ and selector~ in Max, and multiplex~ and demultiplex~ in Pd. These can be controlled by external interfaces, but the signal routing is still limited by the design choices made at patching phase. Moreover, these objects operate in a binary mode, determining audible clicks when signal routing is changed. Max objects such as vst~ and poly~ are not affected by this issue and can be used to dynamically change signal processing aspect of the patch, but they do not provide explicit support for signal routing. The matrix~ object, available in both Max and Pd, is suitable for signal routing at runtime because it includes a non-binary mode with signal fade-in and fade out when connections are created or removed. The functionality of this object resembles the signal routing pin matrix featured in the EMS VCS 3. Open Music Labs developed the rePatcher[2], a system for the signal routing in Max and Pd using real hardware wires. It features an Arduino shield with a 6x6 routing matrix implemented with PCB female headers, and six rotary knobs, as visible in Figure 1. The rePatcher allows users to dynamically route signals across 6 input and 6 outputs of the provided abstraction for Max and Pd by patching wires in the related headers. No specific functionality is pre-assigned to the rotary knobs.

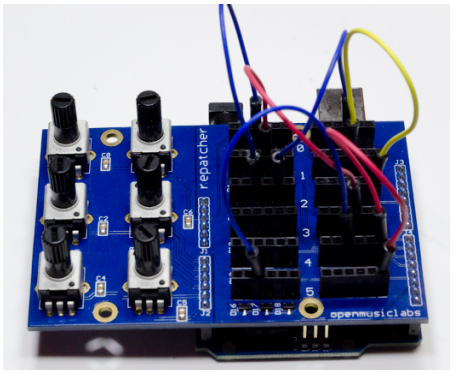

Figure 1. Open Music Labs reParcher shield mounted on the Arduino Uno board.

The rePatcher is based on the matrix~ object, and it mirrors its functionality in the physical domain. It allows input routing to multiple outputs, and sum of multiple inputs to a single output, which is the default behavior in Max and Pd when connecting two signals cords to the same inlet of an object. Although minimalistic and poorly ergonomic due the small size and type of connectors, the rePatcher enables live patching of audio signals in Max and Pd. A similar approach, based on breadboard patching, has been taken in the D-Box to allow user hacking

[1] https://www.g200kg.com/docs/webmodular/

[2] http://www.openmusiclabs.com/projects/repatcher/

the internal functionality of a digital musical instrument [16], while the Patchwerk [17] has been designed with the opposite aim of remotely control a large analog modular synthesizer via graphical user interface.

3. SYSTEM DESIGN

The system we designed enables users to create or modify at runtime virtual connections in Max and Pd using tangible patch cords. These are plugged on an interface which connectors represent signal output from and signal input to the virtual environment. No electrical signal, either audio or control, flow through the patching cables. These are used to open or close circuits – one for each outlet-to-inlet pair – which status is read by a microcontroller and sent to the computer, where an abstraction running in Max or Pd implements an equivalent signal routing. The system includes two layers of software, one running in the microcontroller and one running in Max or Pd. These enable to extend the functionalities beyond the electrical limitations of modular synthesizers. The hardware design and software implementation detailed here are available as open-source[3].

3.1 Functionalities

3.1.1 One Outlet to Many Inlets

The system allows to propagate a signal from one outlet to multiple inlets by using stackable connectors on the patching cords. This configuration is also possible on modular synthesizers. It is equivalent to attaching multiple loads to a driver in parallel, but users must ensure that the load is still within the acceptable range.

3.1.2 Many Outlets to One Inlet

The system allows to propagate the signal from multiple outlets to a single inlet by using stackable connectors on the patching cords. This configuration is connector-wise possible on modular synthesizer, but not recommended as it may damage the modules. Connecting two drivers in parallel will determine unpredictable voltage on the load, and a significant amount of current may leak across the drivers if their output voltages are different. Instead, this configuration is common in Max and Pd, with the software performing the sum between the two signals. In our system, we extend this possibility beyond the default behavior. By operating a switch on the interface, users can choose between two different combination modes. In the current implementation, we included sum or product between signals. The product is the digital equivalent of a frequency mixer implementing a double-sideband suppressed carrier amplitude modulation. This nonlinear operation creates new frequency contents in the output signal, which may violate the Nyquist theorem and generate aliasing. As observed by Puckette [18] this is one of the most common violation of the sampling theorem in electronic music practices. Mitigation strategies, such as digital filters, are not included in the system.

3.1.3 Variable Fade-In and Fade-Out Time

When plugging or removing a physical patch cord, the corresponding virtual connection is established or deleted by fading in or fading out the corresponding routed signal. Fade time is controllable through a rotary knob. The value can be changed at runtime within a user-defined range. Allowing at least a few milliseconds of fade time eliminates the audible clicks introduced by changes in signal routing. Varying the fading time provides an additional control dimension for creative use, which is not possible on hardware modular systems.

3.1.4 Level of Inlet Signals

Each inlet is paired with a rotary knob that allows to attenuate or amplify the resulting incoming signal. These has been included because more than one outlet can be connected to any inlet, using sum or product mode, and levels may exceed the expected range, generating clipping distortions.

3.1.5 Batch Patching

The interface can be set temporarily offline by operating a switch. Changes implemented by moving patching cords or rotary knobs are be transferred to the computer environment all at once when the interface backs online. This feature enables switching between patching configurations that require moving multiple cords, without stepping through audible intermediate stages.

3.2 Hardware Implementation

The AVR Atmega2560 8-bit microcontroller on-board the Arduino Mega 2560 is the core of the system. The large number of I/O available on this device allowed to integrate 22 signal outlet connectors, 22 signal inlet connectors with 22 associated rotary knobs for level control, 1 additional rotary knob for the fade time, 1 SPST switch for inlet signal combination mode selection, 1 SPST switch to set the interface offline or online, and 1 LED to display system activity. The signal routing capability of the system is equivalent to the 22 by 22 routing matrix of the EMS VCS 3. In the current implementation, there are still 2 available I/O to further extend the functionalities, such as integrating other signal combination modes, or providing additional visual feedback.

All elements are installed on the top panel of the interface, as visible in Figure 2. For the patch cords, we selected banana plugs as the connection is established as soon as male and female are juxtaposed. There is no need to fully lock the connector, as for phono jacks. Banana plugs support the creation temporary or intermittent connection. Red female connectors on the top of the interface represent the 22 signal outlets, whereas the black female connectors below represent the 22 signal inlets. Connectors are organized on staggered rows to minimize the

[3] http://www.stefanofasciani.com/tavipaco.html

width of the top panel. The 44 female connectors are directly wired to standard I/O pins of the Atmega2560.

The 23 rotary knobs are implemented with 10 kΩ linear potentiometers, with 100 nF ceramic capacitors inserted between GND and the output pin. The capacitors act as low pass filters contributing to eliminate noise and stabilize the potentiometer output voltage read by the ADC. Atmega2560 features a single 10-bit successive approximation ADC with a 16-channels on-chip multiplexer. We further increase the number of channels integrating the 8-channels analog multiplexer 74HC4051, driven by 3 I/O lines of the Atmega2560. This provides a total of 23 analog inputs – converted sequentially by a single ADC – that we use to read the single-ended output voltage of the potentiometers. As visible in Figure 2, the 22 potentiometers controlling the inlet levels are placed below the respective black connectors, while the potentiometer controlling the fade time is placed on the right edge of the top panel. The Arduino Mega 2560 board is powered using an external 9V power supply because the limited current provided via USB is not sufficient to power the large number of potentiometer circuits.

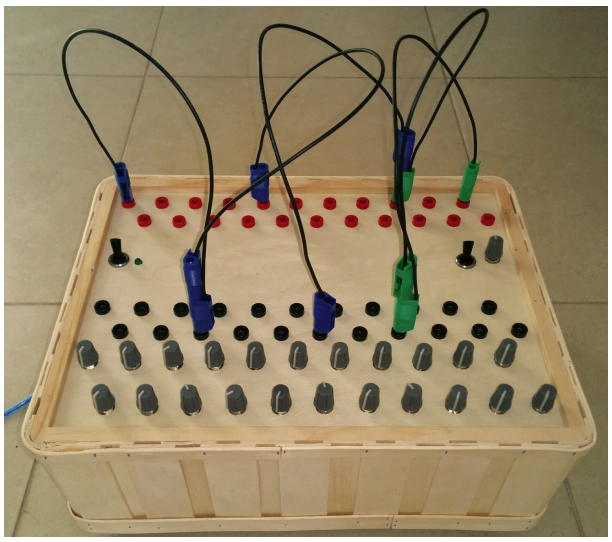

Figure 2. Physical interface of the system with a few patch cords between outlets and inlets.

The 2 SPST switches are in active-low configuration, connected between GND and the Atmega2560 inputs, with the internal pull-up resistors enabled. The circuit of each switch includes a 10 kΩ resistor in series to limit the current flow, and a 100 nF ceramic capacitors between the poles of the switches for de-bouncing purposes. Banana plugs on the patch cables are stackable to support the one-to-many and many-to-one signal routing. We use plugs of two different colors. These do not present functional differences but allow users to color-code connections created in sum or product combination mode.

3.3 Communication Protocol

The computer and the Arduino Mega 2560 communicate via serial UART over USB. We use a baud rate of 76800 bps which is the highest synthesizable by the Atmega2560 using the 16 MHz on-board oscillator, and with an error rate below 1%. The protocol has been designed to minimize the amount of data exchanged between the interface and the computer. There are three types of messages: Mode, Matrix, and Analog. The first type has size of 1 byte, the remaining two are composed by packets of 3 bytes each. The most significant bit of the first exchanged byte is always 1, followed by 2 bits used as message type identifier. The most significant bit of the second and third byte is always 0, providing robustness to the protocol as incomplete messages can be identified and discarded, as implemented in the message parser presented in Section 3.5. The complete protocol is detailed in Table 1.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Mode **Byte 0**	1	0	0	-	-	-	-	Op
Matrix **Byte 0**	1	1	0	-	-	-	-	Co
Byte 1	0	-	-	Outlet ID [4:0]				
Byte 2	0	-	-	Inlet ID [4:0]				
Analog **Byte 0**	1	1	1	Channel ID [4:0]				
Byte 1	0	-	-	-	-	ADC Val [9:7]		
Byte 2	0	ADC Val [6:0]						

Table 1. Communication protocol from the Atmega2560 to the computer running Max or Pd.

In the Mode messages, the single bit field 'Op' identifies the combination mode for multiple outlets connected to the same inlet, where 0 identifies sum and 1 product. In the Matrix messages, the single bit field 'Co' identifies open circuit if equal to 0, and close circuit otherwise, between outlet and inlet identified by the 5-bit fields in the second and third bytes of the packet. In the Analog messages, a 5-bit channel identifier is included in the first byte, whereas the 10-bit result of the ADC conversion is distributed across the remaining bytes. A single byte message is sent from the Max or Pd to the Atmega2560 to enable or disable the interface.

3.4 Firmware

The firmware running on the Atmega2560 continuously checks all interface elements, compose the messages according to the communication protocol, and send these via serial UART. In order to minimize the communication overhead between interface and computer, messages are sent only if the status of switches, rotary knobs, or connections has changed, similarly to Firmata [19]. This is implemented using a memory structure holding a snapshot of the interface status at previous iteration. However, at startup, or when the interface is re-enabled via Max or Pd, a full status update is transmitted. This includes 1 Mode message, 484 Matrix messages, and 23 Analog messages. The ADC converts output voltages from the potentiometers after selecting the appropriate channel via internal and external multiplexers. After initialization, analog messages are sent only if the difference between

previous and current value is larger than a unit, as the accuracy of the less significant bit of the Atmega2560 ADC is poor, determining fluctuating values.

The interface includes 22 signal outlets and 22 signal inlets, resulting in 484 circuits that users can create using patch cords. Any arbitrary number of these circuits can coexist at any time. They are virtually implemented and individually checked by the Atmega2560 firmware. All Atmega2560 pins connected to inlet and outlet connectors are set as inputs, with pull-up resistors enabled. This ensures no floating states for disconnected inputs and no current leakage in case pins are wired together by patch cords. The firmware sets one outlet pin at a time as digital output, driving a low voltage. Then it quickly reads sequentially the 22 inputs pins related to the inlets. A connection with the current outlet is detected if the input voltage is detected as low. If the inlet is disconnected or connected to another outlet, the input voltage is high due to the internal pull up resistors. This procedure is repeated for all 22 outlets and it is summarized in the pseudocode of Figure 3, where Matrix is a bidimensional array containing the current status of the 484 outlet-to-inlet circuits.

```
for i := 1 to 22 step 1 do
        set outletPin i outputLow
        for j := 1 to 22 step 1 do
                if inletPin i is low
                        Matrix[i][j] = Close
                else
                        Matrix[i][j] = Open
        end
        set outletPin i inputPullup
end
```

Figure 3. Pseudocode checking the status of the 484 outlet-to-inlet virtual circuits.

With no changes detected and no messages sent via serial UART, the firmware running on the Atmega2560 clocked at 16 MHz completes a full check of all interface element every 6.3 ms. Instead when all elements change state and a total of 508 messages are sent, it takes approximately 182 ms. These measurements represent the absolute best- and worst-case latency of the interface. However, as the interface is operated by human, we do not expect to find more than a few changes per iteration, determining a typical latency of approximately 7 ms. To compute the overall system latency, we must also include the contribution of Max or Pd, which depends on the audio I/O buffer size selected by the users.

3.5 Max and Pure Data Abstractions

We developed Max and Pd abstractions that routes virtual audio signal in the software environments by mirroring the patch cords connected to the interface. The abstraction integrates the serial interface for the Arduino Mega 2560, a parser decoding the received messages, and the signal routing with functionalities detailed in Section 3.1. The abstractions for Max and Pd are functionally equivalent, but present some implementation differences. In particular, the Max implementation uses JavaScript for the parser and gen~ for the core signal routing elements. These are not available in Pd, and the abstraction is implemented using only native standard objects. The layout of the Max abstraction *TaViPaCoMax* is visible in Figure 4. The first inlet is dedicated to commands which include: verbose mode to print parsed messages; range of fade time in ms – controllable with the dedicated rotary knob; range and exponential scaling for the inlet signal levels; interface enable; serial port utilities such as display of available ports, port selection, and polling interval in ms. The first outlet bangs when serial messages are received, providing minimalistic monitoring of interface activity, mirroring the LED on the interface panel. The remaining 22 inlets and outlets are those dedicated to connect the signals related to the interface outlets and inlets. The *TaViPaCoPd* abstraction for Pd presents an equivalent interface.

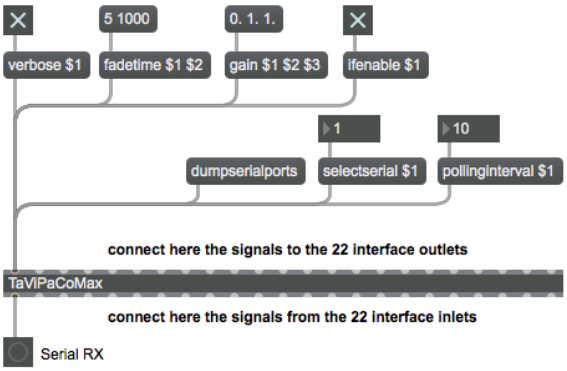

Figure 4. Layout and commands for the Max abstraction TaViPaCoMax.

The *TaViPaCoMax* includes 22 sub-abstractions for routing the 22 interface outlets (inlets in *TaViPaCoMax*), to each interface inlet (outlet in *TaViPaCoMax*). In turn, each of these sub-abstractions includes 22 additional sub-abstraction for fading in/out the incoming signals, and 2 gen~ object performing sum or product between the signals routed to the same interface outlet. All signals routed to the same interface outlet are combined using the same mode, combining sum and product is not supported as not uniquely defined in some configuration. When multiple patch cords are connected to the same interface signal inlet, these are combined using the mode associated with the last created connection. As in Max or Pd all signals exist at all time, even if the associated patch cord is not plugged. The fading abstractions ensure that these have the correct value. In particular, disconnected signals are set to 0 if the mode is sum, whereas these are set to 1 when the mode is product. In the Max version only, signal sum and product are implemented in gen~ using CodeBox. Users can change to different signal combination schemes by editing a single line of code for the entire system. As described above, the rotary knobs on the interface control the level of the inlets' signals after combination. These can be easily repurposed to control the out-

let signal level within the *TaViPaCoMax*, or routed outside the abstraction to control other parameters.

In *TaViPaCoMax* abstraction there is a total of 22 gen~ objects performing the sum of 22 signals and 22 gen~ objects performing the product of 22 signals at all time. These account for the great majority of computational load, requiring 22.1 million additions and 22.1 million multiplications per second when working with a sampling rate of 48 kHz. We measured the computational load at 48 kHz on a MacBookPro8,2 and a MacBookPro13,3. Both feature Max 7.3.4, OSX 10.13, and a quad-core Intel i7. The more recent machine has a clock speed of 2.9 GHz, 8 MB of L3 cache, 256 KB of L2 cache per core and a front bus speed of 2133 MHz. The older machine has a clock speed of 2.4 GHz, 6 MB of L3 cache, 256 KB of L2 cache per core and a front bus speed of 1333 MHz. The results are detailed in Table 2 and reported for the maximum and minimum value of the I/O vector size. The load refers to the percentage displayed in the Max Audio Status window, and it has been measured with no active GUI elements and no external objects connected to the abstraction. In the table, we also report the load of a simplified version, which supports signal sum mode only. The computational load is constant and independent of the number of patch cord inserted in the physical interface, whereas the Max native object matrix~ presents a load that increases with the number of active connections, which is less reliable for live performances contexts.

	MacBookPro8,2		MacBookPro13,3	
Vector Size	32	2048	32	2048
CPU Load	25%	18%	13%	11%
sum-only CPU Load	18%	11%	11%	8%

Table 2. CPU load of the TaViPaCoMax in various configuration and running on different machines.

4. CONCLUSION & FUTURE WORK

We presented a system that extends the virtual patching affordance of Max and Pd to the physical world, enabling runtime signal routing as in modular synthesizers with additional features such as sum or product of signals, and fading time. The system is scalable and works with a different number of inlets or outlets, with additional signal combination schemes, and with different interfaces. A second prototype is currently under development and it features patch board matrix similar to the one included in the EMS VCS 3. The signal routing is implemented inserting pins instead of patch cords. We removed rotary knobs to minimize the size and automated the inlet signal level control. In non-structured user studies with 3 subjects we explored practical applications of the current prototype. Besides using the system for patching from simple oscillators to complex processing modules, users highlighted the novel performing potential of the controllable fade in/out time, and the rich modulations obtained with the product mode. They also explored the use of uncoated patch cords and conducting gloves to create additional temporary connections by tangible interaction.

5. REFERENCES

[1] M. Dalgleish, "The Modular Synthesizer Divided: The Keyboard and Its Discontents," *eContact!*, vol. 17.4, 2016.

[2] M. Puckette, "Pure Data," in *Proc. of the 1996 International Computer Music Conference*, Hong Kong, China, 1996, pp. 224–227.

[3] T. J. Pinch and F. Trocco, *Analog Days: The Invention and Impact of the Moog Synthesizer*. Harvard University Press, 2002.

[4] M. Puckette, "Limitations of Computers as Musical Tools," DMARC Seminar Series, University of Limerick, 2017.

[5] M. Davidson, "Cycling74 Perspectives: Stretta," 2010.

[6] M. Henke, "Cycling74 Perspectives: Robert Henke on Max," 2009.

[7] M. J. Butler, *Playing with Something That Runs: Technology, Improvisation, and Composition in DJ and Laptop Performance*. Oxford, New York: Oxford University Press, 2014.

[8] G. Born, "On musical mediation: Ontology, technology and creativity," *Twentieth-century music*, vol. 2, no. 1, pp. 7–36, 2005.

[9] T. Magnusson, "Of Epistemic Tools: musical instruments as cognitive extensions," *Organised Sound*, vol. 14, no. 02, p. 168.

[10] Navs, "Basic Electricity: An appeal for a greater understanding of rudimentary modular functions," *eContact!*, vol. 17.4, 2016.

[11] N. Auricchio and P. Borg, "New modular synthesizers and performance practice," presented at the SAE 2nd Synthposium, Melbourne, 14-Nov-2016.

[12] P. Schaeffer, *À la recherche d'une musique concrète*. Paris: Seuil, 1952.

[13] A. Belt, *VCV Rack*. 2016.

[14] R. Graham and M. Mus, "A live performance system in pure data: Pitch contour as figurative gesture," in *Proc. of Annual Pure Data Conv.*, 2001.

[15] H. Lobel, M. Mann, D. Berlinerblau, and J. C. Spoon, *Music_SDP The Music & Sound Design Platform*. 2015.

[16] A. McPherson and V. Zappi, "Exposing the Scaffolding of Digital Instruments with Hardware-Software Feedback Loops," in *Proc. of New Interfaces for Musical Expression*, Baton Rouge, Louisiana, USA, 2015, pp. 162–167.

[17] B. D. Mayton, G. Dublon, N. Joliat, and J. A. Paradiso, "Patchwerk: Multi-User Network Control of a Massive Modular Synthesizer.," in *Proc. of New Interfaces for Musical Expression*, Ann Arbor, 2012.

[18] M. Puckette, "The Sampling Theorem and Its Discontents," *Array*, vol. 17, 2017.

[19] H.-C. Steiner, "Firmata: Towards Making Microcontrollers Act Like Extensions of the Computer.," in *Proc. of New Interfaces for Musical Expression*, Pittsburgh, 2009, pp. 125–130.