Augmenting the guitar as an MPE controller with capacitive touch sensors

Jamie R. Pond
School of Electronic Engineering and Computer Science
Queen Mary, University of London
London, United Kingdom
ec20768@qmul.ac.uk

***Abstract*—We explore the development of a prototype MIDI Polyphonic Expression (MPE) enabled, guitar-like instrument for use in the modern music production studio. Following a process of human-centred design and human-computer interaction, we focus on creating a prototype that is conducive to fostering creative flow. This is achieved through the implementation of capacitive touch sensors (Bela Trill Bar) which are affixed to the frets of the prototype, which offers affordances of polyphonic aftertouch and pitch bend. The data thereof is interpreted with a given mathematical and programming framework. We find that the prototype is proficient at supporting novel creative musical ideas and subjectively pleasingly idiosyncratic musical performances, however this is somewhat impeded by practical prototyping choices.**

A brief demonstration of the working prototype is *available here*.

***Index Terms*—Augmented guitar, digital instruments, MIDI, NIME, HCI**

I. Introduction

The NIME community (New Interfaces for Musical Expression) is rich, diverse, and ever expanding. New technologies are continually developed, allowing creatives to interact with machines to create sonic artifacts and experiences that would not have been possible otherwise (Wanderley et al. 2001).

This paper aims to explore the ways in which the expressive capacity of the guitar can be extended by employing new techniques and technologies to this familiar and well-loved instrument.

This will be achieved by the development of an experimental prototype MIDI controller which will incorporate new MPE (MIDI Polyphonic Expression) techniques which allows users to create more expressive performances by allowing pitch and volume manipulations on a per-note basis. This is of course supported on existing traditional guitars, but is not found on contemporary guitar-like MIDI controllers such as the Yamaha EZ-EG (Yamaha 2003). Something which has surprisingly not been supported as standard in MIDI since its inception in 1982, and has recently seen an uptick in adoption and implementation.

The prototype achieves its goals by implementing multi-touch capacitive sensors (Bela Trill Bar) across the frets of the guitar, which afford data on touch position and touch size, which can be combined to determine appropriate note-on and off events, to emulate the playing of a guitar. However, this can also be used to create expressive aftertouch and pitch bend data, which is central to the thesis of this study.

This paper follows three themes which can be summarised by the following questions:

1) What are interesting ways to extend the expressive vocabulary of the guitar as a MIDI controller?
2) How can we create a digital MIDI instrument that guitarists will enjoy using and be conducive to creative flow?

Creative flow is defined as "*the mental state in which a person performing some activity is fully immersed in a feeling of energized focus*" (Csikszentmihalyi 1990). MacDonald et al. (2006) highlight how important flow is to productive and creative musical output. Enabling creatives with digital instruments that can interface with modern tools such as Digital Audio Workstations and modern synthesizers with an interface that is conducive to a state of creative flow, is the key motivation of this paper. *Creatives*, in this context, is defined as any person using (digital music technology) to create novel creative output, such as songs or audio artefacts.

Some studies indicate that there is a feeling among creatives that interaction with digital audio units or MIDI instruments is not conducive to creative flow (Martelloni et al. 2020). This perception is a key challenge to this paper. The author proposes that this is not an intrinsic property of digital instruments, but instead is product of the relative novelty of these digital technologies, whose designs have not been iterated and smoothed-out over generations of musicians. The traditional guitar, over it's near half a millennium of existence (Tyler 1975), has had the chance to have its design iterated and improved by great numbers of people. The MIDI guitar has had mere decades, so it is no surprise that (currently) the design does not compare to its traditional counterpart. This study attempts to take one more step in the iterative process of the collective goal of designing the digital MIDI guitar.

II. Literature Review

A. The Augmented Guitar

The NIME community has had a rich history of augmenting the guitar, among nearly all other instruments, to add new features. Recently, Avila et al. (2019), published a very comprehensive overview/framework for examining and conceiving new augmented guitar instruments.

They observe that guitar augmentations are generally vary across two dimensions, namely the degrees of *invasiveness* and degrees of *transformation*.

For example, Martelloni et al. (2020) experiment with a guitar augmentation that involved wrapping finger-style guitarist's acoustic guitars in bubble wrap, to examine how the new constraint had an impact on their creativity, and playing style. This would be an example of a *low* invasiveness augmentation, since it can be easily done, and easily undone, with no permanent modification to the guitar. It arguably also has a low degree of transformation, since most of the essential properties and characteristics of the guitar remain the same.

Another example is the GuitarAMI system which is put forth by Meneses et al. (2018), which involves adding sensors and pickups to a classical, nylon string guitar, which allow for extended gestures and sensors (tilt, accelerometer, cameras) to map to additional audio effects, for example reverb length/mix and sound reversal. This was controlled and processed by a digital foot-switch. This offers enormous scope for changing the sound of the instrument, but is non-permanent. Therefore, this augmentation has low invasiveness, but a high degree of transformation.

An example of a guitar augmentation with high invasiveness might be that of adding a kill-switch to an electric guitar (a simple button connected to the guitar's inner circuit that disables the sound for the duration of the button press). This involves drilling a hole in the body of the guitar (a permanent change) and also soldering additional components and wires into the guitar's existing circuit, which is not impossible, but difficult to reverse.

The proposed prototype falls into the extreme of both dimensions. As will be discussed in later sections this prototype falls into the extreme end of invasiveness with permanent drilling alterations into the neck of the guitar, but also with the removal of strings and replacement thereof with sensors, the degree of transformation is arguably quite high.

Furthermore there have been more 'smart' and network connected augmentations. Turchet & Barthet (2019) have focused on creating a smart guitar for collaborative musical practice. This is immensely interesting since it offers the opportunity for a guitarist and a smartphone user to practice music together using a smart guitar as a hub for sound reproduction.

Additionally, Khalil (2019) has developed a system for detecting note onsets and pitches using Music Information Retrieval (MIR) techniques which feeds an internet connected sampler, which will fetch audio samples based on the content of voice commands. For example, a user might speak "piano" into the smart guitar's in-built microphone and the on-board Bela will contact the **FreeSound** API, and download an appropriate sample, and play it through the built-in sampler, in response to the user's guitar playing.

Another approach was to implement deep learning in the augmentation of the guitar which is incredibly interesting. John (2020) created a deep learning vision system to personalise the experience of the electric guitar, such that the computer vision system could recognise different individuals who were holding the guitar and swap effects chains and parameter values to match their preferences.

B. The guitar as an expressive MIDI Controller

One might argue that the key theme in the rise of the guitar in modern culture, and perhaps the key to its success, has been innovation. Since 1932 (Millard 2004) when the first electronically amplified guitar was introduced, practitioners and technologists have developed a plethora of technologies to electronically augment the capabilities of the guitar (Lähdeoja 2008). However since the dawn of MIDI in 1982 (Stubbs 2018) the keyboard has become the standard for inputting MIDI information in modern studios, bedroom and professional alike (Roads 2015). This is a less-than-ideal situation, because this, to a certain extent, excludes certain types of musicians from being able to express their musical ideas in a way which is natural and intuitive for them, when composing computer music. This is problematic because there are approximately 7 million musicians in the United Kingdom[1], and 4 million of whom are guitarists (BBC 2006). This means that there are up to 2 million people in the UK alone who are not supported by the traditional paradigm of keyboard MIDI controllers. This kind of discrepancy could be resolved by offering guitarists a viable alternative to using a keyboard to play in their musical ideas to the DAW. We further wish to prevent users from requiring to use a mouse to input musical data, since Nash (2012) highlights how this practice can be highly antithetical to creative flow in the studio.

The current literature appears to be focused on the live performance of music. This presents an opportunity to focus on the *studio* applications of the augmented guitar. MIDI keyboards have been popular in professional and home studios for many years, and with the rise of advanced, expressive MPE products like the ROLI Seaboard (Lamb 2014), guitarists have been left largely without an expressive controller which they can use in the studio to take advantage of the rich expressive world of MPE instruments.

However, there have been a wide variety of very interesting and successful MIDI guitars in previous years. For example, the SynthAxe (White 1984) (see Figure 1) is an incredibly comprehensive MIDI controller, offering strumming capabilities as well as buttons on the body which correspond to note activations.

Though the product was not a commercial success, possibly as a result of its cost-prohibitive $11,000 price (Metlay 1990), there are several elements that contribute to this being a very expressive MIDI controller. The variety of supported playing styles is very wide, and therefore supports the creativity of users in as many ways as possible. The SynthAxe used 'fret wiring' to detect the location of fingers on strings. For example, pressing the low E string on fret 1 would make a distinct circuit to the D string on fret 12, using this, gestures can be mapped into MIDI notes. It was also possible to bend the

[1]Based on there being 60 million people (DataCommons 2020) in the United Kingdom in 2006 (year of the BBC article) and 11.6% of people in that year participating in playing a musical instrument.

strings on the SynthAxe, which adds great expressive potential and leverages existing user expertise. However, Metlay (1990) notes that despite its similarity to the traditional guitar, it "required a fair amount of retraining to use properly".

Furthermore, Roland Corporation have been offering commercially successful synthesizer augmentation for guitars for many years. For example the Roland GR-20 (Roland 2021). This involves a attaching a removable 'divided pickup' (one sensor per string) to user's existing instruments, which then connects to a digital computer nested inside a stomp-box style interface. This again is an example of a low-invasiveness, high-transformation type augmentation. This allows users to use all their existing expert technique in controlling the new MIDI instrument, which is a highly desirable feature, and one which this prototype aims to achieve.

Fig. 1. The *SynthAxe*

The Yamaha EZ-EG (Yamaha 2003) offers MIDI to the guitarist through the use of buttons. This is a good first step, but cannot offer aftertouch or pitch bends to the user, which limits its sonic capabilities. The Artihphon Instrument 1, is also making great steps in the right direction also incorporating elements of polyphonic aftertouch to the design, but cannot perform pitch bends on the neck of the guitar in the intuitive gesture that so many guitarists know and love. This project attempts to create a prototype that is not only capable of MPE pressure, but also of pitch bending.

C. Supporting Creativity and Flow in Studio Applications

This study aims to develop the capacity of the augmented guitar to interact with digital instruments and DAWs in a musically pleasing and spontaneous way. As Martelloni et al. (2020) highlight, musicians can be wary of augmenting their performance with digital technologies, for fear of coming up short with respect to spontaneity and creativity.

This insight could either be framed in terms of how digital instruments are somehow inherently un-spontaneous and not conducive to supporting creativity; however, it could also be viewed as a design problem. This echoes the perspective of Norman (2013), where the burden of creating a compelling digital product is well within reach, but the burden to do so falls on the shoulders of the designer. Thus, the aim of the design is two-fold: to reduce the barriers to creativity that are experienced when interacting with technology, but also to further develop new and creatively useful "contact points", developing on the research of Lähdeoja (2008). Contact points are defined by Lähdeoja (2008) as a mapping of a physical gesture/object interaction with sound.

The way that we create music is becoming more digital and 'in the box' (Nash 2012). This means that the amount that musicians are interacting with their computers in a musical context is increasing, and therefore these perceived "limitations" of digital technology need to be reduced. Since the technological capabilities of the computer are great, we might expect that it should also afford great creative opportunities. It would seem, however, that the limitation is in the *design* of these new musical interfaces, not their technological prowess.

III. Design

A. Prototype overview

Instead of having physical strings strung between the head and the bridge of the guitar, we employ Bela Trill Bar sensors. These will be positioned across the frets which will provide pressure-sensitive multi-touch to the guitar. A basic outline of the prototype can be seen in Figure 4. In this prototype, the strings have been completely removed, and we will only employ the sensor input. This is an experimental design, but is intended to work as a forcing function (Norman 2013) to encourage exploration of the novel MPE features. Furthermore, strings may impact on the accuracy of the sensor data.

This design is akin to the Chapman Stick k(Enterprises 2021), and encourages two hands on the frets as as shown in Figure 2.

The prototype contains only six sensors in total, as any more is prohibitively expensive. The sensors occupy the 4th to 9th frets, as these are the frets which could accommodate the whole height of the Trill sensor. This means that on the frets which are occupied, all of the fret area is available to the user.

The Bela Trill Bar sensors have a Grove connector on the back, so a small hole has been drilled through to neck to allow the wire to be passed through to the other side, so each sensor can be connected in series. Each of the sensors have also been trimmed to size to fit the appropriate width and height of their respective frets.

Fig. 2. Chapman Stick style guitar, with a two-handed playing technique on the fretboard.

With processing of the sensor data, this allows the array of sensors to act as if they were a matrix of fret-string locations, and thus can begin to emulate the behaviour of a traditional guitar.

Implementing the touch size and touch location data associated with each touch point will provide the data required to implement the MPE aftertouch and pitch bend functionality.

The Bela Trill Bar sensors (Figure 3) have been selected to play the central tole in this design for several reasons. Firstly, it affords multi-touch sensing, up to five simultaneous touches. which is sufficient for most guitar playing.

It offers sensing in the horizontal direction only, but this is sufficient since notes are only differentiated horizontally on the guitar, and pitch bends (on traditional guitars) are also only available on this axis.

Secondly, it affords detection of touch size, which can be used as a proxy for pressure. Touch size and pressure are not the same, but are highly correlated. Therefore, we can use touch size in place of true pressure, and the prototype should offer a very similar behaviour compared to FSRs. In a more sophisticated prototype, we might consider incorporating additional sensors that can detect true pressure i.e. force sensitive resistors (FSRs). This would be beneficial as this would afford a more natural mapping between the gesture and the sound (Norman 2013). This is because most acoustic instruments react to *force* and not 'touch size'. Therefore, a force-based interaction would represent an experience which is more representative of users experience with acoustic instruments which may be more successful overall. This is important as it links to Norman (2013)'s 6th fundamental principal of design. Mapping in design, just like in mathematics, represents a connection between one set to another set. In this instance, the mapping of force exerted/touch size on the sensor, to the timbre (e.g. brightness) of the sound.

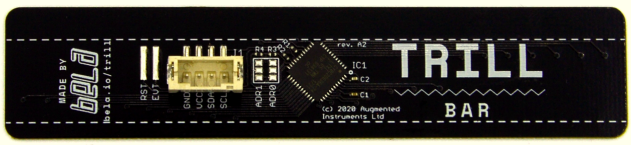


Fig. 3. Bela Trill Bar sensor.

In the design of this prototype, FSRs were also examined to be the principle sensor to comprise this design. As stated above, these do come with the added benefit of *true* force sensing, not just using touch size as a proxy, which may result in a more expressive final product, but they also come with more drawbacks.

However, this would only support pressure detection, and not pitch bending, since we would not be able to detect the continuous horizontal variance across the width of the frets which is essential for pitch bending.

Furthermore, this would require a very large amount of analogue inputs into the micro-controller (one per fret-string location), which would rapidly be exhausted, as the number of frets supported increased.

In future iterations of the design this could offer additional information on top of the capacitive sensors, which could make the instrument more expressive.

IV. Software Implementation

A. Sensor to MIDI Translation

We can summarise the logical overview of sensor input to MIDI instructions with the diagram shown in Figure 6.

1) Sensor discretisation & touch identification: Each fret of the system is represented by a Bela Trill Bar sensor, which has continuous expression in the horizontal axis.

We are then faced with the challenge of discretizing this continuous dimension into 'string zones', such that, if there are 6 strings (which is the case for the traditional guitar), all touches within the first sixth of the bar are categorized as string 0, all touches within the second sixth of the bar are categorized as string 1, and so on.

Trill sensor data is already normalized to the range of 0 to 1, which means that the far 'left' of the sensor represents a mapping of X to 0, and the far right represents where X maps to 1.

Using this knowledge and knowing how many strings we want to use in the system we can determine the width and boundary locations of each string zone. Expression 1 describes the set of string boundaries $\mathbf{B}$, which has length L and where S is the number of strings.

$$\mathbf{B}_n = \frac{n}{L}, \qquad n \in [0 : S-1], \quad n \in \mathbb{Z} \tag{1}$$

Fig. 4. Basic prototype design

It is recognised that $S+1$ boundaries are typically required to delineate a region completely, however, since we are bound to not receive input over 1.0, this 'implies' our final upper bound. We can use this fact to reduce the complexity of our program.

Having discretized all touches into string zones, and knowing their fret index, we can construct a binary matrix that represents touches on each fret-string location. This is required to determine note onset and offsets, which does not directly relate to the MPE behaviour of the system, but is a necessary pre-requisite for all MIDI instruments.

We can determine the string zone of the red dot with the following logic, shown is pseudo-code in Figure 8.

Which will return $\mathbf{B}_n = 2$ for $S = 6$.

2) Populate touch matrix from activations of sensors: Since we know the index of each sensor, we can construct a strings $\times$ frets matrix like so. For example with three sensors, as is shown in Figure 9, we can generate the binary matrix in Figure 10.

Here, we will also define the notation for discussing each individual cells of the matrix diagrams. In Figure 10, the $\mathbf{1}$ in the top row would have position $(0, 3)$, the middle row $(1, 0)$ and the bottom row $(2, 5)$. Fret index increases with vertical movement down, and string index increases with horizontal movement to the right.

3) Determine note onsets & offsets: Note-on and note-off events are not trivial to detect as they can only be determined *between* audio callbacks, not *within*. Meaning, that no single 'snapshot' of a touch matrix is sufficient to determine note onsets or offsets. This means that there must be a system for 'remembering' which fret-string locations were active in the previous audio callbacks.

This is achieved by the comparison of a *pair* of binary matrices, where each element represents a logical Boolean value. One matrix, $\mathbf{C}$, represents the **c**urrent activation (presence of touch) of fret-string locations in the audio callback and one matrix, $\mathbf{P}$, which represents activations in the **p**revious audio callback.

To determine note onsets, we can perform the following element-wise logical operation on $\mathbf{C}$ and $\mathbf{P}$, which will generate a third binary matrix, $\mathbf{Onsets}$, which represents the position of note onsets.

$$\begin{gathered}\mathbf{Onsets}_{f,s} := \mathbf{C}_{f,s} \wedge \neg \mathbf{P}_{f,s}, \\ f \in [0 : F-1],\ \ s \in [0 : S-1]\end{gathered} \tag{2}$$

Where F is the number of 'frets' and S is the number of 'strings'.

What follows is the matching expression for the note off binary matrix, $\mathbf{Offsets}$.

$$\begin{gathered}\mathbf{Offsets}_{f,s} := \neg \mathbf{C}_{f,s} \wedge \mathbf{P}_{f,s}, \\ f \in [0 : F-1],\ \ s \in [0 : S-1]\end{gathered} \tag{3}$$

This is visually expressed with a simplified example in Figure 11.

4) Generate MIDI note-on and note-off events: At this point it is possible to convert fret-string locations to MIDI number, and send the required MIDI note-on and note-off instructions. However, this presents us with a problem that is unique to guitar-style MIDI controllers.

We must convert the $\mathbf{Onsets}$ and $\mathbf{Offsets}$ into MIDI instructions so that our prototype can communicate with digital instruments and DAWs in a format that is understood.

Furthermore, we must 'remember' which notes are currently *active* (have had an onset, but hasn't yet had an offset), and to ensure that all *active* notes are correctly terminated at the appropriate time. Failing to do this will result in notes that do not correctly stop at the right time, creating a cacophonous barrage of notes which will ultimately be frustrating for the user.

This requires that each fret-string location is individually addressable, such that when location $(0, 1)$ has an onset corresponding to MIDI note 45, for example, when that corresponding offset is sent, that note is in fact terminated.

We might be tempted to use MIDI notes as the identifier for fret-string locations such that when $(0, 1)$ is played we store in memory that note 45 is currently active, and when we receive the corresponding offset that matches up with MIDI note 45, we terminate that note. This logic would work fine for a piano or keyboard MIDI controller, because each key has a *unique* mapping to any given MIDI note. This is not the case with the guitar. Figure 12a shows the mapping of fret-string locations to MIDI note number for standard EADGBE guitar tuning.

It can clearly be seen that there are MIDI note duplicates in this matrix, which means that there is not a unique mapping of fret-string locations to MIDI notes. This means that a user might press $(0, 1)$ for example, and then presses $(5, 0)$, this

Fig. 5. Final prototype. See Appendix A for more images of the final prototype.

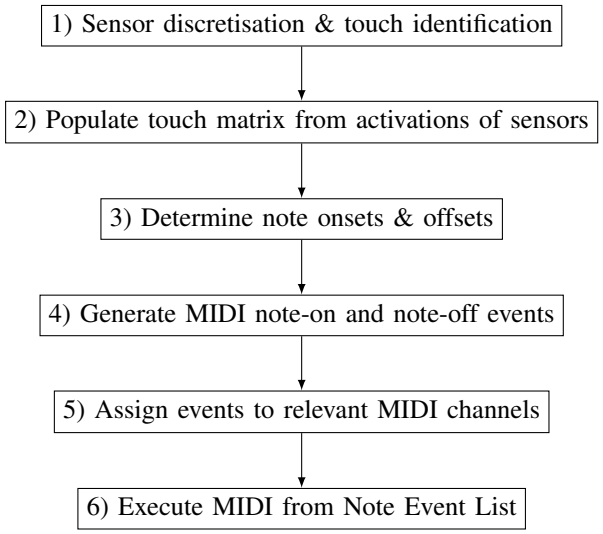


Fig. 6. Logical overview of the sensor data to MIDI event conversion.

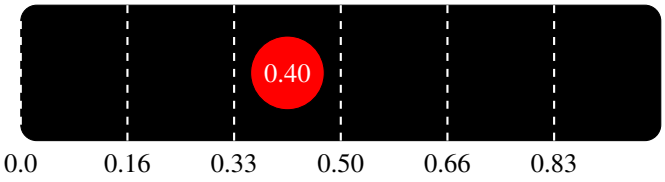


Fig. 7. Demonstration of the sensor boundaries $\mathbf{B}_n$ for N=6. The vertical dashed lines represent the string boundaries $\mathbf{B}_n$, *not* digital strings.

would override the former, and both notes would be terminated if either touch were released.

This behaviour is not analogous to any typical acoustic instrument, and is therefore unlikely to match up with user's expectations. This unexpected behaviour is not only likely to disturb the user's workflow, but is also limits the possible actions of the user, which may not be conducive to their creativity.

```
int getStringZone(float touchLocation)
{
    for i=0:length(Bn):
        if(touchLocation >= Sn[i]):
            return i
    return -1 // Error, out of range.
}
```

Fig. 8. String zone determining algorithm pseudo-code.

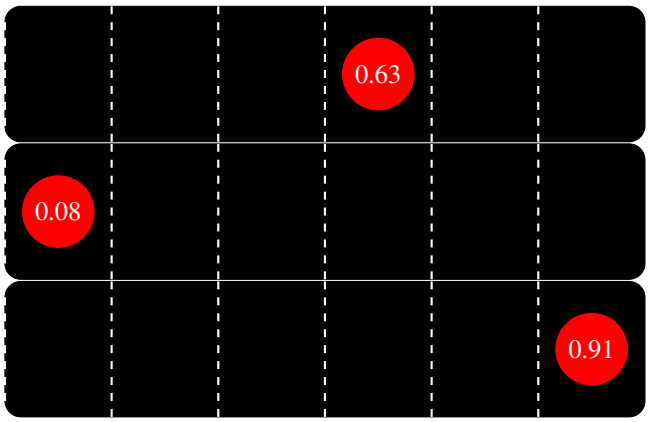


Fig. 9. Example of sensors with touch locations in the relevant string zones. The vertical dashed lines represent the string boundaries $\mathbf{B}_n$, *not* digital strings.

Therefore, we require another method for uniquely identifying fret-string locations. For this we can use the Cantor pairing algorithm. This expression creates an ordered mapping between all pairs of natural numbers to the set of natural numbers. The expression for a two-element Cantor pairing is defined as so:

$$\pi(k_1, k_2) := \frac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2 \qquad (4)$$

Which means that each fret-string location can now be

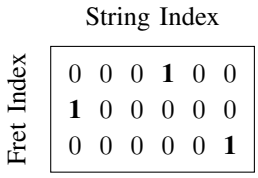


Fig. 10. Binary matrix interpreted from sensor positions.

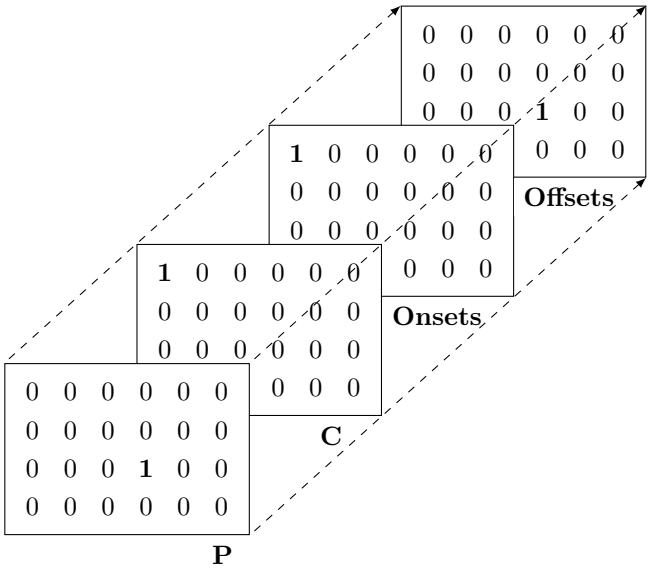


Fig. 11. Demonstration of binary matrix operation for onsets and offsets using the expressions 2 and 3

uniquely identified by a single integer. This means that we can now easily distinguish note events which have identical MIDI note values. We can compare the identifier numbers for each fret-string location by looking at Figure 12. Note that in Figure 12b that there is no duplication in any of the cells of the matrix.

5) Assign events to relevant MIDI channels: Furthermore, to create MPE behaviour, we must be able to assign notes uniquely to MIDI channels in an efficient way. The MIDI 1.0 Protocol works in such a way that each MIDI *channel* has pitch-bend and aftertouch, not each *note*. This means that every note on any given channel would all be equally detuned by the same pitch bend instruction. This is typically very unmusical because very few acoustic instruments exhibit behaviour like this.

This is the motivation to implement MPE behaviour, where each note is assigned to its own channel and thus had independent pitch-bend and after-touch, making for a more expressive instrument.

However this represents a more complex programming task in both the controller/sensor design and the software instrument design. We must remember which notes, or indeed fret-string hashes as determined by expression 4, are currently active. Furthermore, we must free MIDI channels as notes receive their offsets, such that we always choose the lowest

a. String Index (MIDI)

Fret Index					
40	***45***	***50***	***55***	***59***	***64***
41	***46***	***51***	***56***	***60***	***65***
42	47	52	57	61	66
43	48	53	58	62	67
44	49	54	59	63	68
45	***50***	***55***	***60***	***64***	69
46	***51***	***56***	***61***	***65***	70

b. String Index (Cantor)

0	1	3	6	10	15
2	4	7	11	16	22
5	8	12	17	23	30
9	13	18	24	31	39
14	19	25	32	40	49
20	26	33	41	50	60
27	34	42	51	61	72

Fig. 12. Demonstration that there are not unique mappings from fret-string locations to MIDI notes, which therefore cannot be used as unique identifiers for notes. Highlighted numbers are non-unique. Diagram represent the first seven frets.

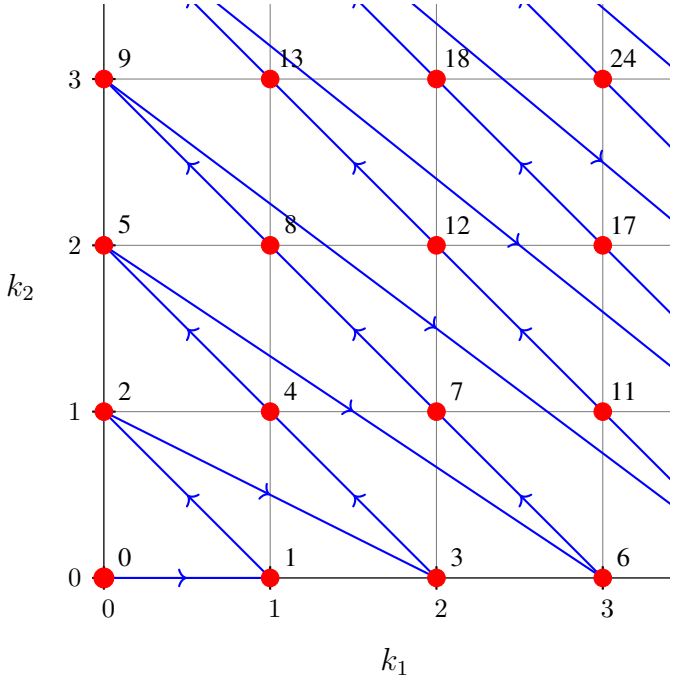


Fig. 13. Visualization of the two element Cantor pairing algorithm.

index MIDI channel as possible. There are only 16 in total, and wish to avoid sending an invalid MIDI instructions, which may crash either one of the controller or MIDI host, which will prevent the user from working.

To complete this task we can create a length 16 list of 'active notes'. Which represents the maximum number of MIDI channels and the maximum number of possible simultaneous notes playable on this controller. This limit is very likely to be sufficient, a maximum of 6 notes are expected (one per string) and a theoretical maximum of 10 notes are playable (one per finger), though the latter would represent a rather experimental technique.

To create the behaviour required, we require multiple data points about each note event, and thus the list (implemented with a C++ standard library `vector`) is populated with the

data structure shown in Figure 14.

```
struct NoteEvent
{
      int midiNote = -1;
      int midiChannel = -1;
      int velocity = -1;
      int pressure = -1;
      int pitchBend = -1;
      float initialTouchLocation = -1;
      int fretStringHash = -1;
      bool isActive = false;
      bool dueOnset  = false;
      bool dueOffset = false;
};
```

Fig. 14. `NoteEvent` data structure.

In this data structure there are some other elements which will be discussed in later sections about MIDI continuous control (CC) instructions.

We can use this list of `NoteEvents` in order to describe which note events should be executed.

When a new note onset is detected, we can determine which element in the list it should occupy with the algorithm described in Figure 15. This algorithm ensures that the selected index in the `noteEventList` is as small as possible.

```
let noteEventList = vector<NoteEvent>{16}
let newOnset = {fret, str, ...}
for i=0:length(noteEventData)-1:
{
    if(!noteEventData[i].isActive)
    {
         noteEventData[i] = newOnset
         noteEventData[i].dueOnset = true
         noteEventData[i].isActive = true
    }
}
```

Fig. 15. Pseudo-code algorithm for determining the index of and creating new `NoteEvent` objects.

Conversely, when a note offset is detected, and stated in the **Offsets** matrix, we can perform the following algorithm to edit the `noteEventList`.

```
let noteEventList = vector<NoteEvent>{16}
let newOnset = {fret, str, ...}
for i=0:length(midiList)-1:
{
    if(noteList[i].fsHash==cPair(fret,str))
    {
         noteList[i].isDueNoteOffset = true
    }
}
```

Fig. 16. Pseudo-code to mark `NoteEvents` as ready to recieve a MIDI note off event.

Conveniently, we can use the indices of the `NoteEvent` list to determine the MIDI channel for each note, as they are always the lowest value possible for a given note, and unique to each note. This is the case because as we can see in line 5 of Figure 15 as we iterate over `i`, we will 'stop' at the lowest value of `i` that does not contain an active note and assign it the new note.

6) Execute MIDI from Note Event List: Now we have a list of which MIDI notes to turn on and off, appropriately matched to a channel such that we can take advantage of MPE style behaviour.

This can be done by the C++ function in Figure 18, which implements the Bela MIDI API.

```
#define CH0_NOTE_ON  144
#define CH0_NOTE_OFF 128
using Midi = midi_byte_t;
void writeNote(int event, // 0->ON, 1->OFF
               int ch,    // midi channel
               Midi note, // note number
               Midi vel = 0) // velocity
{
    int eventType = (event == 0)
          ? CH0_NOTE_ON : CH0_NOTE_OFF;
    Midi status =  Midi(eventType + ch);
    Midi out[3] = {status, note, vel};
    midi.writeOutput(out, 3);
}
```

Fig. 17. C++ code using the Bela MIDI API to send note-on and note-off instructions.

We then need to iterate though all elements of `NoteEvents` and perform the proper function, which can be implemented with the code outlined in Figure 18.

B. MPE Aftertouch and Pitchbend

Currently, we have only discussed how to implement basic note onsets and note offsets. While necessary, this is not sufficient to meet our requirement to implement MPE style behaviour. In order to create a compelling MPE experience, it is typical to implement aftertouch and pitch bend functionality.

Aftertouch typically is an expression of 'pressure' which can be expressed *after* the onset of the note. It is typically used to map a force/pressure on a key to a note's amplitude or timbre. This might be likened to a saxophonist who may start a note by breathing quietly, and then crescendo the amplitude of the note before its offset. As stated above, force from an FSR would possibly make a more appropriate input for this parameter, however we are limited by the capabilities of the Bela Trill sensors, which can only interpret touch size. However, this does serve as a very reasonable proxy for force/pressure.

Since the acoustic and electric guitar is a plucked/strummed instrument, this type of expression is typically not available. This attempt to extend the expressive capacity of the guitar or guitar-like instruments is central to the motivation of this study.

```
void writeMidiFromNoteEvents()
{
    for(int i = 0; i < gNoteEventData.size(); i++)
    {
        if(gNoteEventData[i].isActive)
        {
            if(gNoteEventData[i].isDueNoteOffset)
            {
                writeNote(1, i, gNoteEventData[i].midiNote);
                gNoteEventData[i].isDueNoteOffset = false;
                gNoteEventData[i].isActive = false;
            }

            if(gNoteEventData[i].isDueNoteOnset)
            {
                gNoteEventData[i].isDueNoteOnset = false;
                writeNote(0, i, gNoteEventData[i].midiNote, gNoteEventData[i].velocity);
            }

            writeAftertouch(i, gNoteEventData[i].pressure);
            midi.writePitchBend((midi_byte_t)i, gNoteEventData[i].pitchBend);
        }
    }
}
```

Fig. 18. C++ code using the Bela MIDI API to send note-on and note-off instructions.

Pitchbend is a feature which is typically implemented by a wheel to the left of the keys on standard MIDI keyboards (see Figure 19).

Fig. 19. Typical pitch bend wheel

However since we would like to emulate the expressive bending of guitar strings with this design, we must convert the touch location data into meaningful MIDI pitchbend data.s

These features can be implemented by a simple extension of the framework laid out above. In our basic touch matrices shown in Figures 10 and 11, we store only a single Boolean value. Instead, we could store a more complex data structure which will help implement our MPE behaviour based on other sensor data such touch size and touch location.

For each touch on each sensor, we must collect three important pieces of information.

- `bool isActive`: Whether this location in the fret-string matrix currently has a touch present.
- `float touchSize`: The size of the touch. This is a proxy for pressure, and can be used to implement our aftertouch functionality.
- `float touchLocation`: The precise location on the sensor where the touch currently is.

Now, each matrix $\mathbf{C}$ and $\mathbf{P}$ can be considered a 3D data structure, which would represent the arrangement of touches shown in Figure 9 in the following diagram, Figure 20. This assumes all touch sizes are 0.5 for simplicity, however this is entirely variable between 0 and 1.

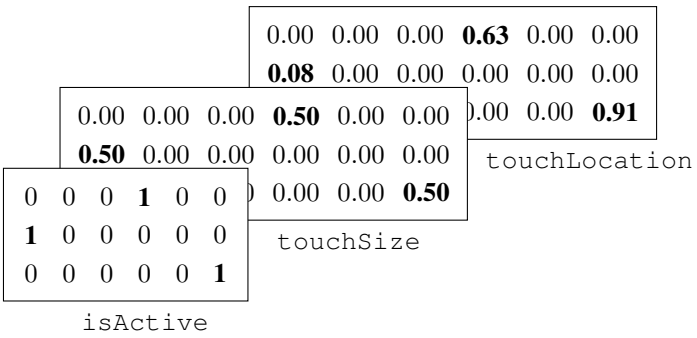


Fig. 20. A visual representation of the updated data structure used to represent touch information.

In the case of aftertouch, all that must be done is to look up the fret-string/Cantor hash of the given location, scale the touch size to the given aftertouch range as specified by the MIDI 1.0 documentation, and write the result to the output as per Figure 18. This scaling can be achieved by the following means:

$$y = \lfloor x \cdot AT_{max} \rfloor \tag{5}$$

Where AT_{max} is the maximum valid aftertouch value. This expression can also be used with maximum velocity V_{max} to give meaningful and valid velocity readings. From the Bela Trill API and touch location and touch size are already normalized to between 0 and 1.

Implementing pitch bend behaviour in this application requires more creative decisions to be made, as there is not a predefined or standard implementation.

On a typical electric or acoustic guitar, pitch bending is achieved by the physical *bending* of the string which causes the tension of the string to increase, which results in the string oscillating at a higher frequency.

In the case of this prototype, we do not have strings at all, only the `touchLocation` data point per note.

In this prototype we have chosen to implement the feature like so.

1) When a note is onset, start the pitchbend amount at zero, regardless of the `touchLocation`.
2) Make a note of the initial `touchLocation`.
3) On each subsequent audio callback, we can subtract the initial `touchLocation`, with the current `touchLocation`, and scale that to within the valid MIDI pitch bend range.

This behaviour was chosen since, this is a new instrument, defining pitch bend absolutely with respect to touch position within each string region would require users to have very exact positioning on the sensor. Failure to do this would result in out of tune notes.

The implemented behaviour only requires that users touch the right string zone, and they do not have to worry about tuning. They can just bend from wherever they land their finger in the touch zone, which should make for a more pleasurable playing experience.

To create the mapping from the difference between the initial `touchLocation`, and the present one to calculate the pitchbend amount took some experimentation, but the expression is shown below in expression 6:

$$PB = 2 \cdot PB_{max} \left(\frac{T_i - T_c}{\Delta T_{max}} \right)^{\alpha} - \frac{PB_{max}}{2} \tag{6}$$

Where PB_{max} is the maximum pitch bend value of 16383; T_i and T_c are the intial and current touch locations respectively and ΔT_{max} is the maximum difference between T_c and T_i (i.e. the width of a given string region).

The maximum pitch bend value refers to the maximum numerical value that is accepted in the MIDI 1.0 specification. 16383 maps to maximum pitch bend *up* (high pitch), 0 maps to minimum pitch bend *down* (low pitch), and 8192 maps to *no* pitch bend. These values are arbitrary units, and are mapped onto real units by the software instrument or DAW it is controlling. It is typical for the maximum value (16838) to map to +2 semitones and the minimum value (0) to map to -2 semitones, but this mapping can be manually edited by users in their software instruments and is not directly within the control of this design.

The exponent α is used to reduce the bending sensitivity near the initial touch location, and exaggerate it when the touch strays further away. This aims to reduce unwanted bending, and to help increase the expressiveness of intentional bending. Several different values were tried for this exponent, but more through user testing will be valuable in refining this expression. In this implementation it has been set empirically to 3. Perhaps it could even be left as something that users could alter in further developments of this prototype.

Following on the theme on extending the creative capacity of the guitar, this technique also allows the guitarist to bend the pitch *down*, which as stated above, is impossible on a traditional electric or acoustic guitar.

V. Hardware Implementation

A. Creating a proto-prototype

Before committing to augmenting a real instrument, a mock-up was made that simulated the rough behaviour, and allowed for software debugging before the full version was created. This is referred to as the 'proto-prototype', since it is the prototype of the main prototype of this project.

Fig. 21. Basic proto-prototype.

This proto-prototype consisted of a small cardboard box that contained the main Bela, and a Trill Hub, with the Trill Bar sensors protruding from the top, which was sufficient to confirm that all the functionality was correct.

B. Choosing a guitar

The width of string regions on a typical 6 string guitar is roughly 0.73cm. This may cause the width of each string zone to be quite small, and, at least, reduce the area for possible for pitch bends. This may also have presented a possible frustrating experience for the user, where it may have be to deliver precise finger movements.

Intention not mapping properly to outcome is likely to be frustrating for users.

This is why, a seven string guitar was used as the basis for the prototype. Although, the prototype does fundamentally have a six string topology, the additional width per string allows for greater margin of error in finger placement. Although no direct comparison to a six-string has been made, this appears to help in the playablility of the prototype.

C. Fitting the sensors

To fit the sensors to the guitar required a few steps of minor carpentry. Because the Trill sensors have a substantial Grove connector on the rear as can be seen in Figure 3. This meant that there had to be a hole cut on each of the frets that were to have sensors mounted. This was achieved by the use of a drill.

Furthermore, the fret boards of typical guitars are not flat, and thus required sanding to accommodate the flat figure of the Trill sensors.

Additionally, the Trill sensors have their small chip protruding from the rear, which make mounting the sensors flush to the fret board difficult, so a corresponding area on each of the frets was extracted with a drill, such that each sensor was able to lie flush with the fret board.

VI. Evaluation

A. Practice-as-research

This prototype has been evaluated with the use of a *practice-as-research* methodology. This evaluative methodology has been chosen for a number of reasons, first of all being the lingering presence of the COVID-19 pandemic which makes in-person user testing very difficult. Furthermore, it offers a unique lens though which to evaluate this prototype.

Practice-as-research is an extension of the *research through design* methodology presented in Gaver (2012), which is described as the "the inference of generalisable principles from a set of design artifacts" (Martelloni et al. 2021).

In Martelloni et al. (2021)'s work, the authors highlight the following benefits of practice-as-research, which are logical extensions of the benefits of Gaver (2012)'s work.

These benefits include the affordance of *pre-paradigmatic research*, which is beneficial because the artefact can be evaluated outside of an established set of rules and standards, and this diversity of analysis, Gaver argues, is beneficial to the scientific and research effort.

Secondly, Martelloni et al. (2021) highlight a second benefit, put forth by Carey & Johnston (2016). This is shown by the concept of the *bricolage* approach to research. This means that "there is quick feedback on the design, as evaluation and prototyping are brought forward in parallel by the same person or the same group of people". This may mean tighter, and more effective circles of design iteration.

To perform this practice-as-research, the author has used the prototype to compose a new song. This presents an ecologically valid situation for the prototype (in a music studio, in the hands of a guitarist), as well as the benefits listed above.

The piece is available to listen to **here**.

The piece was composed using Ableton Live 11 for Microsoft Windows 10. This was chosen since the software has native MPE support (Ableton 2021). It would still be possible to complete the composition in another DAW which does not have native MPE support, but this would require creating 16 instances of each software instrument, each with a unique MIDI channel, to take advantage of the prototype's MPE functionality. It would also make editing software instrument parameters extremely cumbersome since each of the 16 instances would need to be manually updated individually. Using a DAW with native MPE support avoid this.

In the following sections we will be exploring a critical analysis of the process in the creation of the author's composition, with respect to the role that the prototype played, and how it changed the way the track was composed.

1) Sound design: In the composition of the piece, the first thing that was a barrier to creativity was the size of the palette of sounds available that takes advantage of the MPE features of the prototype. The author had developed a preset for Ableton's *Wavetable* synthesizer for debugging and development of the prototype, but beyond this, there did not exist a large palette of sounds to draw from.

To solve this, composition was stopped, and the author performed a sound design session, creating a set of 16 sounds which take advantage of both the aftertouch and pitch bend abilities of the controller. This step made the process of composition much more straightforward, and less frustrating.

Sounds that employ the full feature-set of this prototype are typically legato, that map aftertouch to amplitude and brightness (usually by mapping to a low-pass filter cutoff frequency). One typically expects the pitch bend range to be roughly -2 to +2 semitones, which is typical, however there are more experimental sounds that fully employ the pitch bend feature of the prototype, mapping from -24 to +24 semitones.

2) Playing the parts: The actual prototype is overall quite expressive, and perceivably offers a high degree of fine control over aftertouch and pitch bend features. However, since it was a novel instrument to the author, much of the 'muscle memory' that had built up as a traditional guitarist was not as directly translatable as was expected. This meant that performing some parts of the composition such as the lead line, had to be performed a number of times to record a performance that was free of mistakes.

3) Composition: Overall, the process of composition was subjectively very removed from using a typical non-MPE MIDI controller.

In the following ways, the prototype was more creatively inspiring than a non-MPE controller.

With the appropriate MPE enabled software instruments, the individual note expression made for a novel experience in comparison to the author's previous experience inputting MIDI into a DAW. This novelty perceivably was in itself a component of the success of the instrument, but this does not speak to it's specific nature that much.

Speaking directly as a function of the individual note expression, this was able to create much more idiosyncratic

synthesizer recordings, that would be incredibly tedious to manually program with a mouse. The lead line of the given composition is a prime example of this. Within the four bar loop of the main theme, each note event is completely unique with respect to its duration and aftertouch envelope. This has the subjective effect of creating a more 'organic' or 'human' feeling in the music, which represents a successful implementation of the MPE techniques.

However, the composition process was made more cumbersome with the use of the prototype, insofar as the author's muscle memory had not completely entrained with the nature of the instrument yet, and this nature of being unable to convert intention to reality was at times frustrating. This can be understood as a natural function of it being a new instrument, but one of the aims of this prototype is to be able to more directly translate previously available muscle memory to the MIDI domain. This offers a direction for improvement in future iterations of the design.

Though the novelty was very inspiring, this layer of frustration prohibited some level of creativity.

In comparison to the desktop, keyboard style MIDI controllers, guitar-like MIDI controllers, MPE enabled or otherwise, may be less suitable for studio applications than previously thought.

In a practical sense, having a keyboard on the desk in front of the musician or producer is a 'natural', and well established studio practice. However, the practicalities of balancing a guitar on one's lap, while leaning over to try and start recording, or editing MIDI, after a few hours in the studio is more creatively draining than simply accessing a MIDI controller that is at rest on the desktop.

Though this is perhaps a minor observation, if our goal is to optimise the creative potential for users, this is a problem worth examining.

Subjectively, the author believes that the addition of authentic human expression in the composition is successful. This links to studies such as Witek et al. (2014), which discuss the effect of groove and imperfect timing on the increased enjoyment of music, at least in particular styles.

It was also interesting that, since editing many notes of MPE material in post-production is laborious and cumbersome, this invited performing more takes of the performance to get a satisfactory recording. This practice supports musicianship in favour of over-editing, which can often make for a more authentic musical experience.

B. Heuristic Evaluation

To further extend the evaluation of the prototype, we can conduct a heuristic evaluation. Heuristic evaluation is a method for identifying the usability problems in a user interface design (Nielsen 1994, Nielsen & Molich 1990). Since our prototype is indeed a type of user interface, this evaluative process can add some value to this study.

Heuristic evaluation consists of generally a number of 'expert' user interface designers who will assess the system (our prototype) to a number of 'heuristics', these are outlined briefly in Appendix B.

By walking step by step through these heuristics we highlight a number of successes of the design, but also elements which could be improved. By highlighting these areas for improvement, we hope to contribute to the better understanding of prototyping this type of NIME for future researchers and practitioners.

It is noted that not all of the points analysed here relate directly to the MPE functionality of the prototype, but the frustrations of points raised will cumulatively impact the prototype's ability to support creativity in general, and creating a device that is supportive of creativity is one of the goals of this study.

Furthermore, the author notes the work of Rodger et al. (2020), noting that traditional HCI evaluation of NIME is not necessarily the most effective. Rodger et al. (2020) argue that since instruments are more completely described with reference to their context. The affordances of any one instrument interact with and constrain other affordances and processes of its interconnected systems, for example connected DAWs and digital synthesizers. The authors summarise the issue neatly with the following quotation:

> "Instruments can do and mean many different things that are not easily delimited by a predetermined functional teleology, and this is affected by their evolution among other instruments and musical practices."
>
> *Rodger et al. (2020)*

Traditional HCI evaluation makes the implicit assumption that precision, control and accuracy are valued Rodger et al. (2020), but many subjectively successful NIME do not conform to this assumption Gurevich & Treviño (2007). However, this instrument *does* happen to value control and precision, and since it is a new type of instrument, the following heuristic evaluation offers a broad-strokes analysis of the benefits and shortcomings of the design as it stands.

1) Visibility of system status: Since the system only has a limited set of states, either 'on' or 'off', it is tempting to imagine that there is not need for system feedback beyond a simple power LED. However, this is only a prototype, and as the feature-set grows to incorporate strumming modes, and other input modes, users should be able to know that the system is in fact in the right 'mode'. This will help avoid mode errors, which can be very frustrating for a user. One might imagine a user on stage with this digital instrument, about to perform a solo, but their instrument not properly signify which mode it was in, and result in an undesired output, and unhappy concert-goers.

Currently, while the prototype is indeed a prototype and bound to wired studio use, the power indicator on the Bela board is likely sufficient, but as the features grow, this will change and more sophisticated signifiers of mode will be required.

2) Match between system and the real world: Since this is a new system, which does not behave the same way as a 'traditional' guitar, we say there is an unnatural mapping

between our prototype and the traditional guitar. In this sense there is arguably not a great match between the system and the 'real world' of the traditional guitar, which some users may find jarring.

We might consider two solutions to this. Either, the prototype should support traditional guitar style playing, so that users can use the system in a way which matches up with their expectations, or it is somehow able to make it clear to the users what *is* possible with the prototype and what is *not*. This could be done with forcing functions by having definitively no place to strum, and thus prohibit this behaviour, for example.

However the latter is not conducive to this study's goal of creating a system which is conducive to creativity. A system that aims to foster creativity should have affordances for self-expression that support users in as many ways as possible.

3) User control and freedom: This heuristic is effectively a question of whether the device supports undoing mistakes, or cancelling unwanted tasks.

This heuristic is not so applicable to our prototype in it's current state just like heuristic *1)*. However this is something to be aware of in future iterations. If a user accidentally changes mode, it should take one *easily identifiable* 'click' or button press to get back to where they were.

4) Consistency and standards: This prototype implements MIDI, which is a standardized digital protocol. This prototype should, and broadly does, apply this standard. However, this is necessary but not sufficient. Most MIDI instruments exhibit a 'plug-and-play' functionality, which means that as soon as they are plugged into the computer, they immediately begin to work. This is a convention that the prototype does follow by using the Bela 'Run project on boot' functionality as can be seen in Figure 22.

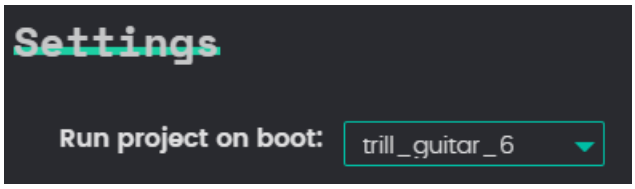


Fig. 22. The Bela IDE's 'Run project on boot' functionality.

However, the Bela ambiguously identifies its MIDI port as 'Midi function', as can be see in Figure 23. It is a typical practice for MIDI devices to identify themselves with a helpful name such as "AugGuitar Prototype V1.0" for example.

Fig. 23. The Bela MIDI device appearing with as 'MIDI Function' in the MIDI ports list of Ableton Live 10.

This is a type of *external* inconsistency (Nielsen & Molich 1990), and is likely to confuse users on the first time using it, which may impede their creativity.

5) Error prevention: As highlighted in the section above, the lack of visual and haptic (for example emulating the physical sensation of strings across the sensors) feedback as to the position of string centres does not support error prevention particularly well.

It is not very forgiving of accidental incorrect finger placement, and only permits users who already have a muscle memory of the position of the strings to work without error. Despite being the developer of the prototype, even the author does not possess this ability.

The addition of haptic feedback (e.g. dummy strings across the sensors) would also be more supportive of visually impaired users, which would be beneficial.

On the other hand, the prototype's pitch bend functionality is more successful in this sense. Where the prototype does not bend the note, regardless of initial touch location and the addition of the α exponent in expression 6 helps in the reduction of unwanted pitch bends. This should help make the experience less frustrating, and ultimately more creatively supportive.

6) Recognition rather than recall: There are few things to have to remember about the current design with respect to it's usability. Insofar as there are no deep menus to traverse. Thus, this heuristic is less applicable to the prototype than some of the others.

7) Flexibility and efficiency of use: Under heuristic *2)* we touched on a point about some users that will want, or expect, the prototype to function like a traditional guitar. In this way, the prototype is not flexible, since it only supports one style of playing. To amend this, a greater number of playing styles could be supported, which would increase the flexibility and usability of the prototype.

8) Aesthetic and minimal design: Currently the prototype is very minimal indeed, and certainly does not show superfluous information to the user. It might be argued that the vestigial volume and tone knobs, and pick up selector tack against this heuristic, but they naturally would not continue into later iterations of the design since they serve no functional purpose in the system.

9) Help users recognize, diagnose and recover from errors: Currently, the prototype offers implicit feedback to slips where the wrong note is pushed, because the wrong note will sound. However this point is somewhat of a triviality.

As the prototype develops, it should offer more sophisticated feedback to general troubleshooting issues such as drivers failing to load, etc. However, issues like this are not the concern of this prototype.

10) Help and documentation: This heuristic is also less applicable to this prototype. The interface being a guitar shape is reasonably self-evident, but as the feature-set grows supporting documentation about driver installation should be created as required.

VII. Summary and Conclusion

To conclude, we have explored how capacitive touch sensors can be applied to the guitar to create an expressive MPE

instrument. The interaction design for this instrument is not yet perfected, but offers a direction for further development.

We have proposed a framework for mathematically interpreting the sensor data which can scale to any number of frets or strings, which means that this system can scale indefinitely to accommodate any sized system.

We have evaluated how the prototype is overall effective in delivering the desired expressive features, but requires some refinement to be truly creatively liberating.

ACKNOWLEDGEMENT

I would like to thank everyone in the **C4DM** at QMUL, each of which in their own way has helped to inform my understanding and interest in digital audio, music and human-centred design. In particular Mathieu Barthet, Andrew McPherson, Paul Curzon and Sophie Skatch who have all been instrumental in supporting my learning in the modules they teach.

In particular, Mathieu Barthet has been an extremely attentive and supportive supervisor to this project and I thank him immensely for his support on this journey.

REFERENCES

Ableton (2021), 'MPE in Live 11'.
URL: *https://help.ableton.com/hc/en-us/articles/360019144999-MPE-in-Live-11*

Avila, J. P. M., Hazzard, A., Greenhalgh, C. & Benford, S. (2019), Augmenting Guitars for Performance Preparation, *in* 'Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound', AM'19, Association for Computing Machinery, New York, NY, USA, pp. 69–75.
URL: *https://doi.org/10.1145/3356590.3356602*

BBC (2006), 'UK guitar sales at £100m high'.
URL: *http://news.bbc.co.uk/1/hi/entertainment/5281620.stm*

Carey, B. & Johnston, A. (2016), 'Reflection On Action in NIME Research: Two Complementary Perspectives', *NIME* p. 6.

Csikszentmihalyi, M. (1990), Flow: The Psychology of Optimal Experience.

DataCommons (2020), 'United Kingdom - Place Explorer - Data Commons'.
URL: *https://datacommons.org/place/country/GBR?utm*$_m edium$ ¥ork, New York.
$explorem prop = countpopt = Personhl = en$

Enterprises, S. (2021), 'Stick Enterprises - Home of the Chapman Stick'.
URL: *http://stick.com/instruments/*

Gaver, W. (2012), What should we expect from research through design?, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', Association for Computing Machinery, New York, NY, USA, pp. 937–946.
URL: *https://doi.org/10.1145/2207676.2208538*

Gurevich, M. & Treviño, J. (2007), 'Expression and Its Discontents: Toward an Ecology of Musical Creation."'.

John, J. (2020), 'Personalisation of Augmented Guitar: A Machine Learning Approach', p. 10.

Khalil, C. (2019), 'CROWDSOURCED GUITAR SYNTHESIS USING PHASE VOCODER TECHNIQUES', p. 42.

Lamb, R. O. (2014), 'The Seaboard: Discreteness and Continuity in Musical Interface Design - ProQuest'.
URL: *https://www.proquest.com/openview/5042899d203aa06568b9696c origsite=gscholarcbl=51922*

Lähdeoja, O. (2008), 'An Approach to Instrument Augmentation : the Electric Guitar', p. 4.

MacDonald, R., Byrne, C. & Carlton, L. (2006), 'Creativity and flow in musical composition: an empirical investigation', *Psychology of Music* **34**(3), 292–306. Publisher: SAGE Publications Ltd.
URL: *https://doi.org/10.1177/0305735606064838*

Martelloni, A., McPherson, A. & Barthet, M. (2020), 'Percussive Fingerstyle Guitar through the Lens of NIME: an Interview Study', p. 6.

Martelloni, A., McPherson, A. & Barthet, M. (2021), Guitar augmentation for Percussive Fingerstyle: Combining self-reflexive practice and user-centred design, PubPub.
URL: *https://nime.pubpub.org/pub/zgj85mzv/release/1*

Meneses, E., Freire, S. & Wanderley, M. (2018), *GuitarAMI and GuiaRT: two independent yet complementary Augmented Nylon Guitar projects*.

Metlay, M. P. (1990), 'The Musician-Machine Interface to MIDI', *Computer Music Journal* **14**(2), 73–83. Publisher: The MIT Press.
URL: *https://www.jstor.org/stable/3679720*

Millard, A., ed. (2004), *The Electric Guitar: A History of an American Icon*, illustrated edition edn, Johns Hopkins University Press, Baltimore.

Nash, C. (2012), Supporting Virtuosity and Flow in Computer Music, PhD thesis, University of Cambridge.

Nielsen, J. (1994), *Usability Engineering*, Morgan Kaufmann. Google-Books-ID: 95As2OF67f0C.

Nielsen, J. & Molich, R. (1990), Heuristic evaluation of user interfaces, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '90, Association for Computing Machinery, New York, NY, USA, pp. 249–256.
URL: *https://doi.org/10.1145/97243.97281*

Norman, D. (2013), *The Design of Everyday Things: Revised and Expanded Edition*, 2nd edition edn, Basic Books, New

Roads, C. (2015), *Composing Electronic Music: A New Aesthetic*, illustrated edition edn, Oxford University Press, Oxford ; New York.

Rodger, M., Stapleton, P., van Walstijn, M., Ortiz, M. & Pardue, L. (2020), 'What Makes a Good Musical Instrument? A Matter of Processes, Ecologies and Specificities', p. 7.

Roland (2021), 'Roland - GR-20 | Guitar Synthesizer'.
URL: *https://www.roland.com/uk/products/gr-20/*

Stubbs, D. (2018), *Mars by 1980: The Story of Electronic Music*, main edition edn, Faber & Faber, London.

Turchet, L. & Barthet, M. (2019), 'An ubiquitous smart guitar system for collaborative musical practice', *Journal of New Music Research* **48**(4), 352–365.

URL: *https://www.tandfonline.com/doi/full/10.1080/09298215.2019.1637439*

Tyler, J. (1975), 'The Renaissance Guitar 1500-1650', *Early Music* **3**(4), 341–347. Publisher: Oxford University Press.
URL: *https://www.jstor.org/stable/3125401*

Wanderley, M., Pompidou, I.-C. & Stravinsky, P. I. (2001), Gestural Control of Music, Technical report.
URL: *https://www.semanticscholar.org/paper/Gestural-Control-of-Music-Wanderley-Pompidou/8d6990b1c2164e36a18cd57ac8609f297c524288*

White, P. (1984), 'The Guitarist's Revenge (EMM Jun 1984)', *Electronics & Music Maker* (Jun 1984), 10–16. Publisher: Music Maker Publications (UK), Future Publishing.
URL: *http://www.muzines.co.uk/articles/the-guitarists-revenge/7885*

Witek, M. A. G., Clarke, E. F., Wallentin, M., Kringelbach, M. L. & Vuust, P. (2014), 'Syncopation, Body-Movement and Pleasure in Groove Music', *PLOS ONE* **9**(4), e94446. Publisher: Public Library of Science.
URL: *https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0094446*

Yamaha, C. (2003), 'Yamaha EZ-AG Manual'.
URL: *http://www.midimanuals.com/manuals/yamaha/ez-ag/owners*$_m anual/ezag_e$*.pdf*

APPENDIX A
FINAL PROTOTYPE PHOTOS

Fig. 24. Front of the prototype.

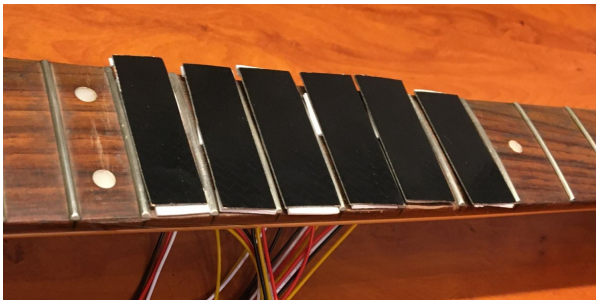


Fig. 25. Close-up of neck sensors.

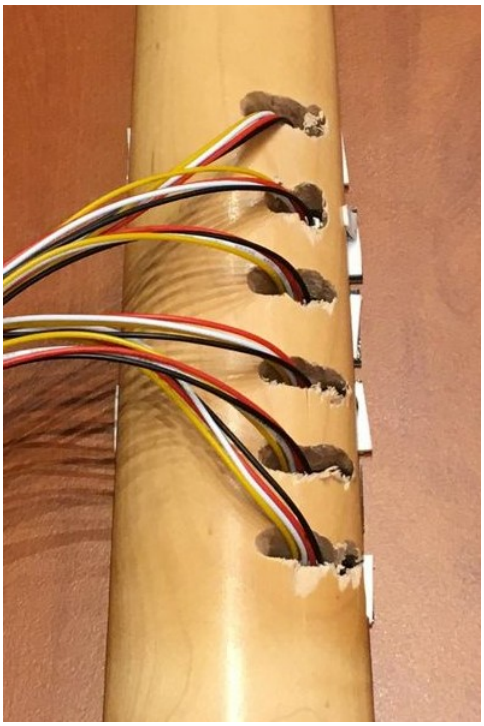


Fig. 26. Back of the neck, to show I2C wiring to the sensors.

APPENDIX B
NIELSEN USABILITY HEURISTICS

1) **Visibility of system status**: does the design keep the user informed about what is going on?
2) **Match between the system and the real world**: does the design follow real-world conventions and use phrases and concepts which are familiar to the user?
3) **User control and freedom**: are users able to easily correct mistakes? This helps foster a sense of freedom and confidence in the user.
4) **Consistency and standards**: does the design follow platform and industry standards/conventions?
5) **Error prevention**: it is important to let the user know when things have gone wrong, but it is even better to help prevent errors in the first place. Does the design help prevent common errors from happening?
6) **Recognition rather than recall**: does the design minimise the user's memory load by making the interface as available as possible? Does the user have to remember different parts of the interface are hidden in deep context menus?
7) **Flexibility and efficiency of use**: does the design offer shortcuts to expert users such that they may be able to speed up their workflow, but at the same time allowing novice users to access all the features at their own pace?
8) **Aesthetic and minimal design**: does the interface contain information which is not needed? Interfaces should aim to minimise irrelevant information, since this reduces the relative visibility of important, relevant information.
9) **Help users recognise, diagnose and recover from errors**: are errors expressed in plain and simple language? Does the system precisely indicate problems and suggest solutions?
10) **Help and documentation**: does the system require additional documentation? Though it's best if the system doesn't need additional explanation, if this *is* required, does the system offer this help?

MSc Project - Reflective Essay

Project Title:	Augmenting the guitar as an MPE controller with capacitive touch sensors
Student Name:	Jamie R. Pond
Student Number:	160568512
Supervisor Name:	Dr. Mathieu Barthet
Programme of Study:	Sound and Music Computing MSc

Introduction

Before I take a deeper look at the process involved in this research, I would like to extend my deepest gratitude to my supervisor Dr. Mathieu Barthet, who has fostered my learning through his encouragement and insightful feedback.

The most successful element of my project is the amount that I have learned about all aspects of computer science and music, and how much I have personally grown over its duration. At the start of the course in September, I was relatively new to programming in C++ and many of the mathematical notions used in this report. This MSc, and this project in particular, have been especially challenging. However, this challenge has offered much personal growth, and thus has endowed my struggle with meaning, for which I am incredibly grateful.

I am indebted to the wonderful modules and lecturers who have, respectively, informed and inspired this project. Special mention goes to Mathieu Barthet, Andrew McPherson and Paul Curzon for their insightful and creative teaching of *Sound Recording and Production Techniques, Music and Audio Programming,* and *Interactive System Design,* respectively.

This project was borne from a desire to create musically meaningful technology, something that is helpful in a creative setting. After having been introduced to Dr. Barthet's research on the augmented guitar, and as a guitarist myself, I was keen to delve deeper into the topic.

Analysis of strengths/weaknesses

I was especially pleased to have come up with a solution to the fret-string location addressing problem. Initially, I had an idea that I could somehow 'hash' fret and string data into a single number and tried to come up with an expression myself after having written a short test script. This was particularly problematic, even not minding the efficiency of the mapping of pairs, all that mattered is that they were unique. Eventually, I thought to research this question further, and was delighted with what I found. Cantor's pairing algorithm was a real delight to get to know, and I was blown away by its beauty and efficiency. I was pleased to have been able to incorporate such an interesting and elegant piece of mathematics into my project, and moreover one that was new to me.

I am overall pleased with the effectiveness of the prototype. Though it has its limitations and shortcomings with respect to its design and conception, I am overall very pleased with its ability to be musically expressive.

Initially, I was worried that the lack of *pressure* sensing as opposed to 'touch size' sensing from the capacitive sensors would be a limiting factor for the expressiveness of the prototype, but this was much less the case than I thought it would be. Though the prototype is not very responsive to extreme hard presses as expected, this is generally not perceived, and most gestures are intuitively mapped into sound. This is especially the case for legato-style sounds, which are especially successful.

Though the composition was more or less a showcase for the abilities of the instrument, I believe that the instrument afforded some inspiration that resulted in the subjective success of the piece. I was especially pleased with the expressive sound of the main lead line.

I was also very pleased to have practised my C++ programming skills. To implement the matrices above, and to copy, store, compare and move them around, which initially took some thinking about to figure out the best way to structure it all. I was especially pleased with the 'trick' that I implemented when it came to marking the 'current' matrix **C** as the 'previous' matrix **P**. Initially, I was copying the values of **C** into the memory of **P**, and this was working just fine. However, this was highly inefficient since there is no need to copy the data, since, if **C** and **P** were pointers, the pointers can just be swapped over, which would have the exact same effect, and eliminate all of the copy instructions. This didn't actually seem to have an enormous impact on the CPU performance of the program, however, if this system was to be implemented on a more stripped back embedded chip, then every line of assembly counts, and this optimisation would have been very useful indeed.

On a more personal level, I derived immense pleasure from the process of designing and programming the diagrams included in the paper with LaTeX. I was excited that each of the Bela Trill Bar diagrams was proportionally correct, and this further motivated me to ensure that the paper as a whole was aesthetically appealing. However, creating the Cantor pairing diagram was especially challenging. Ironically, LaTeX is a well-equipped programming language for displaying mathematical notation, but not so well equipped to actually perform mathematics! To display each node on the diagram required the calculation of that number. Since LaTeX does not support mathematical symbols like Python and C++ for example, performing the simple Cantor expression required multiple lines of LaTeX that were much more reminiscent of assembly than any modern programming language.

However, there are some other elements of the project that I would have liked to have done differently if I had the opportunity to do it again. This links to the later section of *'Work that you would have conducted if you had more time',* but we will try and keep this section for experimental criticism of the project, instead of additional features or things I didn't get a chance to add, which I will reserve for the latter section.

It is a shame that real user testing was not able to take place, which would have enabled more types of analysis of the prototype. This possibly could have offered greater insight to the NIME (New Interfaces for Musical Expression) community about how the prototype could be improved, and therefore increasing the overall utility of the study. However, I was very interested to further explore the world of practice-as-research, which has been very inspiring from a research perspective.

I would have liked to have written more songs and material with the prototype, which would have given a broader range of material and experience to analyse. Again, this is likely to have increased the validity and value of the paper by being able to provide, if not more *numerous,* then more *valid* insights about the design. Additionally, it would have been fun, because the prototype is indeed, rather fun to use! It will make for an enjoyable composition companion.

It could have been interesting to have performed more deliberate listening tests. One could imagine creating two versions of the composition, one that employs the MPE recorded material, and one that contains the same musical material, but has this information entered in with a mouse or standard MIDI keyboard. These different compositions could then be shown to participants in a study and their reactions and feelings about the different recordings could be analysed. It would be interesting to see if one or the other elicited a stronger feeling of 'emotion' or 'enjoyment'. If our aim is to understand what interfaces elicit the intended emotions from listeners, this would be important.

Presentation of possibilities for further work

For this project, there is much overlap between the *possibilities for future work* and *work I would have conducted if I had more time.* However, there is a distinction, and I will endeavour to make this clear. Since the whole field is rich and exciting, if I would have had more time, I would have done more of the interesting 'future work' for this project! However, I will limit this section to work that is self-evidently outside the scope of an MSc project, the latter section will contain work that conceivably could have been in the scope of this MSc project.

As far as future work goes, I would be most interested in implementing FSRs (Force Sensing Resistors) into the design of the fret sensors. Enabling the combination of FSRs and capacitive sensors would enable both the pitch-bending techniques which are available to the prototype, but also more receptiveness to a greater dynamic range of physical input, which may make the prototype more perceptibly expressive.

It would also be interesting to try and minimise the amount of hardware such a system would require to run. The Bela/Beaglebone Black is very overpowered to perform the given task. A much simpler architecture could be designed that would achieve the functionality at a fraction of the cost, especially if produced in bulk. This would of course increase the complexity of the engineering effort required to achieve this goal. However, minimising the hardware requirements for the code would require greater awareness of the performance of the code, and thus it would be very important to measure its efficiency so that goals could be set, and then worked towards with respect to the new hardware requirements. This would be essential if deploying this as a commercial product and cost-per-unit had to be minimised.

As mentioned in the report, sometimes the guitar felt like it was a cumbersome shape. This is not so much a function of the shortcomings of the prototype specifically, but guitars in general. As a guitarist, this is something I was already aware of, having tracked guitars in studios before. To address this problem, one might imagine proposing an entirely new shape for the body of the guitar, which is less cumbersome to hold for long periods of time in the studio. A standard MIDI keyboard does not have this problem, since it is able to simply sit on the desk until needed, not requiring any additional support from the user.

It could have been very interesting to add an X-Y pad to the system, possibly by employing a Bela Trill *Square*, which has capacitive touch sensing in both the X and Y directions. This would allow more complex sound manipulations on the fly. For example, this would allow a user to play a part in the left hand, and continuously vary two parameters with the right hand. This mapping may be to a low-pass filter cutoff and reverb mix control, for example.

It could have been interesting to implement some aspect of IOT (Internet of Things) connectivity, building on previous research in this area. For example, Cynthia Khalil's (2019) work on a guitar that was able to communicate with *FreeSound.org*, and download and play samples in real-time, which was very interesting to read about. It could be imagined that this prototype could be augmented to also download content from the internet and play sounds directly from the Bela audio output. This would also incorporate designing an MPE sample player, which would be very an interesting and rewarding endeavour in itself.

Work that you would have conducted if you had more time.

On the prototype, I would have been more particular about tidying up wires. Subtle things like the short USB cable to the guitar was generally frustrating and was possibly limiting the creative potential of the prototype. It would have been very advantageous to have been more prudent with the tidying of the wires. However, with the ongoing COVID-19 pandemic, it is more difficult to get to campus to make use of facilities such as the Electronics Lab and the Creative Hub. I think it would have contributed a lot to the success of the project if all the wires were able to be tidied up, perhaps into the neck somehow and a longer USB cable was acquired which would allow the user to move further away from the computer to which it is attached. It would have been great to make the prototype wireless.

One of the biggest limitations of this prototype is the lack of a way to 'strum' the guitar as the guitar is traditionally used. Adding in this functionality would add the affordance of being able to play the prototype as if it were much more like a traditional guitar. However, the playing style intended for this prototype was the two-handed neck approach which is used by several MIDI guitar practitioners, for example, Rob Swire from Pendulum. Though this prototype supports this style of playing well, it doesn't particularly support traditional playing nearly as well. All that would be required is a Trill Bar sensor on the body of the guitar, in roughly the place a guitar pickup might go, and to be discretized in the same way as the fret sensors are, and apply the following logic:

1. Let **S** (strum) be a vector of length numStrings {0, 0, 0, 0, 0, 0} containing touch locations.
2. For each *string* index find the highest value that is pressed. E.g. if fret 1 and 7 are being pressed simultaneously, return only 7. This mimics the behaviour that lower frets do not sound if a higher fret is pressed simultaneously on the same string.
3. If the corresponding string index of the strum bar to each fret-string location that is marked by the step above is active at the same time, write that fret-string location's note on.
4. When the fret touch is released, send the corresponding note off.

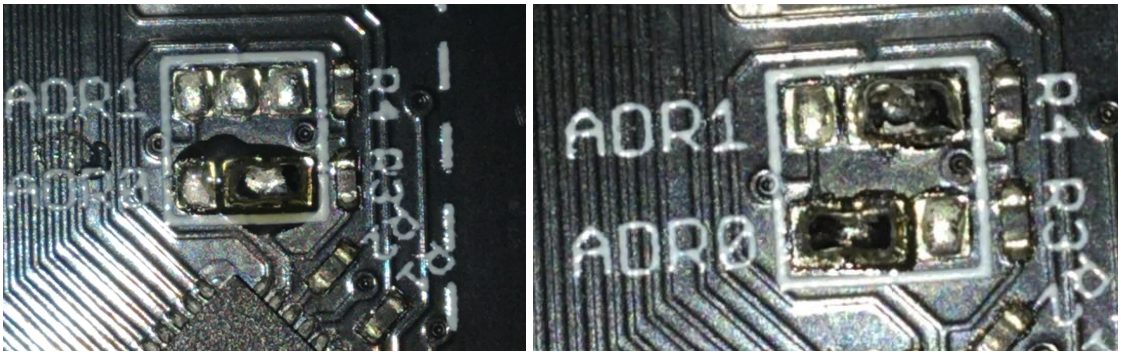
This would not be an immense programming challenge with the existing code infrastructure. One could even have the touch sizes of the sensor bar correspond to the aftertouch of the notes, not the frets sensor's touch size. This could make for a more intuitive playing experience, since typically on the guitar the (left) fret/neck hand takes care of the pitches of the note, and the (right) body hand takes care of the amplitude of the sound by plucking the strings. This would keep this same dynamic.

It also could have been really interesting to have added a gyroscope to the head of the guitar, to add another 'contact point' of expression. By tilting the guitar to map to a MIDI CC such as the control wheel, or other modulation, as is expressed in Lahdeoja (2008). This would have been a straightforward way to add additional expressivity to the guitar. However, time was not permitted in this instance.

I would have liked to augment the touch sensors with some kind of haptic equivalent to strings, since that was demonstrably something that made the prototype more difficult to play. Even with some kind of visual indication of where the centre of the string zones are, this is likely to improve the playing experience. I frequently found that I was unable to hit the correct *string* on a given fret. I had planned (and even purchased!) a can of Plasti Dip plastic spray that could be applied with a stencil across each of the sensors, which would give the visual appearance of strings, but also not impede the sensing of that region too much. This also comes with the benefit of giving some haptic feedback and the fingertips reach over the strings. This would make the experience much more like a traditional guitar, where you can *feel* the strings across the neck of the instrument, and let one's muscle memory guide you. Regrettably, time was not permitted for this addition.

Furthermore, there are aspects of the existing project that I have been unable to write about in the final report due to time constraints. For example, the particularities of the wiring and the use of the I2C protocol are scarcely mentioned, and are more or less implied through the reference to Bela and Trill. However, there was an interesting learning experience for me, having to solder the tiny blobs of solder required to change the I2C addresses of the Trill Bar sensors.

This was a new experience for me, the images of which can be seen below, and don't quite communicate the subtlety of the handwork required.



However, this was a very rewarding learning experience, as this was something I had never done before.

Critical analysis of the relationship between theory and practical work produced

I learned that theory is important across a broad spectrum of disciplines that informed this project. I was especially impressed by the importance of theory in mathematics and in design over the course of this project.

The relationship between the theory and practice of design thinking was very profound and interesting. Having taken, and thoroughly enjoyed the *Interactive System Design* module led by Paul Curzon and Sophie Skach, I was keen to apply what I had learned to this project. Having the vocabulary and cognitive toolkit to thoroughly analyse the prototype has been invaluable. I was particularly interested in Don Norman's *The Design of Everyday Things* which was enlightening with respect to how it reframes thinking about designing anything and everything. Norman's concepts offer applicability from everything to designing augmented guitars, to lamps to programming languages, which was an incredible perspective shift for me. Evaluating this prototype with respect to Nielsen's heuristic design principles was incredibly useful in highlighting the shortcomings of the design, and makes me excited to perform another design iteration on this project.

It was also fascinating to see the applicability of what is typically very abstract mathematics in very practical use with respect to the Cantor pairing. I was not expecting to have to use such a niche part of mathematics in such a practical implementation.

These two examples made me further appreciate the importance of theory (in all related fields!), because of its inextricable and empowering link to one's practice.

Awareness of legal, social, ethical, and sustainability issues.

In a world without COVID-19 I would have mentioned the importance of ethically treating participant's data from user testing stages.

I would however like to highlight the ethical issue of academic integrity. This is something that is important, and absolutely an ethical challenge. In all walks of life, we must hold ourselves to the highest possible standards such that we are able to positively contribute to the world. Academia is no exception. It is arguably one's moral duty to ensure the utmost academic rigour since, without doing this and involving plagiarism in one's work, for example, diminishes the value of one's work to effectively zero, and this represents a massive waste of public resources (given that many academic degrees are funded at least initially by public money). This offers little growth for the individual in question, which is a great shame for them, but also since education is a Merit Good (as an economist might tell you), this also deprives the broader society of one more, better educated and sharper thinking individual, which is in my estimation, not only a great shame, but morally unacceptable. Furthermore, it specifically deprives research areas of another genuine step forward in the quest for better knowledge, and this is also a great pity. With so many other scientists and academics working so hard to develop their field and earning the recognition they deserve, it is morally and ethically reprehensible to attempt to bypass having done the work. This is why it was so important to me personally to ensure that my work was not only of the highest standard that I could muster, but also the most integral and honest that I could produce too.

I would also like to touch briefly on the point of inclusive design. Though these considerations do not fall squarely in the scope of this project, this is worth bearing in mind. This prototype was initially conceived with accessibility in mind, and to be a focus of the project.

Being able to support a wide range of people with one's product is ethically important. If one believes that people should not be excluded from participating in given activities based on their physical (or mental) differences. Of course, there are upper bounds to what is possible, even with the most considerate 'universal' design, however, it is arguable that there is an ethical duty to create products and experiences that as many people as possible can enjoy, so as many people as possible can benefit from a given product.

Conclusion

To conclude, I would like once again to thank everyone at the C4DM, and in particular, Mathieu Barthet, who has been invaluable in supporting this project. I would also like to thank Andrew McPherson for also being so generous with his time and offering feedback on the final draft of this report.

I believe that I have grown much as a computer scientist, engineer, mathematician and musician as a result of this project, and I am so grateful for the opportunity to have completed it.