MUSICAL™

The Musical
TapeMeasure - TroMbone – ThereMin

By

Becky Currano
Peter Solderitsch
Unnur Gretarsdottir

Introduction – Motivations

Musical instruments with continuous pitch manipulation capabilities are fascinating in the breadth of musical tones that they can produce. Traditionally, however, their interfaces present significant limitations and disadvantages. They tend to exclude the use of chords, limiting the performer to one note at a time, and they often give insufficient, if any, user feedback (other than the sound itself), making it very difficult for a novice user to accurately command the musical output.

This problem motivated the creation of Musical™, a new computer instrument that combines the continuous controllability of the trombone and the theremin with a unique and programmable interface for enhanced user control.

Model and Metaphor

Musical™ was inspired by the pull and snap-back action of tape measures. The tape measure provides a new model for continuous pitch control. This model offers the possibility of visually 'measuring' musical pitch on a one-dimensional scale, through markings or colors. It also presents the potential for developing new representations of musical compositions in commonly understood units such as inches or millimeters.

Musical™ can be operated in either continuous mode or discrete mode. In continuous mode, the user can sweep smoothly across a pitch range by pulling the tape measure. The measure represents an unbroken pitch continuum, from which the user can produce a virtually infinite number of notes. But the user still faces a challenge in precisely locating individual notes. In discrete mode, this difficulty is removed, as chromatic steps are mapped onto existing markings on the tape measure. In this mode, pulling the tape out (while pressing button(s)) will vary the pitch of the note(s) being played discretely, and within a scale. (In the current pd patches, one chromatic "half-step" is mapped to 1/2 inch on the slider).

In discrete mode, Musical™ resembles a harmonica in some respects. Sliding a harmonica left or right when blowing through it will produce a pre-selected scale of notes whose pattern depends on the type of harmonica used. The PD patches in Musical™ allow the user to program a variety of scales into the sliding motion of the instrument. Currently, Musical™ includes in a major scale setting, a harmonic minor scale setting, and a "blues-like" scale setting. The user, however, can create new scales by programming any sequence of notes that can be represented in tabular form in PD, even if the sequence does not monotonically increase in pitch.

In addition, the intervals between buttons are mapped to *scale note* intervals rather than absolute pitch intervals. This makes it very easy to play chords. For example, playing a particular triad on the buttons while sliding up a major scale, for example, would produce the canonical music theory chord sequence I, ii, iii, IV, V, vi, vii.

The choice of mapping pitch increase to pulling the tape measure and mapping pitch decrease to retracting the tape measure was intuitively driven by the familiar layout of a piano keyboard, where the notes increase in pitch from left to right. (Musical™ is designed to be held in the left hand, so the tape pulls out to the right). This is in contrast to the trombone and theremin, where extending the hand maps to a decrease in pitch while retracting motion maps to increase in pitch. Deciding which mapping is more appropriate is not obvious, and could be debated extensively before a definitive conclusion is reached.

Thinking of the motion driving pitch control in terms of stretch opens up the notion of note distortion, as a spring is distorted when pulled. The hardware of a tape measure incorporates a type of torsional spring, called a constant force spring, which exerts a roughly unchanging force over a large angle of deformation. However, the spring does distort and the force does increase somewhat, and the analogy can be applied, so that pulling the tape or string can represent stretching the note.

This analogy brings its own bearing on the debate of pitch mapping. Considering the length of the tape alone implies that the longer it is pulled, the lower it's natural frequency, and thus the lower the pitch it represents. However, considering that stretching a spring increases its stiffness implies that the longer it is stretched the higher the natural frequency. To be truly scientific about it, one would have to examine the equation for natural frequency of a 2-node string. However, it may just come down to an intuitive 'feeling'. If the spring force is strong enough, the illusion of stretching the stiffness may more strongly align with the user's intuition than the predominantly visually-influenced judgment based on length.

Mapping and Controls

Devising a control scheme presented many challenges in the creation of Musical™. Musical™ was designed with the intention of outfitting the interface with many degrees of freedom, to give the user the sense of "shaping and moving" the music with his or her hands. This required sensors, which could measure different types and subtleties of motion. In addition, the design sought to map motion in a meaningful way, to represent the desired changes in Musical™ quality.

Pitch

The team chose from the start to integrate pitch control into the main mechanism of Musical™, the tape measure. A rotational potentiometer was chosen as an ideal means of measuring the rotary motion of a coiling and uncoiling tape measure. A 10-turn pot offers a broad sensing range equal to 20 Pi * D, where D is the diameter of the pulley around which the tape measure turns. The rotary motion of the potentiometer also provides a simple and elegant mapping to pitch variation.

Many tape measures have one end of the spring fixed to the stationary housing, while the tape pulls the other end of the spring, winding or unwinding it around the fixed end. This type of tape measure does not lend itself easily to coupling with a potentiometer. We found a more suitable type for this application, with a rotating housing in which the tape and spring are coiled. We glued one end of the potentiometer to the rotating housing and fixed the other end to the circuit board, so that as the housing rotated, it varied the potentiometer resistance. The resulting effect is that the potentiometer turns 1 rotation for every length Pi*D that the tape measure is pulled, where D is the diameter of the coiled tape inside the housing.

A programmable motor could provide an alternative means to map rotary motion. This method would allow for haptic feedback to be integrated electronically in various ways and to varying degrees. For example, electronic detents could be added to alert the user to note locations along the length of the tape's travel. Alternatively, variable resistance could be added and controlled more precisely for different lengths, and a program could be implemented to cause the motor to automatically rewind the tape when the user stops pulling. The motor method, while an attractive alternative, was not chosen because of the time constraints of this project.

The team also initially considered replacing the rotary control with a retractable telescoping device such as a pointer. This idea seemed attractive because a pointer can withstand force in bending, and could then be coupled to a force sensor that would measure the angle of the pointer for stereo panning. However, the team decided against this idea mainly because extending and retracting a telescoping device involves a series of discrete motions. The more continuous motion of a tape measure seemed more appropriate for continuous sound control.

Stereo Panning

Initially, we intended to implement an altered version of the tape measure – using a string instead of a tape. The string, being flexible in all directions, would give the user more freedom of expression while opening up avenues for new dimensions of Musical^{TM} control. By varying the string angle in a clockwise direction, the user could change the balance from left to right, and vice versa in the counterclockwise direction. Sensing the direction and degree of the string angle could present a spatially intuitive mapping for stereo panning, or even surround sound, with the right speaker hardware.

Incorporating the string in the hardware construction, however, proved more difficult than expected. The pulley had to be axially offset from the spring, which put a cocking moment on the shaft when the string was pulled. The moment caused a high friction in the system, resisting turning of the shaft. This friction, exacerbated by the imperfections in the materials and manufacturing of some of the parts used, caused the string to stick at various points when retracting.

Because the emphasis on the continuous capability of the instrument made it important to achieve smooth pulling and retraction of the string, we eventually decided to incorporate

an entire smooth-acting tape measure directly into the instrument. We also decided not to pursue the stereo effects due to time constraints.

Chords/Individual Notes

Musical™ incorporates eight buttons on the main body, which can be used individually or collectively to activate the music. The first button is assigned the root pitch, determined by the sliding tape, and subsequent buttons are configured to be offset a distance from the root.

Each button is comprised of a 0.2" FSR covered with a foam-rubber pad. Taking advantage of the continuous nature of the FSR sensors and mapping resistance variation to volume gives the buttons an added dimension. Through the buttons, the user can independently and simultaneously control the volume of different notes. The foam provides a passive form of haptic feedback, which allows the user to relate volume to deformation of the foam.

These features give the performer the ability to play different combinations of tones, either in sequences or in chords, at any position along the slider. It also provides another compelling advantage over other continuous-pitch-control devices: the ability to slide multiple notes at once, and actually *change* those notes *during* the slide.

Timbre

The team also liked idea of incorporating an amorphous "squeeze" motion into the system. This provided a favorable means of adjusting the timbre, which can be viewed as a sort of organic sound modulation. This was accomplished by outfitting the end of the tape with a foam ball. The foam ball was cut in half and then glued back together with an FSR mounted between the two halves, and attached to the end of the tape measure. When the user squeezes the foam the FSR sees a varying force, which is communicated to the microcontroller as a varying voltage, and then to the computer as a varying data signal. PD then uses FM synthesis to modify the timbre of the music accordingly. Squeezing the ball increases the modulation index of the synthesized voices, achieving a "wah"-like effect (inspired by wah-pedals commonly used with electric guitars).

We originally hoped to use velocity and acceleration of the slider or button presses to affect the timbre of sound based on intensity of each. However, limited knowledge of PD and time constraints precluded the possibility of pursuing these paths.

Portability

An important element guiding the design of Musical™ from the beginning was portability. We wanted this instrument to be a mobile means of creative entertainment, that could be played in virtually any setting – at school, on the bus or subway, walking in the park… It had to be small and non-cumbersome. The design of Musical™ assumes a somewhat steep learning curve as a natural consequence of the continuous control

capabilities. However, we hope that the combination of portability and a fun interface will enable and encourage frequent practice, wherever and whenever the user has time.

Computer Interface

Musical™ interfaces to a Linux machine through an ATMEL AVR ATMega163 microcontroller, which contains an on-board 8-channel, 10-bit A/D converter, programmable serial UART, and 4 8-bit programmable data ports.

Interfacing 10 continuous sensors (8 FSR buttons, 1 FSR squeeze modulator, and 1 potentiometer) to the AVR board, which only supports 8 analog inputs, required multiplexing multiple inputs to a single pin. Musical™ includes two 4-to-1 multiplexors, both on the same chip, for the button inputs. This enables all 10 of the analog sensors to be routed through only 4 of the AVR's A/D channels.

PD buffer overflow problems necessitated the reduction of MIDI messages sent by the C code. Where originally, the code logged the status of each button continuously, the revised version continually reads their status, but (keeping track of the last value that was sent on each MIDI channel) only *sends* values that have just changed. Since normally only a few buttons are changing at a time, this significantly reduces the communications load on the system. Instead of sending 10 Midi notes per cycle (many redundantly), the system only sends a few (the ones that are actively changing).

To send data about the position of the tape measure (that is, the 'a-to-d' value of the 10-turn pot), we wanted to be sure to capture all 10 bits from the a-to-d converter. Since each midi note is only 7 bits long, we packed the 10 bits into a single midi message, by using the "note" value of the message to send the 7 most-significant bits, and appropriating the "velocity" value of the midi message to send the last 3 least-significant bits. When this information is received in pd, it gets re-packed into a single number that ranges from 0 to 127 that has 3 bits of sub-integer precision (see "inputs.pd" file).

Appendix

Part 1

PD Notes

The following is a file-by-file description of the main PD patches that have been provided to interface with the Musical™. PD screenshots are included after the notes.

"inputs.pd":

This is the file that receives all of the midi messages that the Musical™ sends. Any future PD patches created to support Musical™ should to use this to grab all the inputs. It contains the following:

1. An arrangement of 8 button inputs that map to the 8 buttons on Musical™. They are mapped to their physical position and their signals are processed and sent out as "wireless" PD messages. (This means that the button input that comes in on midi channel 1 isn't necessarily the top button under the index finger on the device. For example, in the current setup, the button that sends data over midi channel 1 is actually the "pinkie" button on the top row of Musical™. Because there is often some low-level noise from the FSR's, we have incorporated a hard cutoff on the buttons called "b_cutoff" that can be adjusted to eliminate this noise and prevent unintended signal activation.

2. A section that receives "string" (or "tape measure") position. This is done by packing 7 bits into the "midi note", and the remaining 3 bits into the "velocity", to retrieve the full 10-bit-precision number. The PD code then passes that input through a simple low-pass filter (which is reconfigurable) to smooth out the signal a bit.

3. A section that receives data on midi channel 15 that represents the "squeeze" FSR value and the "panning" value (which is currently unsupported on the actual Musical™ device, but was an original planned feature).

"proj_discrete.pd":

This subpatch combines everything needed to operate the Musical™ in "discrete scale" mode. The 2 scale arrays are set using the "pd majscale" subpatch. The scale note offsets are set for each button by changing the int values that go into the right-most inlet of "button_discrete2pd" and banging the "s init_tones" message to initialize the notes. The user can change the root *key* of the current scale type by adjusting the "string_offset" parameter on the left side of the patch. In order to get proper sound out, the user must do all these things, then open the "inputs" subpatch by clicking on it, adjust the "pan" parameter in the lower right to some value ("50" will give you even balance between left and right channels), then activate the metro object component of the low-pass filter (and if desired set the level of filtering as appropriate).

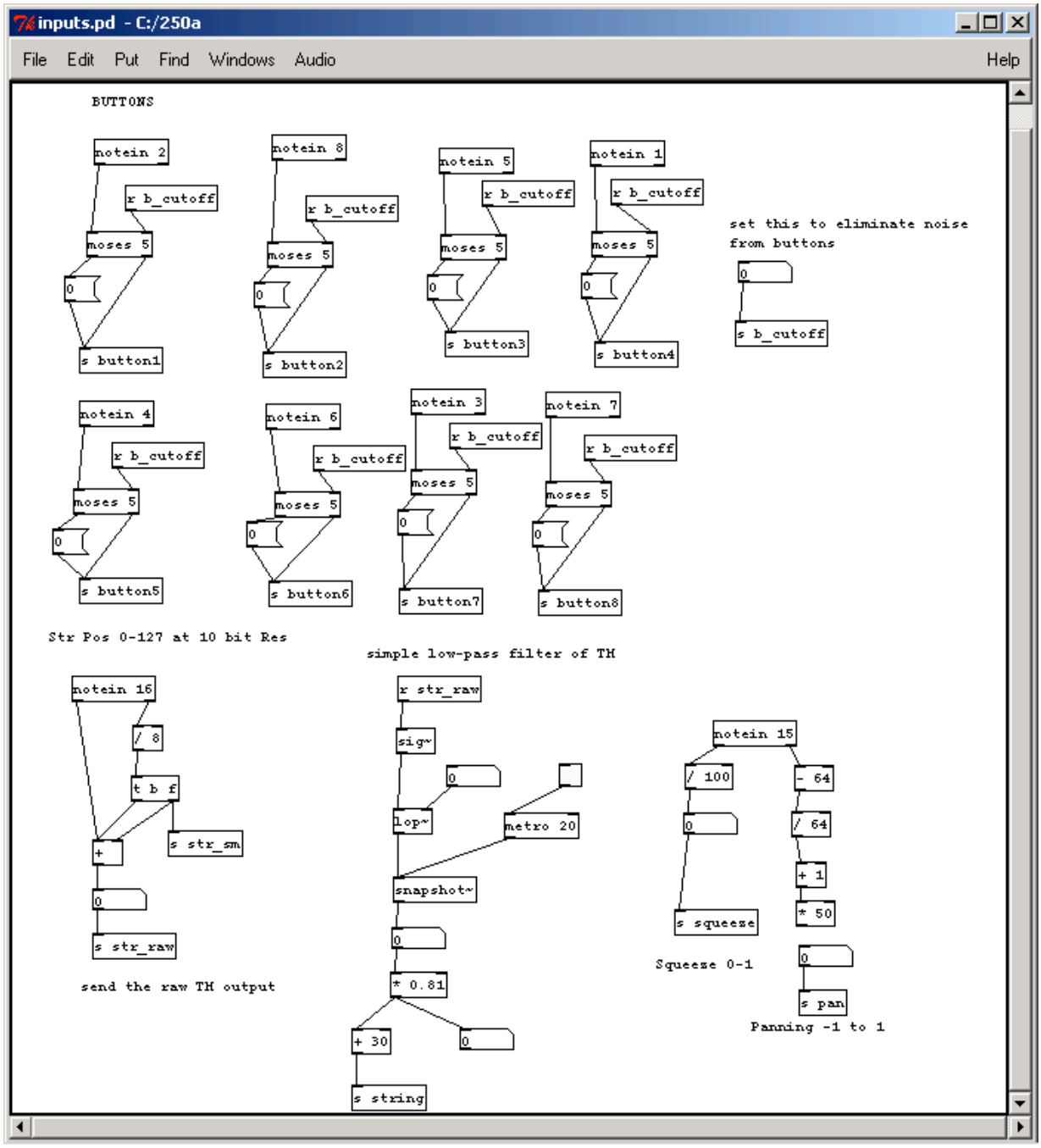
A few points on the way the scale lookup tables work: the "array_maj" table has x values from 0 to 13, which represent numbers in the scale (note that there are really only 7 notes

in each of the current example scales, but the table has been extended to 13 spots to allow the user to offset up to 7 places away from a root string that's already *at* the 7th note (index = 6) of the scale. The y-value for each X is the actual chromatic position away from the root note at that scale index. The "array_lk" table does the reverse of this. Each "x" value is a chromatic note. If that chromatic note is not actually a *member* of the current scale, its y-value will be "-1". If it *is* a member, its y-value will be equivalent to *which* note in the scale it represents (in a major scale, the 5th half-step away from the root of the scale is actually the fourth note of the scale (or 3 scale notes away from the root). button_discrete2pd.pd uses these lookup tables to translate the tape measure position into the correct scale notes.

"button_discrete2pd.pd":
This is a subpatch that produces a single, FM-synthesized "voice". It takes as inputs a "volume" parameter (a "button" value), a "root pitch" parameter (the "string" value), and a "note offset" parameter, which for 'discrete' mode represents the number of steps *in the scale* away from the root pitch you want the voice to be. It takes in the root string pitch, uses the "lookup_note" subpatch to translate that root pitch into a position in the currently-set scale, then computes the offset interval from that root scale note. It feeds all of that information into "fm_voice.pd" subpatch, which actually generates a tone using simple fm synthesis.

"proj_continuous.pd":
This subpatch brings together everything needed to operate Musical™ in "continuous" mode, and is very similar in structure (though simpler) to the "proj_discrete.pd" patch for operating in "discrete" mode. It has 8 button inputs configured that feed into the dac.

"button_fm.pd":
Inlets for a volume control (button pressure), root pitch, and a "pitch offset" value, and outputs a fm-synthesized voice representing that root pitch + offset.


inputs.pd - C:/250a
File Edit Put Find Windows Audio Help
BUTTONS
notein 2
r b_cutoff
moses 5
0
s button1
notein 8
r b_cutoff
moses 5
0
s button2
notein 5
r b_cutoff
moses 5
0
s button3
notein 1
r b_cutoff
moses 5
0
s button4
set this to eliminate noise from buttons
0
s b_cutoff
notein 4
r b_cutoff
moses 5
0
s button5
notein 6
r b_cutoff
moses 5
0
s button6
notein 3
r b_cutoff
moses 5
0
s button7
notein 7
r b_cutoff
moses 5
0
s button8
Str Pos 0-127 at 10 bit Res
simple low-pass filter of TH
notein 16
/ 8
t b f
+
s str_sm
0
s str_raw
send the raw TH output
r str_raw
sig~
0
lop~
metro 20
snapshot~
0
* 0.81
+ 30
0
s string
notein 15
/ 100
0
s squeeze
Squeeze 0-1
- 64
/ 64
+ 1
* 50
0
s pan
Panning -1 to 1