NICOLAS COLLINS

THE DEVELOPMENT OF THE !TRUMPET[1]

INTRODUCTION

The !trumpet is software synthesis system controlled from, and playing back through, a trumpet. It is not an electronically extended trumpet (like those of Ben Neill, Axel Dörner or Jonathan Impett, among others): the player produces no acoustic sounds by blowing through the mouthpiece. Instead, breath pressure and valve movement on the brass instrument are read by an embedded Arduino microcontroller and sent to a laptop, where the data is mapped onto various parameters in synthesis software; the resulting electronic sound is returned to the trumpet, where it plays through a loudspeaker inside the bell, and is further processed *acoustically* by valve position (changes in the length of tubing filter the speaker output), movement of a plunger mute (wah-wah style filtering), and orientation of the instrument in space (panning).

The built-in speaker gives the !trumpet a self-contained acoustic quality, rare among electronic instruments, that blends well with more conventional instruments on stage. The speaker is constrained to the bandwidth of a conventional trumpet (mid- to high-frequencies), but the performer can direct a full-range signal to stereo line outputs for connection to a PA system when bass frequencies or higher sound levels are desired.

The mute contains seven momentary switches for controlling various functions in the software. Switch closures are sent to the Arduino on the trumpet body via an infrared link (similar to a TV remote control). Two additional momentary switches, mounted on the trumpet itself, control the routing of the audio to the built-in speaker and the line output.

In a nod to David Tudor's legendary composition *Bandoneon !* I dubbed this instrument "!trumpet". But where Tudor employed the "!" to indicate "factorial", I use the sign for its logical property of negation: this is definitely *not* a trumpet[2].

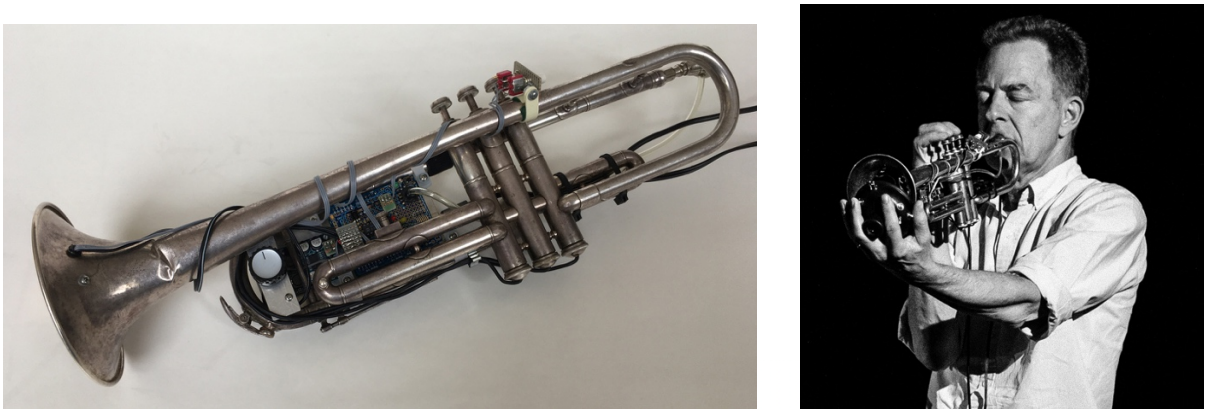


Figure 1: !trumpet

BACKGROUND

The !trumpet is the latest iteration of a design concept that dates back to instruments I built as an undergraduate at Wesleyan University in the 1970s: adapting conventional instruments for the *acoustic* manipulation of electronic sound. Under the influence of Alvin Lucier I composed a number of pieces that used feedback to articulate acoustical characteristics of architectural spaces, and eventually extended those techniques to "playing" acoustic instruments [3]. In *Feetback* (1975), for example, I fitted small microphones inside the mouthpieces of two brass or woodwind instruments, and connected them to speakers (high-frequency horn-drivers) coupled to the mouthpieces of two other instruments[4]. The players change fingering and spatial orientation of the instruments to elicit different feedback pitches as they walked through the performance space – using feedback to overblow the harmonic series' of the instruments as they intersected with that of the room.

Figure 2: Drivers used for saxophone (left) and trombone (right), 1975.

In 1982 I built the first of a series of "backwards electric guitars": electric guitars whose pickups are wired to the speaker outputs of amplifiers, so that the strings can be resonated with sound (similar to shouting into a piano with the sustain pedal down); chording and dampening the strings filter the sounds[5]. As with the feedback-driven wind and brass instruments, the overtones of the guitar strings were elicited electronically instead of through the usual playing techniques.

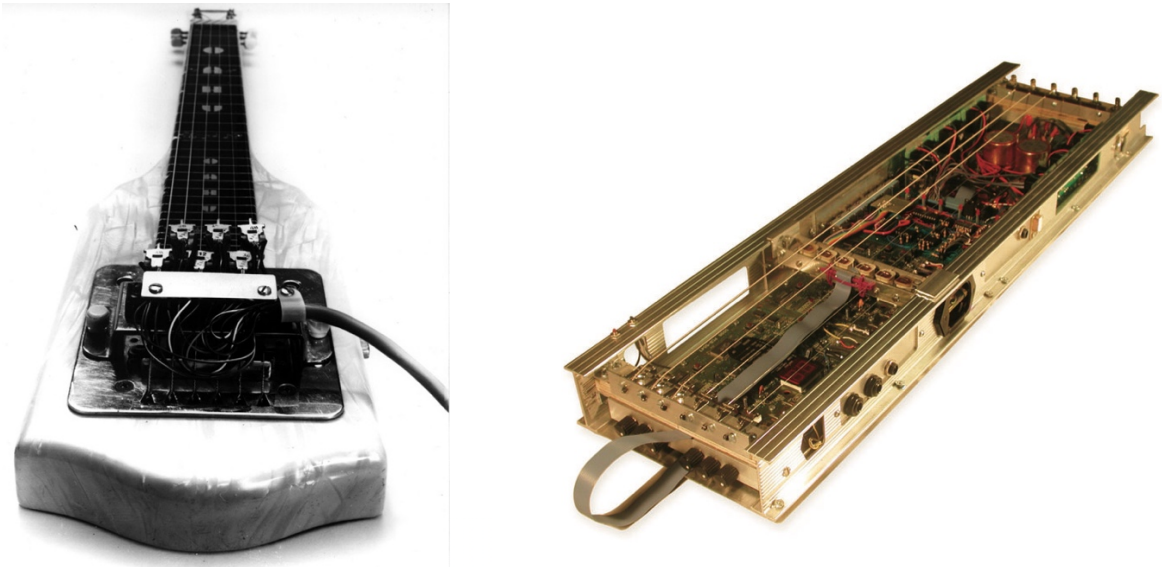


Figure 3: Backwards electric guitars, 1987 (left) and 2002 (right)

In 1986 I added a small keypad to a speaker-loaded trombone, linked an optical shaft encoder to the slide, and wired the instrument to a home-made digital signal processor and sampler. By pressing switches and moving the slide I could increment and decrement values in a computer program – in effect clicking and dragging a mouse without having to look at a screen. The resulting sounds, articulated by breath, played back through the trombone, where they were further filtered acoustically by the movement of the slide and mute, and directed spatially by aiming the instrument like a flashlight. An early entrant into the world of alternative controllers that sprang up in the age of MIDI, the "trombone-propelled electronics" overlaid the acoustic transformations of the speaker-equipped trombone on the emerging vocabulary digital sound processing[6].

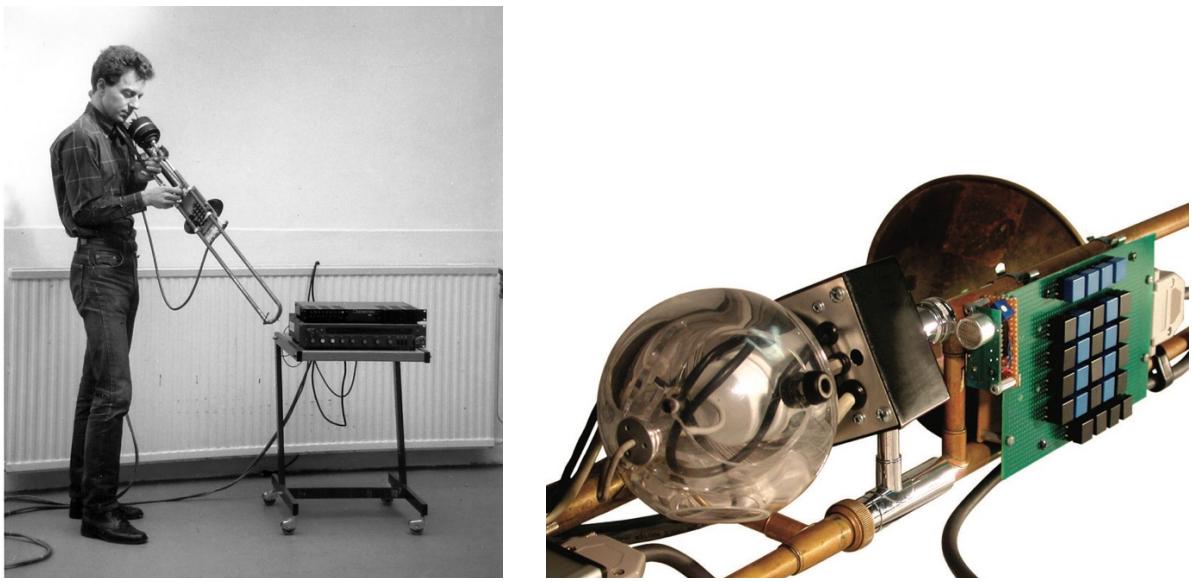


Figure 4: Trombone-propelled electronics, first version (1988, left) and last version (2005, right)

The acoustic manipulation at the heart of each of these hybrid instruments -- filtering feedback or sound sources in tubes or strings -- could have been accomplished in simple purpose-built devices: telescoping PVC plumbing tubes, guitar strings on a wood plank. But from early on I was drawn to creative re-use of found objects over pure invention, and I preferred to stress the connection between possibly unfamiliar experimentation and extant musical practice.

Initially developed for a specific composed work, *Tobabo Fonio* (1986)[7], the "trombone-propelled electronics" opened the door to improvisation. This was one of the first practical systems for live sampling, and thanks to its self-contained acoustic identity, it merged easily with more conventional instruments on stage. I spun variations out of fragments of sound grabbed from my fellow players, moving from mimicry to the unexpected[8]. The original instrument was crushed beneath the wheels of a taxi at Schipol airport (Amsterdam) in 1994, but over the next dozen years I built two variations that updated the core concepts of live sampling and signal processing using new technologies. By 2008, however, commercially available looping devices had matured and proliferated to the point that, as Robert Poss pointed out, "anyone can buy your trombone in a pedal"[9]. Bored with the sonic vocabulary that had enthralled me for over three decades, I retired my last trombone[10].

My entry into electronic music pre-dated personal computers and came through homemade circuitry instead. Although I began working with early microcomputers in the 1970s, I kept my soldering iron warm – experience taught me that sometimes hardware offered a more efficient (or at least different) path through a musical thicket than software could. MIDI came and went and was replaced by DAWs and programming languages like Max/MSP and SuperCollider. But by the time I joined the faculty of the School of the Art Institute of Chicago in 1999, glimmers of anti-digital backlash could be detected in rise of Circuit Bending and the revival of analog synthesis. At my students' request I introduced a course in hardware hacking, which in turn led to a book and workshops around the globe[11]. The experience of sitting in a large room surrounded by 20 novice hackers, each with her own speaker, rekindled my interest in the glitchy sounds and chaotic structure of circuit experimentation after years of immersion in more rational digital soundscapes. I began composing more hardware-centric works.

Salvage (Guiyu Blues) (2008)[12] was inspired in equal part by:

- The sounds that arose from the workshop tables, which evoked the glitchy texture of my earliest electronic work, and that of my peers and mentors in the pre-computer 1970s. They contrasted refreshingly with the sampling, looping, granulation and other DSP-based effects so prevalent at the time.
- The peculiar form that these sounds took in the hands of a room full of hackers: the semi-controlled chaos of parallel but unsynchronized experimentation with similar, not-easily controlled circuits.
- The desire to shrink my silicon footprint. I had recently read about the environmental impact of electronic recycling in the town of Guiyu in the Guangdong province of China, and I decided to make a pointed, if token, gesture of reducing that toll by one circuit board.

Out of a bank of six oscillators from a single CMOS 74C14 IC I constructed an instrument that would extract sound from any dead, landfill-bound circuit board: computer, cell phone, printer, television, etc. The frequency range of each voice, from sub-sonic through audible pitch to ultrasonic whistling, is determined by a choice of a fixed capacitor; the precise pitch would typically be set by a resistor in the feedback path from its output to its input, but in my implementation each output and input is connected to test probe (like that on a multimeter) instead. When a pair of probes is pressed against traces on the dead circuit board, the frequency is determined by whatever components lie in the path between them – usually a complex (and unknowable) array of diodes, transistors, integrated circuits and other components, rather than a simple resistor. As a result, moving the probes across the surface elicits an unpredictable and unstable sequence of pitches -- given enough time, a player could learn to associate specific sounds to specific points on the grid of the board, but corroded contacts and shaky hands increase uncertainty. This is a work for six players, each holding a pair of probes. When multiple probes touch the board, short-circuits and feedback between them further complicate the sounds, as does the mixing of voices through diode multiplication (rather than simple resistor addition). In 2014 I adapted the core technology of *Salvage* for solo performance, replacing the 12 probes with a dozen small fishing weights (lead shot) that I rolled beneath my fingers across a scavenged circuit board[13].

Figure 5: *Salvage* and *The Royal Touch*

In February 2017 I attended a concert by musicians from the Merce Cunningham Dance Company[14]. Despite former music director John Cage's longstanding aversion to improvisation, and the presence of titled compositions on the program, a spirit of improvisation ran through the evening. I was struck that night by how much I missed the musical and social aspects of

improvisation. None of the systems I'd developed in the previous decade had the improvisational potential of the abandoned trombone – the chaos of *Salvage* was more interesting (to my ear) within the structure of a composed work than in a sextet free-for-all. While my trombone had arisen from my interest in the intersection of acoustical and *digital* transformations of sound, now I began looking for a way of combining a familiar acoustic armature with my newfound affection for the *analog* sonic world of chaotic circuitry, in the hopes of arriving at a new instrument.

I chose a trumpet for the core ($30- on eBay): smaller and lighter than the trombone, but still capable of acoustic transformation of an embedded speaker, with control by breath, and with the potential for mapping the valves top three continuous controllers, in lieu of a single slide. Initially I ran down two technologically divergent but ultimately unfruitful alleys: a Pd-on-Raspberry-Pi option that kept veering in the familiar direction of signal processing, despite my best efforts to avoid this; and an analog circuit in the style of *Salvage* that was chaotic but too limited in dynamic range and sonic variation to be broadly applicable in improvisation. After some weeks of reflection I came to acknowledge the hobbling of a methodological bias that I had perpetuated for years.

Since my first exposure to the Kim-1 microcomputer in 1977 I had divided my musical resources into three distinct categories according to what I perceived as their "intrinsic" strengths:

- *Hardware* (circuits and physical instruments) was great for nuanced control, interesting sounds, and instability.
- *Software* was best for interfacing controllers, making logical decisions, and generating compositional structure.
- *People* made choices (not always predictable) based on musical assessment and personal preference.

For decades I had distributed these resources accordingly, in both my composed and improvised work. I was not the only one guilty of this tendency: in the heyday of MIDI composer Ron Kuivila lamented, "we need to make computer music that sounds like electronic music" – and this was no mere semantic distinction, they did sound different[15]. In 2011 I went so far as to write a paper on the musical implications of the choice of hardware or software in compositional and performance practices[16]. But in the aftermath of two false starts on a new instrument I decided to confront these assumptions head on and attempt to write a program that would behave less like software as I knew it and more like unstable circuitry in the hands of a novice hacker.

I began by modelling a network similar to those in *Salvage* and *The Royal Touch*: a bank of square wave oscillators, controlled by a non-linear array of values, with the voices mixed down by multiplication. The challenge lay in emulating the odd affordances of the range of unknown components distributed across an unfamiliar dead circuit board.

My prototype (in Max/MSP) consisted of:

- A set of three square-wave oscillators.

- The frequency of each oscillator is set by a value in a table (the *collective* object), filled 256 random numbers.
- The table value is indexed from the position of a linear *fader* object.
- Moving the fader from the bottom to the top causes the oscillator to jump from one pitch to another, rather than ascend in a linear fashion. But because the table was static, the sequence was repeatable rather than different on each pass (as if the fader simply generated a new random number at each increment). This produced a similar effect to the non-linear adjacency of moving probes across a fixed but unknowable array of components on a circuit board.
- The timing of the transitions from one frequency to the next is randomized, between instantaneous to a noticeable glissando, mimicking the charge and discharge of capacitors in the probed circuit.
- A small fluctuation is added to each table value so that the frequency of the oscillator slowly drifts, in imitation of unstable contact between the probes and circuit traces, and of temperature change in the components on the board.
- The three voices are mixed by multiplication (the *~ object), complicating the sound in the manner of a ring-modulator or the diode mixing scheme of *Salvage*[17].

A modest amount of programming produced a surprisingly convincing emulation of the hardware antecedent – satisfying enough to justify the rather more time-consuming adaptation of my eBay trumpet as a controller, and subsequent programming of a more extensive software environment.

DESIGN GOALS

I began with the following general guidelines:

- The instrument should occupy a sweet spot between controllability and unpredictability, so that the player would have to engage improvisationally with the instrument itself, as well as with the other musicians. Every improviser periodically longs for the freshness and accidents that accompanied first picking up the instrument, lost to the rise of control and virtuosity over time; I wanted to program an instrument that frustrates virtuosity by remaking itself every time it's turned on.
- The majority of my circuits, programs and instruments have been compositionally specific, each with a finite sound palette or processing vocabulary; most traditional instruments, by contrast, have elastic borders, especially in the hands of good improvisers. This instrument might not have as full a range of kinesthetic nuances and affordances as a "real" one, but it should be able to evolve, through the programming of new voices, whenever it feels too constrained.
- The hardware and software should be sufficiently fixed to present a stable environment for performance, but open enough to incorporate additional voices and moderate changes in control structure. The instrument itself should be able to evolve much like a player evolves and expands technique on a more "closed", traditional instrument.

I chose to preserve several characteristics of my earlier trombone-propelled electronics:

- A performable instrument that combines a physical controller with a local acoustic presence.
- The familiar form of a conventional acoustic instrument, rather than a purpose-built device.
- Breath articulation of volume (expressive dynamics are sorely lacking in a lot of electronic music, especially in the post-2000 revival of analog synthesis).
- Controller data extracted from native functions of the original instrument (i.e., slide on the trombone, valves on the trumpet).
- Embedded speaker, with acoustic transformation by slide/valves and mute.
- Secondary line output to the PA for when wider frequency response and louder sound are wanted (everybody loves bass).
- Not linked to a specific composition like the circuits and software of recent works.
- Optimized for improvisation, group and solo.

But the new instrument should deviate from the trombone in a number of ways:

- It should be more portable: a smaller instrument (suitable for the overhead bins on planes), with lighter electronics.
- Whereas the speaker in the trombone was attached to the mouthpiece, subsequent experiments demonstrated that a speaker mounted in the bell was more efficient, with better bass response (back-pressure seems as sensitive to changes in tubing length as transmission from mouthpiece); both the speaker and amplifier could be smaller and lighter.
- Abandon completely the vocabulary of live sampling and signal processing.
- Instead write software that sounds and responds like the hardware from *Salvage*, *The Royal Touch*, etc.: glitch, non-linear adjacencies, instability, unpredictable glissandi.
- Multiple simultaneous continuous controllers (three valves), instead of a single data wheel coupled to the slide of the trombone. Traditional trumpet technique assigns each valve two states, up or down, with half-valving occasionally used as an effect. But in this instrument each valve would function like a slide pot, with continuous linear output over its full excursion. The net result should be less like the click and drag of single parameters in software, and more like the multi-axis response of gestures on most conventional instruments.
- Fewer switches and simpler control structure than the trombone (which had two dozen main keys and a few function keys to remap them), for more intuitive performance. Holding the trumpet with the right hand suggests a small number (5-7) of switches in the mute for left hand, to complement the continuous valve data. An exercise in self-discipline.
- Adaptations in software of the trumpet's fundamental acoustic characteristics and performance tropes: overblowing the harmonic series, valve-based pitch intervals, etc.
- Multiple voices, with a wide range of sounds, rather than a single core technique like the sampling and multi-tap processing of the trombone.

- The Arduino should serve as a "dumb" interface, converting sensor data into a serial stream to the host laptop, to reduce the need to modify its code unless new sensors are added.
- Laptop software (Max/MSP) should be structured to facilitate both playing and updating:
    - Same software format for all voices, as far as practical.
    - Controllers mapped as similarly as possible for all voices.

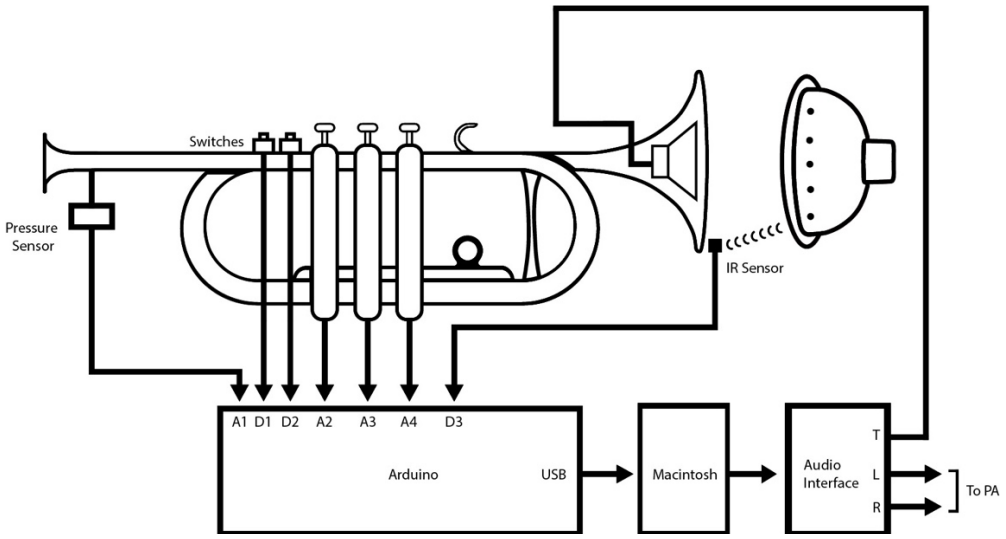


Figure 6: !trumpet system

THE HARDWARE

To the physical trumpet I added:

- An Arduino Uno mounted between the legs of the long crook (a third version of the instrument uses a smaller Arduino Mini Metro).
- A sensor to measure the position of each of the three valves: a small neodymium magnet is embedded in a rubber grommet fitting into the cup at the base of each piston, with a linear Hall Effect sensor (Sprague UGN3503) is set in the screw-cap at the base of the valve housing. The sensor outputs a voltage proportional to the strength of the magnet field, which increases exponentially as the magnet approaches.
- An air pressure sensor (NXP MPXV5010) connects to a plastic tube running to a hole drilled in the cup of the trumpet mouthpiece. The stem of the mouthpiece is blocked with silicon caulk; blowing into the cup is measured as pressure at the sensor.
- Two momentary switches are mounted on a small bracket near the valves, accessible by the thumb of the right hand when holding the instrument.
- A speaker (Aurasound NS2-326-8AT) is embedded in the bell of the trumpet.
- An infrared (IR) detector module is mounted inside the bell, to pick up signals from the mute (see below).
- A daughter board mounted on the Arduino contains:
    - Smaller headers connecting the sensors, switches and the IR detector in the bell.

  - The air pressure sensor.
  - A second IR sensor to pick up the mute when off axis from the bell, and a CMOS logic chip (CD4093) to mix the two IR signals.
- A class D mono amplifier circuit board (TPA 3118) is mounted between the legs of the tubing and connects to the trumpet speaker. Audio level is controlled by a stepped potentiometer. (The third version of the instrument uses an externally package version of the same amplifier board.)
- A lightweight bundle of six-meter cables connected to the trumpet circuitry:
  - A USB cable carries power to the Arduino and serves as a bi-directional serial interface between the Arduino and the laptop.
  - A shielded cable carries voltage from a remote 12-volt DC power supply (2000 ma) to the amplifier board. Optionally a remote battery pack can be substituted for the power supply for mains-free performance (Abenic 12v 1800mAh rechargeable LiPo).
  - A second shielded cable carries audio from the laptop audio interface to the amplifier and trumpet speaker. (The power cable and audio cable are replaced by speaker cable in the third instrument, connecting to the external amplifier.)

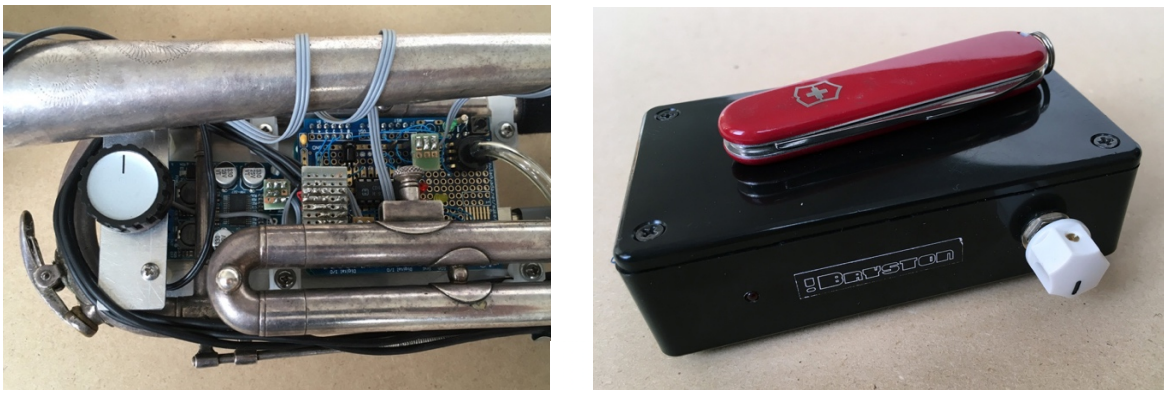


Figure 7: On board circuitry showing built-in class-D amplifier with volume control (left), and remote amplifier for 3rd revision of the instrument (right) (Swiss Army knife for scale)

Figure 8: valve sensor: Hall effect sensor (left), and magnet embedded in valve base (right)

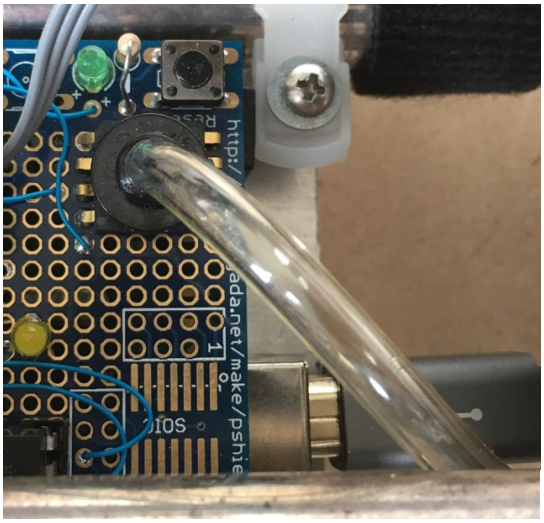


Figure 9: air pressure sensor and hose coupling to mouthpiece

Figure 10: switches near top of valves

Figure 11: speaker in bell, with IR detector at top

A toilet plunger cup has been adapted as a mute and secondary controller:

- Holes were drilled for seven momentary switches that align with the four fingers and thumb when holding the plunger as a mute. (The functions of these switches are explained in the **Software** section of this paper, below.)
- The switches are wired to a programmable infrared transmitter circuit (Tauntek IRMimic). Switch closures are sent as infrared data to the IR sensors on the trumpet.
- A small LiPo (Lithium Polymer) battery provides power for the transmitter circuit, with on/off toggle switch.

- A charging circuit for the LiPo battery is embedded in the threaded neck of the mute. This connects via a mini USB jack to a cable from a charger as needed (any phone charger or USB jack on a laptop can be used).

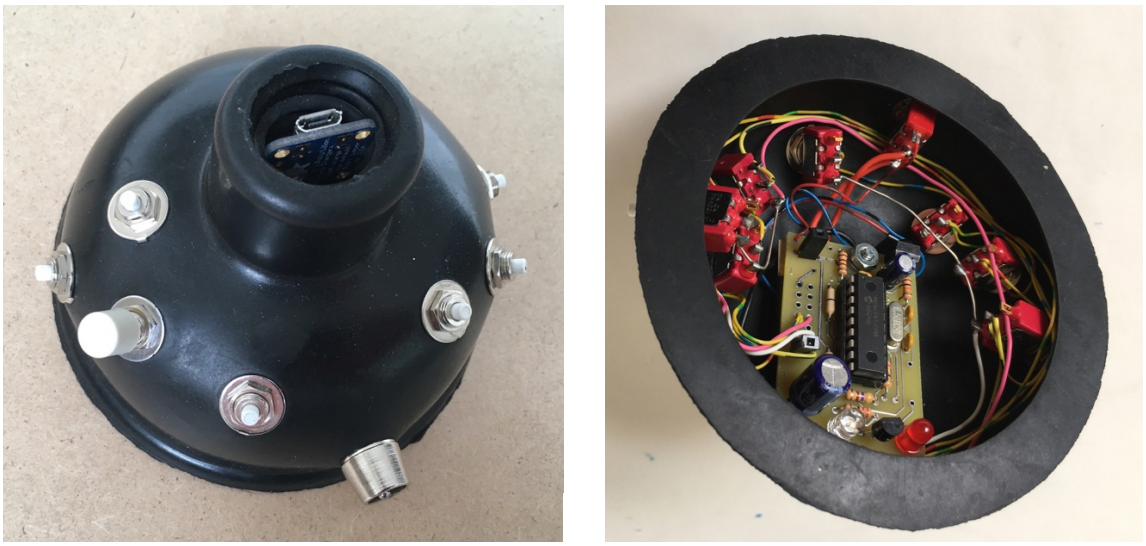


Figure 12: mute, exterior with switches, interior with circuitry

THE SOFTWARE

Software is divided between two processors: an Arduino Uno mounted on the trumpet, which communicates via USB to a Macintosh PowerBook.

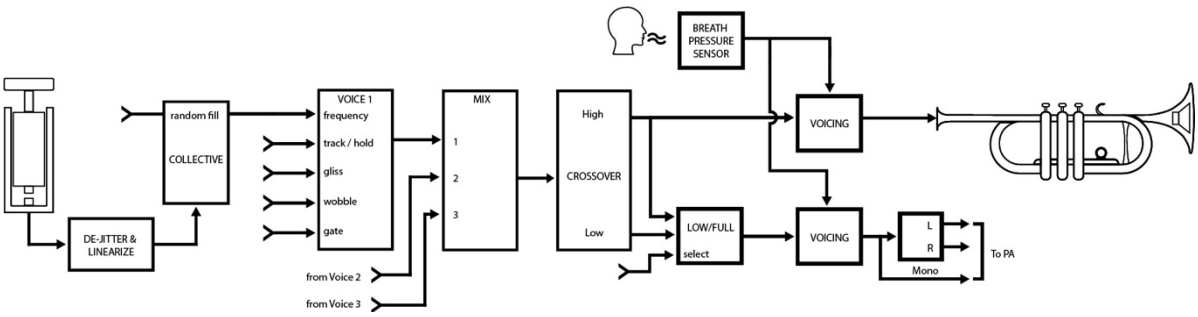


Figure 13: flowchart of software

ARDUINO

The Arduino polls the various sensors and switches on the trumpet and mute, and formats the data for transmission to the laptop:

- Three analog inputs read voltage outputs from the Hall Effect sensors measuring the position of the valves.
- One analog input reads the voltage output from the pressure sensor measuring breath strength.
- Two digital inputs read the state of the two momentary switches mounted near the valves.

- One digital input receives the data from the two IR detectors (mixed through the CMOS 4093).
- All sensor data is compiled into a single serial stream sent via USB to the laptop.

MACINTOSH

Software on the Macintosh is written in Max/MSP. The program is divided into a number of sub-patches that handshake with the Arduino, map sensor data to sound parameters in 20-odd voice programs, and perform other functions in support of digital sound synthesis. The patches provide visual feedback (meters, number values, color-coded indicators, etc.), but these are intended primarily for de-bugging and rehearsal purposes. The software can be played from the trumpet with no need for visual feedback from the computer screen or interaction with the keyboard, with the occasional exception of the large-font elapsed time display (see **tptdisplay** below), which functions as a handy stopwatch during performance.

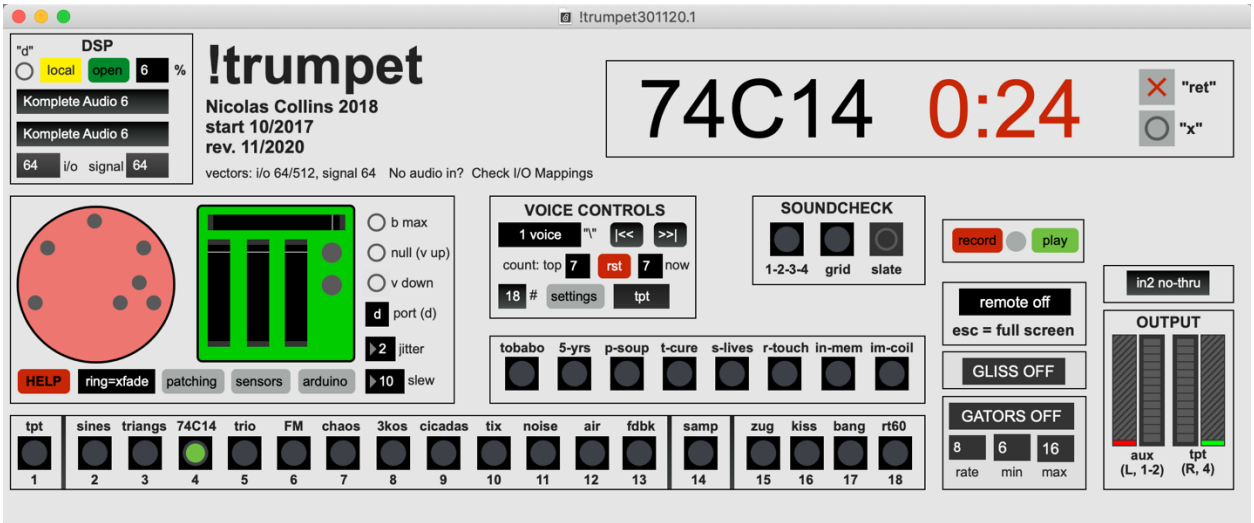


Figure 14: !trumpet main patch

Core Patches

On the left side of the main patch window the **trumpet interface** patch requests data from the Arduino every 10msec and breaks the data out into the individual sensor bytes. When launched, the program calibrates the zero state of the breath and valve sensors (no breath, valves up); an on-screen button sets the maximum level for breath and the lowest position of the valves, to account for the player's breath strength and any mechanical slippage in the valve sensors (the player "tunes" the instrument by clicking this button after power-up, if necessary). Since the strength of the magnetic field drops with the square of the distance between the magnet and sensor, a log converter linearizes the valve data. *Number* boxes and *umenus* specify the serial port and set the slew time and jitter rejection for the analog sensor data, which is sent on to the rest of the program, along with the states from the switches on the trumpet and mute. All this data is displayed graphically in this patch. Hot buttons in the window ("patching", "sensors", "arduino") open sub-patches for inspection and trouble-

shooting, and a red “HELP” button opens a short text-file summarizing control assignments and ASCII key mapping from the laptop.

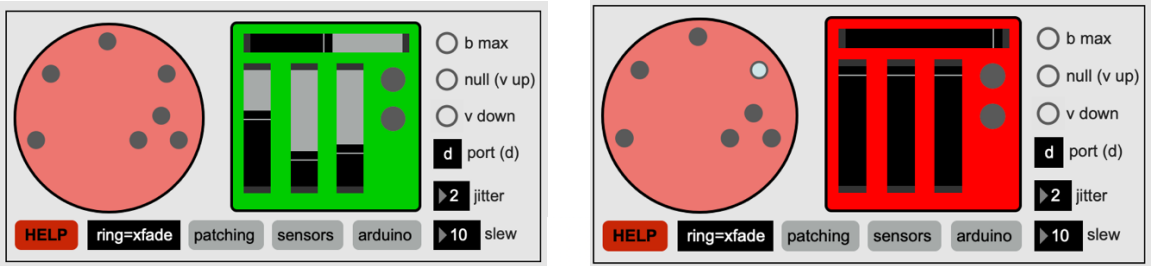


Figure 15: trumpet interface, showing active valves & breath (left), mute switches (right)

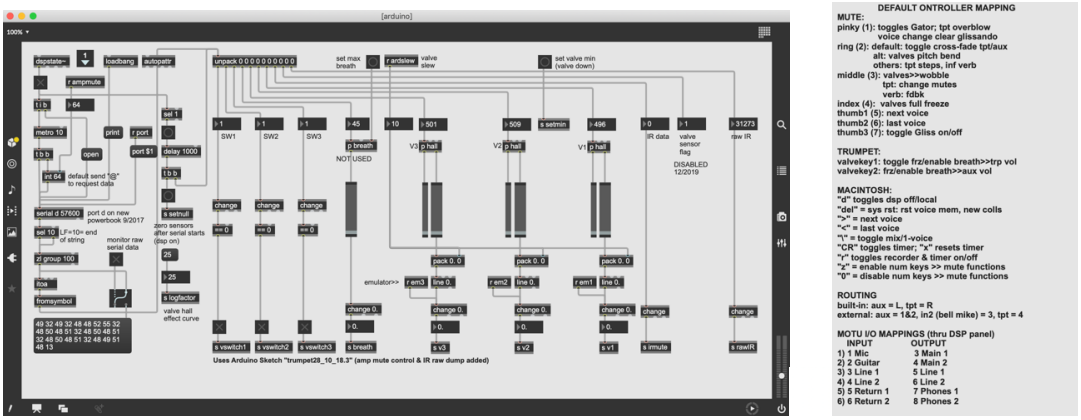


Figure 16: Arduino processing (left) and HELP pop-up (right)

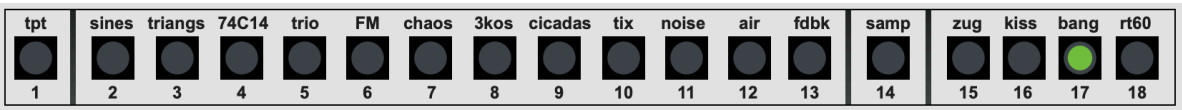


Figure 17: voices bank

Along the bottom of the main window the **voices** patch contains the core sound-generating routines. The majority of these share a similar structure and control-mapping to preserve consistent playing technique across the instrument:

- Each voice contains three sound generators – oscillators, filtered noise sources, pulse trains, sample playback modules, etc. (See **Inventory of Voices**, below, for detailed description of specific voices).
- The pitch/rate of each sound generator is set by values read out from a *collective* (table or array) of numbers, 32 – 254 values, depending on the voice.
- The values in each *collective* are generated randomly (*random*) when the program is launched. Each generator in each voice has its own individually randomized *collective*.
- Each *collective* is indexed by the linear output from one of the three valves.
- In contrast to the two-state behavior of the valves in a traditional trumpet, here each valve is treated as a continuous controller (like a slide pot).

- Because valve position controls the generator via the randomized *collective*, the player cannot predict the sequence of frequencies that pressing a valve will produce, unlike direct control from a slide pot.
- But since the *collective* is static, its sequence will repeat every time the corresponding valve is moved, so that the performer becomes more familiar with the mapping the longer the voice is played.
- The outputs of the three generators are mixed together, sometimes by simple linear addition (*+~*) but usually by multiplication (**~*), which produces a more complex signal, similar to the effect of a ring-modulator.
- The transition time from one *collective* value to the next is controlled by a *random* number, so that sometimes the pitch/rate changes instantly and sometimes it sweeps noticeably, like a glissando. There are two modes for transition time, fast (default) and slow, selected from the mute. The glissandos mimic the charge and discharge of capacitors often heard in hacked circuits.
- When the valve is not moving, the value from the *collective* is "wobbled" within a small range. This produces a slow drift in pitch/rate whenever the valve is still, mimicking the effects of temperature change and corroded contacts in analog circuitry. Wobble can be disabled from one of the switches the mute.
- A *gate* function can be toggled on and off. When disabled (default) the voice is always active, and heard whenever articulated by the breath (see **output** below). When the gate is enabled, each generator in a voice is turned on for a short period of time whenever valve moves a millimeter, then snaps off until the valve indexes the next value. This produces a percussive sound, in contrast to the more continuous default character of most voices. Variables of gate behavior, such as on-time, are set in the **gatormaster** patch (below).
- Each voice is embedded in a *poly~* object, so that its DSP functions can be turned off when the voice is inactive. The !trumpet program contains some two-dozen voices; shutting down unused processes reduces the load on the CPU to less that 10% for most voices.

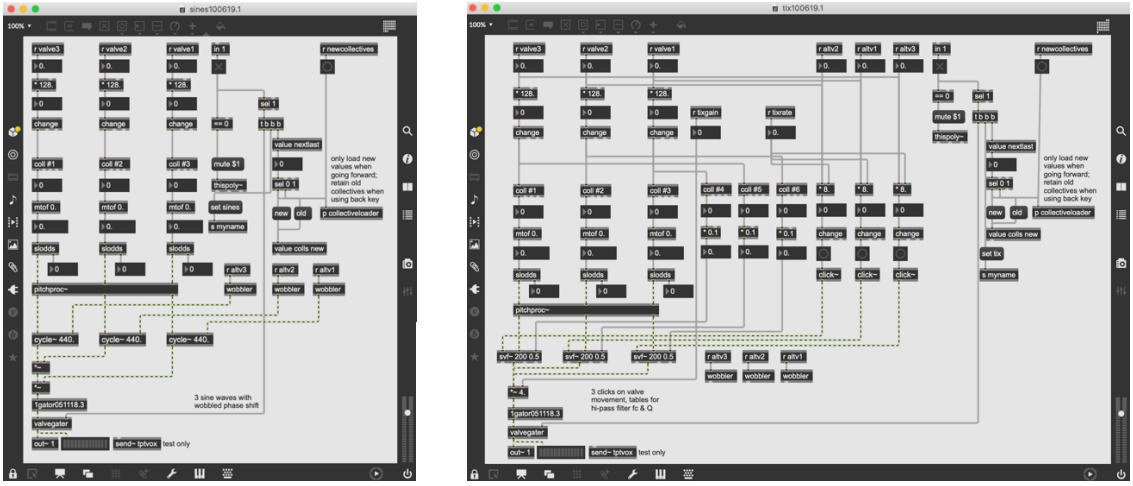
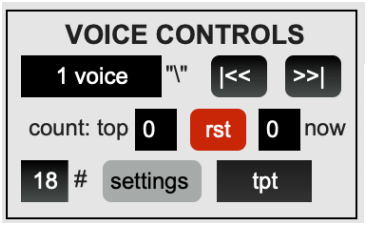


Figure 18: two examples of a voices: *sines* (left) and *tix* (right)



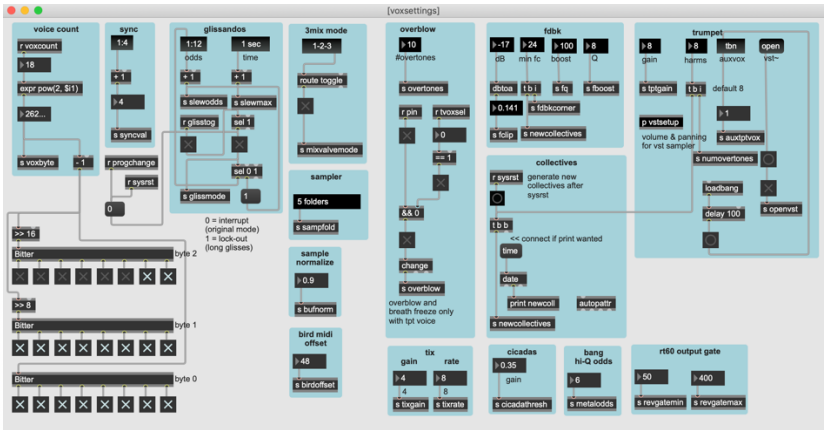


Figure 19: voicecontrol (left) and pop-up settings screen (right)

The **voicecontrol** patch is the shell for selecting and managing the sound synthesis routines, including mapping sensor data to sound parameters. One of the three mute switches activated by the thumb randomly selects a new voice from the available set; a second mute thumb switch provides an "undo", stepping back thru the sequence of selected voices. The *collectives* for each voice are randomized every time the voice is called up, except when it is returned to with the "undo" command, in which case the old *collective* is recalled. By default only one voice is active at a time; the "\" key on the Macintosh keyboard toggles mode between one voice and three voices active, for greater complexity and density. The "settings" on-screen hot-button opens a patch for setting numerous global and voice-specific parameters.

The switches on the **mute** are mapped to various functions in the software:

- Thumb 1: randomly select the next voice.
- Thumb 2: undo voice change (step back to previous voice).
- Thumb 3: toggle glissando mode between fast and slow.
- Index finger: freeze *all* software changes in the voice except breath articulation. Moving the valves now does not index through the *collectives*, "wobble" is disabled. As a result, the only audible changes to the sound are performed *acoustically*: valve movement filters the sound through changes in the length of the air column, mute movement produces a wah-wah effect, aiming the instrument reflects sound around the room, etc.
- Middle finger: freeze *collective* control of the voice frequency, like the Index finger, but links each valve's movement directly to the "wobble" factor of the active sound generator in place of the automatic slow sweep. This effectively remaps the valves from coarse non-linear frequency control (via the *collectives*) to fine linear adjustment (c. 5% of range). This is useful for fine-tuning a "frozen" sound.
- Ring finger: A *umenu* in the **trumpet interface** patch toggles the function of this switch between two modes:
    - Swaps the audio output between the trumpet speaker and PA, for rapid alternation of signal presence.

    - Freezes all default software changes in the voice, like the Index finger, but remaps valves to direct linear control of frequency over a one-octave range – larger changes than the wobble enabled by the middle finger, but linear unlike the default *collective* control.
- Pinky finger: toggles gating function on and off.

(A few voices call up voice-specific re-mapping of one or more mute switch.)

The **output** patch controls the routing and articulation of the voices to the trumpet-speaker and line-outputs to the PA. By default, blowing into the mouthpiece controls the volume of the signal sent to the trumpet-speaker; one of the two momentary switches near the valves (accessed by the thumb of the right hand) freezes this volume setting to keep the trumpet-speaker on without having to keep blowing (lazy-man's circular breathing), or to mute the trumpet-speaker when only the line outputs are wanted. The second switch near the valves similarly enables and freezes the line outputs. A crossover splits the audio output into two bands. By default frequencies above 200 Hz (roughly the lowest pitch on a Bb trumpet) are sent to the trumpet-speaker, and those below 300 Hz are sent to the PA outputs – the trumpet serves as the tweeter, the PA as the woofer. But if the line outputs are *on* when the trumpet-speaker is *off*, the full frequency range is sent to the line outputs (compensating for the lack of trumpet-tweeter). When the software is run with a simple stereo audio interface (like the built-in audio on the Macintosh) the right output channel is sent to the trumpet-speaker and the left is sent as a mono output to the PA. With a multi-channel interface the line signal is "stereo-ized" through a phase-difference network and sent out channels 1 and 2, channel 4 is sent to the trumpet-speaker, and channel 3 serves as an auxiliary output for a few specific programs. Connecting certain audio interfaces automatically opens interface-specific patches for routing and monitoring audio signals.

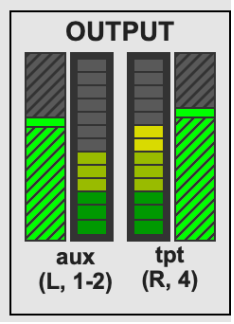


Figure 20: output

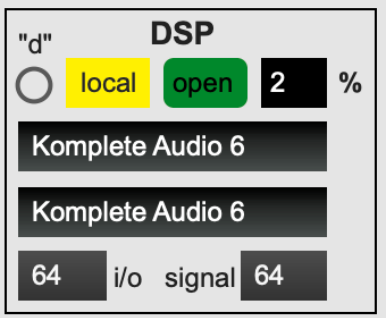


Figure 21: DSP control

Auxiliary and Utility Patches

The **DSP** panel in the upper left selects the audio interface and enables DSP functions. When an external interface is connected to the computer, it is listed in the pull-down *umenu* and automatically selected for audio I/O; if the interface has four or more output channels the audio output is routed accordingly (see description of **output** patch, above). DSP can be enabled for the !trumpet patch and sub-patches only ("local") or all Max/MSP patches open on the desktop ("all"). Turning on the DSP also enables ASCII hot-keys to control functions within

the !trumpet program. This panel includes a readout of percentage of CPU being utilized in the program, a hot-button for opening the Max's full DSP configuration panel, and *umenus* for setting the size of the i/o vector and signal vector.

Figure 22: tptdisplay

The **tptdisplay** window in the upper right displays the name of the voice currently active and the elapsed time since the DSP was turned on or the software was reset. On-screen buttons, linked to ASCII hot-keys, start and stop the timer ("return") and reset it ("x"). In a large font, this panel serves as a useful performance stopwatch, legible some distance from the computer.

Above the **voices** patch **compvoices** contains Max patches for several composed works, some of which do not make use of the physical trumpet instrument. This patch is included to streamline changeover between improvising with the !trumpet and performing other software-based works, running pieces that require routing external signals through the audio interface. The various sub-patches are wrapped in *poly~* objects like the main voices, to minimize CPU load. Selecting any of the sub-patches sometimes remaps the function of the mute switches or valves if the !trumpet is used in performance, and often open composition-specific auxiliary patch windows. This patch is expanded when new Max-based works are added to repertoire.

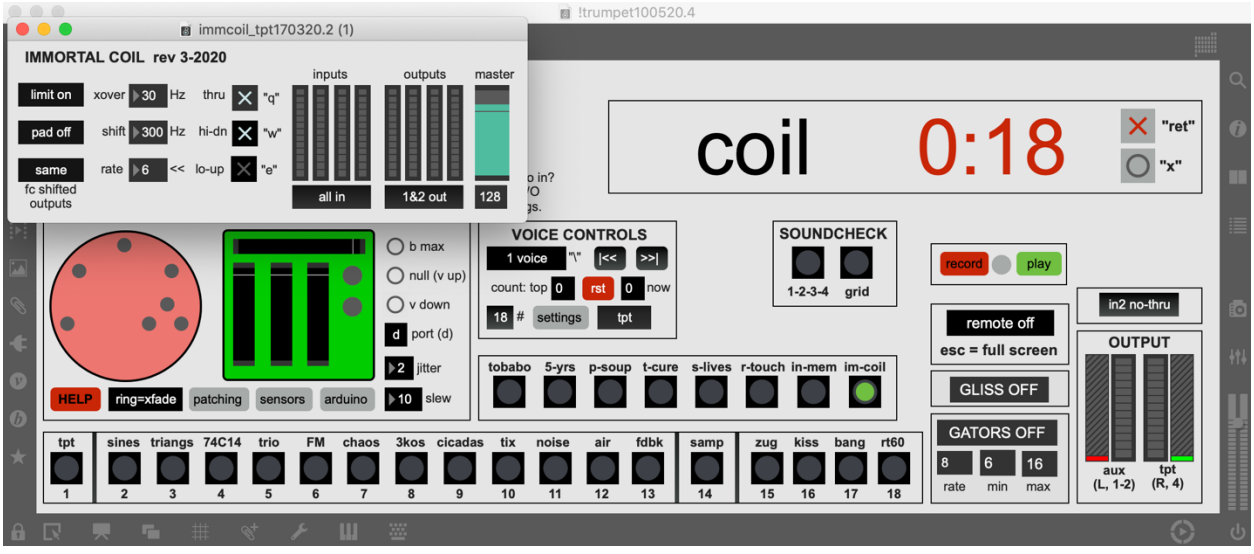


Figure 23: example of auxiliary window opened from compvoices patch

Glissdisplay indicates the state of the glissando mode: "gliss off" and grey background = fast glissandos; "gliss on" and green background = slow glissandos.

Gatormaster sets the range of behavior for the self-gating of the voices:

- "Rate" sets how many times in the excursion of each valve the gate is switched on.
- "Min" sets the minimum on-time for the gates, in msec.
- "Max" sets the maximum on-time for the gates, in msec. Each time a gate is turned on its on-time is set to a random value between "min" and "max".

"Gators off" and grey background = gates disabled; "gators on" and green background = gates enabled.

R**emoteblack** is a utility for muting video projection used for some of the pieces run from the **compvoices** panel.

Recplay records a seven-track AIFF file directly to hard drive of the computer:

- The stereo line output going to the PA.
- The mono line output feeding the trumpet speaker.
- Four input signals from the audio interface, if available. Typically these consist of:
  - A pair of microphones recording the overall sound in the room.
  - A small electret capsule mounted inside the bell in a pressure-zone (PZM) configuration, which provides a very vivid representation of the acoustic modulation by the vales and mute (figure 25).
  - An additional microphone placed at a short distance from the trumpet (the instrument has a fitting to hold a modified headset-mounted vocal microphone).

This patch provides a fast, easy way to document performances with no external recording equipment. The "play" button opens a simple file player that mixes the five tracks down to stereo for checking the recording or making a rough stereo mix file.

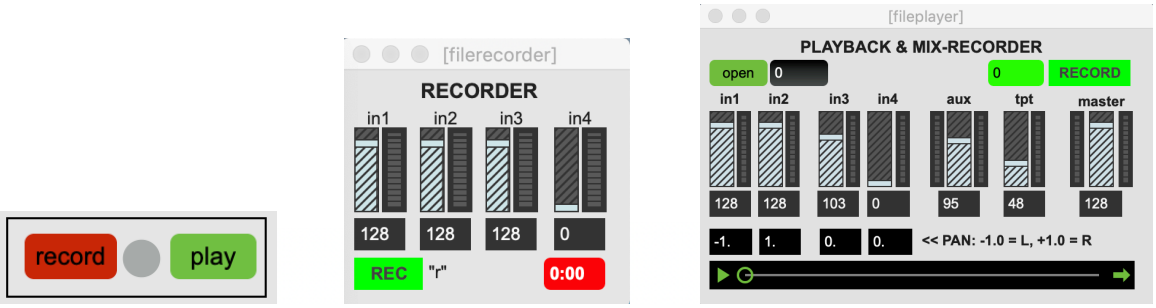


Figure 24: recplay patch, file recorder panel open, file player panel open

Figure 25: pressure-zone electret microphone mounted inside bell

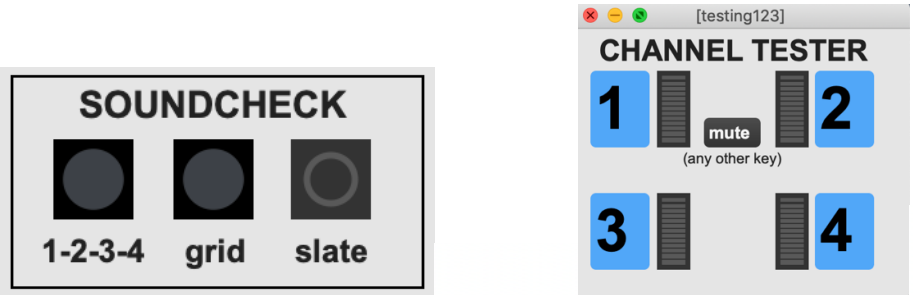


Figure 26: soundcheck panel (left), 1-2-3-4 audio channel tester (right)

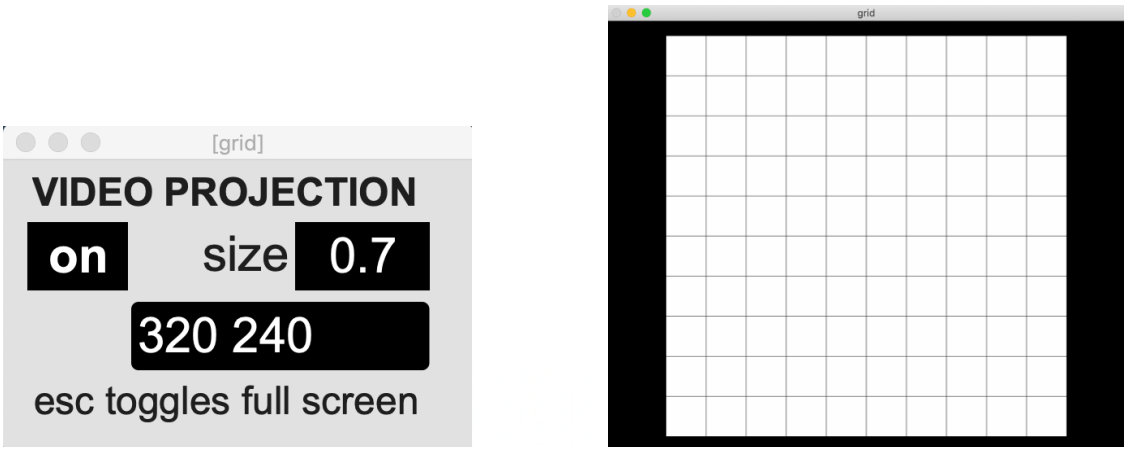


Figure 27: video grid control sub-panel and projected image

The **soundcheck** panel opens three dull but useful concert utilities:

- *1-2-3-4* plays a loop of my voice speaking a number through the corresponding output channel of the computer audio interface. Easier than intoning "test 1, test 1…" over and over.
- *Grid* sends an accurate checkerboard grid to the video projector. Settles arguments with technicians over where the fault in the aspect ratio error lies.
- *Slate* send a short pulse of white noise out the main audio outputs and trumpet speaker; useful for aligning camera mike with the internal multi-track files when synching audio to video in post-production.

Inventory of voices[18]

The voices in the software fall into three general categories: continuous sounds, pulsed sounds and sample playback. The first voices I programmed were designed to imitate hacked circuitry. Improvising with other trumpet players was a humbling experience (such cool noises, no computer needed!), but one that led directly to a new voices in (crude) imitation of my talented collaborators.[19]

- **Continuous sounds**. More-or-less continuous synthesized sounds, ranging from clean sine waves through chaotic functions to filtered noise, articulated by breath.

- *Sines*: 3x *cycle~* sine waves. Each *cycle~* phase modulated by *wobbler*. Multiplying mix by *~ (multiplication, i.e., ring modulation).
- *Triangs*: 3x *tri~* triangle waves. Each *tri~* has waveshape (ramp>>triangle>>saw) modulated by *wobbler*. Multiplying mix by *~.
- *74C14*: 3x *phasor~* sawtooth waves, clipped (>~) to produce pulse waves. Each clip threshold (i.e., pulse width) controlled by *wobbler*. Multiplying mix by *~. (This was the first voice I programmed, closest in sound to 74C14 waveforms of *Salvage* and *The Royal Touch*).
- *Trio*: 1 *sine*, 1 *pulsewave*, and 1 *chop* (clipped low frequency noise) generated as in the above voice sets, with same *wobbler* functions. Multiplying mix by *~.
- *FM*: 3x *cycle~* sine waves. Each *cycle~* phase modulates another *cycle~*, modulation range adjusted by *wobbler*. Multiplying mix by *~.
- *Chaos*: Chaos oscillator based on Chirikov's Standard Map, from Siska Ádám's "sadam" library (2015). Valve1 controls frequency, Valve2 = X0, Valve3 = P0. K (noisiness) given random value after all valves are up > 1 sec; *decide* object selected range of random between noisy and less noisy end of spectrum. No *wobble*.
- *3kos*: 3 Chaos oscillators as above, with cross-mixed valve control of fc, XO and PO in each oscillator. Linear mix (no multiplying).
- *Cicadas*: 3 whistly high *cycle~* sine waves with *wobbler*-controlled overdrive for ring-mod style difference tones. Linear mix.
- *Tix*: 3 low frequency, narrow pulse waves pinging resonant filters. Linear mix.
- *Noise*: 3x *rand~* low frequency noise. No *wobbler*. Multiplying mix by *~.
- *Air*: broadband white noise through 3 *reson~* bandpass filtered, linear mixing. Evokes Dörner/Ulher breath noises. No *wobbler*. Linear mix.
- *Fdbk*: Highly amplified microphone input passes through 3 *reson~* bandpass filters to output for filtered feedback. Clipped with 3 *pong~* objects. No *wobbler*. Linear mix.

- **Pulsed sounds**. Shorter synthesized sound bursts. Several use real-time *random* instead of *collective* to control some parameters.
  - *Zug*: 1 *click* pulse train per valve, *metro* controlled, tempo randomized each time valve is at top position. Each pulse train passes through *svf~* resonant filter, frequency and Q controlled from *collectives*. Linear mix.
  - *Kiss*: lip-smack and popping sounds. White noise burst triggered at top and bottom of valve position, 1 noise source per valve, through *svf~* filter. *Random* fc, Q, envelope and filter choice (low-pass, high-pass, bandpass, notch) with each noise burst. Linear mix.
  - *Bang*: similar to *kiss* but with valve extremes triggering *click~* through *gen*-based waveguide model. Linear mix.
  - *Rt60*: similar to *kiss* but with valve extremes triggering *noise~* burst through *gen*-based reverb model. Linear mix. Valve 1: *random* reverb time at extremes of valve movement. Valve 2: *random* other reverb parameters at extremes (size, early reflections, dampening, etc.) Valve 3: linear control of frequency shifting output signal. Mute Index finger: freeze valve functions. Mute Middle finger:

enable internal reverb feedback (modelled on an Ursa Major function used in first trombone system). Ring finger: enables infinite reverb time. Pinky finger: reset all parameters.

- **Sampled sounds**. Sound file playback (no live sampling).
    - *Samp*: 3-voice sample playback module, with pitch control from valve *collectives*. Each time the voice is selected, one of set of short prepared samples is randomly selected, which play back in a shuffle (*random*) mode. Pinky switch advances sample players to next sound file. Specific sample sets were chosen because they resemble extended trumpet technique:
        - *Fish*: 27 hydrophone recordings of fish and crustacea (from Smithsonian LPs from the 1960s).
        - *Birds*: 91 recordings of bird calls and bird songs.
        - *Radio*: 54 files of noisy radio signals, mostly shortwave, I recorded in East and West Europe and the USA in the 1980s.
        - *Crackle*: 38 recordings of surface noise and lock grooves from vinyl and shellac records.
        - *Coils*: 33 short recordings of electromagnetic feedback from my composition *Immortal Coil*.
    - *Tpt*: *Vst~* wrapper of an Aria sampler plays trumpet samples from the Garritan "Jazz and Big Band 2" instrument set. Implementation in imitation of Ben Neill's Mutantrumpet.
        - Valve1 assigned to pitch *collective*, with random velocity and random mute selection (open, harmon, straight, bucket, cup) on each change (valve1 movement quantized to 32 positions). The rapid change of mutes is especially evocative of Neill's Mutantrumpet.
        - Valve2 assigned to note duration, linear mapping from shortest duration when valve is up, longest when valve is down.
        - Valve3 mapped via *collective* to frequency of *cycle~* multiplying sample output for ring modulation effect, disabled when this valve is full up, for clean trumpet sound.
        - The pitch *collective* (valve1) stores values as an array of midi number for pedal note, overtone #, and resultant midi #, with specific pitch calculated at moment of retrieval; this aids *overblowing*. The number of partials in the *collective* set limits upper range for basic playback, default = 8; in overblow mode the default = 26 for more extreme high notes.
        - Overblow mode toggled with Pinky switch, jammed on when volume is locked on with valve switch.

TRIED AND REJECTED

The core hardware and software of the !trumpet were developed over the last four months of 2017, with the first performance (duo with Ben Neill) taking place in Beacon, NY on February 11, 2018. Subsequently I've added new voices, incorporated patches for several compositions in repertoire (see **compvoices** above) and utility purposes (**soundcheck**), and tweaked the core

program. Happily, my original software shell has proven quite robust in its ability to absorb this expansion – as I had hoped, but nonetheless much to my surprise, since I am not a very practical programmer. Several significant changes to the structure of the instrument and its software were implemented but subsequently rejected; likewise, several voices were programmed but dropped after trial by performance. The process of rejection honed the instrument's identity as much as any of the additions I accepted. An inventory of these false starts articulates what the instrument is not, which in turn helps clarify what it is.

Figure 28: Original Bluetooth mute, detail and in use in concert.

- Bluetooth mute. I was quite proud of my first wireless mute: essentially the same switch array currently in use, but connected to a hacked Bluetooth A/V remote control instead of an IR remote. Data was routed through the Macintosh Bluetooth handler (accessed through the main menu bar). I would have preferred an interface that mapped through my Max patch, but the final nudge to seek an alternate approach was seeing a photo of myself in action: the white controller made me look as though I was flashing the audience my middle finger. Frank Baldé (STEIM) encouraged me to use a simpler, one-way protocol, instead of the bi-directional handshaking of Bluetooth. I experimented with a few radio-frequency (RF) controllers, but they responded to key-closures too sluggish. I eventually settled on a compact programmable IR pc board, currently in use (Tauntek IRMimic) – essentially a TV remote control, about as dumb as you can get these days.
- Additional sensors and controllers were added to the trumpet, on the assumption that more data would be more useful: simple mechanical tilt switches, a 9DOF multi-axis tilt/direction sensor, a rotary shaft encoder (data wheel), additional pushbutton switches. Despite concerted effort, I was never able to incorporate the additional

information in a musically useful way. The various position sensors offer no resistance or haptic feedback, which is something I need in an instrument: I was always more comfortable with the old trombone slide -- a "sticky" controller that stayed where I put it – than with a free-air controller like a Theremin or Michel Waisvisz's "Hands"; and the spring-loaded valves (in a poorly maintained trumpet) had just the right amount of resistance and inertia. There also seems to be a limit to how many switch-to-function options my fingers and brain can map onto a single integrated instrument; more would require something like a mode swap – a complicating trait in the trombone system that I wanted to avoid repeating.

- Alternative valve sensors. The inverse-square fall-off of magnetic field strength reduces the resolution of the Hall-effect valve sensors as the piston approaches the top of its excursion, and requires some extra math to linearize. In pursuit of greater accuracy and simplicity I experimented with, and rejected, two alternatives:
    - Mechanical linear sensors. I coupled a pricey Bourne linear position sensor (3048L) to each valve. Resolution was excellent, but the drag of the sensors slowed valve movement, and the mechanisms cluttered the instrument and made it look too steampunk.
    - IR distance sensors. I experimented with a VCNL 4010 IR distance sensor, which measures short distances accurately, but quantizes the data to the nearest millimeter, which is too coarse for the 15mm of travel of the valve.

  (I am still on the lookout for other valve sensing options).
- Accompaniment software. Although the audio output of each voice consists of a mix of three sound generators, the !trumpet is an intrinsically monophonic instrument, like the traditional trumpet. For solo improvisation I wrote a program that generated an accompaniment by analyzing my valving and breath gestures and superimposing a delayed version of this control data on secondary set of voices that went to the PA. Perhaps due to its simplicity, my accompaniment algorithm didn't seem to add much value to solo improvisation. I eventually deleted this section of my software, focused instead on emphasizing the positive role of gaps and extended dynamic range in my solo improvisation, and sought live musicians for polyphony as needed.
- Sampling and DSP. Despite my frustration with the ubiquity of live sampling at the time of retiring the trombone-propelled electronics, and my vow to exclude it from the new instrument, from time-to-time I would – seemingly involuntarily – incorporate characteristics of that sound world in the development of new voices (like a recovering alcoholic veering toward the beckoning door of a pub). Aside from the sample playback patches used for the trumpet voice and the noisy sample sets, and the reverb and waveguide objects in a few percussive voices, I ultimately rejected most recognizably-"digital" effects.

THE FUTURE

This essay comes three years after I began the development of this instrument. The structure of both the hardware and the software facilitate periodic updates, and the present state of the instrument has evolved through cycles of addition and subtraction of features. At the moment

it seems relatively stable[20]. The most recent changes have been the addition of features to facilitate recording: a pressure-zone microphone embedded in the bell (see figure 25) and a "slate" function in the program. The pandemic has quashed performance options for the past nine months, so some solo recording sessions present a timely opportunity to evaluate the versatility and "instrument-ness" of the !trumpet.

FIGURE CREDITS:
All figures by Nicolas Collins except:
Fig. 1 right: Cindy Voitus
Fig. 3 left: André Hoekzema
Fig. 3 right, Fig. 4 right: Simon Lonergan
Fig. 4 left: Susan Tallman
Fig. 6 and 13: Zoe Adler (zoe@zoeadler.com)
Fig. 28 right: Robert Poss

---

[1] Initial draft May 2020. Update November 2020.

[2] An early (December 2017) solo performance of the !trumpet can be heard here: http://www.nicolascollins.com/music/!trumpet_Orpheus_12_2017.mp3. This is a live recording in the concert hall of the Orpheus Institute (Ghent) in which you can hear a few different voices and the acoustic transformations of the valving and mute movement. The PA is brought in 3'45" into the eight-minute performance, with notable rise in loudness and bass response. All reverberation is from the hall acoustics, none was added in post-production.

[3] http://www.nicolascollins.com/texts/allthisandbrains.pdf and http://www.nicolascollins.com/texts/peasouphistory.pdf.

[4] http://www.nicolascollins.com/texts/feetbackscore.pdf

[5] http://www.nicolascollins.com/texts/BackwardsElectricGuitar.pdf

[6] http://www.nicolascollins.com/texts/TrombonePropelledElectronicsReduced.pdf

[7] https://www.youtube.com/watch?v=89jbI0ZuaH4

[8] Nicolas Collins, *100 of the World's Most Beautiful Melodies*, Trace Elements CD, 1989. See http://www.nicolascollins.com/100melodiestracks.htm

[9] Email correspondence, 2008.

[10] In 1989 I developed a concertina with an embedded speaker and flex sensor on each of the six sides of the bellows. The goal was a "polyphonic" extension of the trombone. Despite several years of effort, assisted by the clever engineers at STEIM, I only managed to produce one piece for the instrument, performed twice. See Nicolas Collins. "Cargo Cult Instruments." *Contemporary Music Review* 6 (1991.)

[11] https://www.routledge.com/Handmade-Electronic-Music-The-Art-of-Hardware-Hacking-3rd-Edition/Collins/p/book/9780367210106

[12] *Salvage (Guiyu Blues)* (2008), https://www.youtube.com/watch?v=XV50-Cwy1RI&t=504s

[13] *The Royal Touch* (2014), https://www.youtube.com/watch?v=DtGcueEsuDE

[14] https://mcachicago.org/Calendar/2017/02/Music-For-Merce

[15] Private conversation, New York, NY, late-1980s.

[16] "Semiconducting – Making Music After The Transistor." *Musical Listening in the Age of Mechanical Reproduction*. Gianmario Borio, editor. Ashgate (London) 2015. A later revision can be accessed here: https://nmbx.newmusicusa.org/what-to-ware-a-guide-to-todays-technological-wardrobe/

[17] In 2018 I gave a performance with the !trumpet at an event at CERN in Switzerland. A small group of particle physicists came up afterwards and asked me about the instrument and how the sounds were formed. When I mentioned that each voice was the result of three oscillators mixing by multiplication, one of the scientists turned to the others and said, "Didn't I tell you he must be using multiplication?!" The sound of math.

[18] As of 11/2020

[19] It is worth noting that in the five decades I have been pursuing what many would describe as "electronic music" I have seldom had any interest in, or talent for synthesis. My work was usually based on found sound material – from feedback, to live sampling, to improvisers' decisions, to unstable circuitry. Programming the voices for the !trumpet was my first serious attempt at synthesis since my early years in the electronic studio at Wesleyan (1972-74).

[20] Liz Allbee, on trying out my instrument, observed that the breath sensor only responded to blowing: "I like to make sounds by sucking through the mouthpiece as well as blowing." I purchased a variant of my current breath sensor that measures negative pressure (vacuum), as well as the usual positive pressure, but at the time of writing Covid-19 holds these components on the opposite side of the Atlantic from me – experiments delayed.