Controlling MIDI via OSC and applying it to a two-handed layout for a keyboard

Omkar Bhatt
Stony Brook University
obhatt@cs.stonybrook.edu

Rohit Rawat
Stony Brook University
rorawat@cs.stonybrook.edu

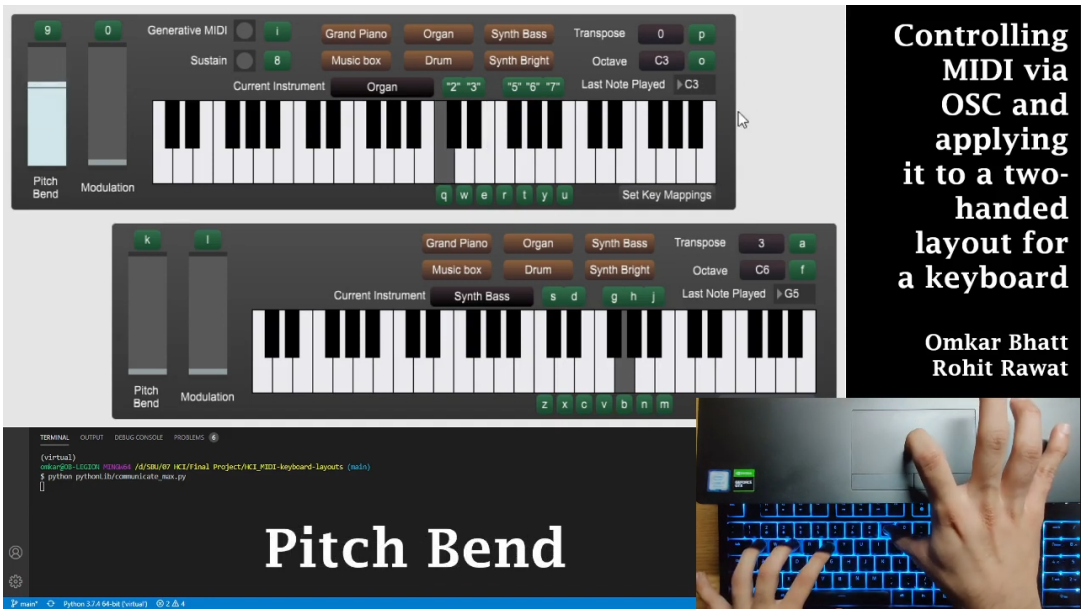

Figure 1: Two-handed layout for a MIDI keyboard controller, screenshot taken taken from our demo video

ABSTRACT

MIDI (Musical Instrument Digital Interface) is a fairly common communication protocol used by many musicians today for electronic music. It helps to convert analog signals from a device, like a MIDI controller, to digital data, which can be interpreted and played as musical notes. There are various MIDI devices available in the market today to help musicians experiment and play around with their play style. We are proposing an architecture to help MIDI devices out of anything that can be connected to a computer and inputs can be received from it. We primarily plan to do this over a computer keyboard. A keyboard layout already exists for the computer keyboard used by many music production software like Ableton Live, but its is not editable. With our approach, we plan to introduce a way to play MIDI dynamically, using custom key mappings and that too over multiple MIDI keyboards at the same time. We feel this would give musicians a better interface to customize their musical inputs, and this would also help music software developers, with a better architecture to interact between devices and MIDI.

Our code is open-sourced and available on Github, and a demo video is also available here.

KEYWORDS

OSC, MIDI, MIDI Controllers, Heuristic Evaluation, Asynchronous Communication

1 INTRODUCTION

MIDI was invented as a standardized means of synchronizing electronic musical instruments manufactured by different companies, as each manufacturer has their own proprietary standards. Ikutaro Kakehashi who invented MIDI felt this limitation of standardization was limiting musical expression as musicians would need to understand different protocols for different instruments. Smith and Sequential Circuits engineer Chet Wood devised a universal interface to allow communication between equipment from different manufacturers. Smith and Wood proposed this standard in a paper, Universal Synthesizer Interface. [10] Even though it was earlier limited to just professional musicians and record producers for electronic music, with time, it has become popular among amateur musicians and beginners as this communication protocol reduces hardware cost easily as well as helped in development of many musical software over the years. The most common MIDI instruments today would be samplers and digital synthesizers.

Over the years, not only has MIDI helped reduced costs for new musicians, but also made it easy to record music at home without a professional studio setting. One other feature MIDI provides is, changes analog signals to digital signals, so capability for remote control allows full-sized instruments to be replaced with smaller sound modules, and allows musicians to combine instruments to achieve a fuller sound, or to create combinations of synthesized instrument sounds, such as acoustic piano and strings. MIDI also enables other instrument parameters (volume, effects, etc.) to be controlled remotely.

MIDI has been very useful in the field of composition as well, be it with inputting data into a DAW (Digital Audio Workstation) like Ableton Live, FL Studio or Reaper as well as using it for algorithmic composition. The ability to compose ideas on the go and play it back to a composer helps them experiement with musical styles and expressions. [6] This also makes it easier for artificial intelligence algorithms to run over the MIDI data to get meaning information and generate new music on the go as can be seen while reviewing algorithmic composition. [4] One other example would be referring to the paper [5], where they use deep learning network techniques to augment virtual environments by the audio generated in MIDI.

Keeping that in mind, we want to propose a way where musicians and software developers could design and assemble a better interface for MIDI to be played from any device. In our case, we are sticking to a computer keyboard, but the possibilities to this are endless and our approach can be used to control and communicate between any device to produce musical outputs. Our approach will also let anyone convert any MIDI data files into a synthesized music, and will also provide a outlet to visualize it this. This not only helps play music but also help musicians see what exactly is being played on a GUI. With further work, we can also try interactions to communicate with various machine learning techniques to perform analysis on MIDI data and generate music on the go. We were able to make a basic Recurrent Neural Network which understands how music can be structured in MIDI and generate new MIDI music with a random seed. This has been integrated to our library to demonstrate how generative music can also be easily integrated and played with our system. Previously, MIDI keyboard has been made for the QWERTY keyboard. We have taken it to further level and implemented the whole MIDI keyboard with 2 layouts on a QWERTY keyboard and incorporated real-time controllers to control pitch and modulation using touch pad of a laptop or a mouse device. It will ease the replacement of original standard piano keyboard by a QWERTY keyboard and provide the essential experience to the Musician users.

In our interactions with musicians who use MIDI or are interested to work with it, the most common problem was their lack of manipulation of MIDI easily via a GUI. It seems that it could be difficult for such users to comprehend heavy machine learning research evolving in the field of computer science and apply it to music. One of of our goals also is to make it easy for such users to integrate our library into their music system to experiment and generate new music. It has been seen that generative music can let a musician understand their repertoire well and express better musicality in themselves. [2]

In general, we feel any machine learning algorithm trying to work can be improved using our approach. This helped musicians to configure their compositions to involve generative music. [2] [5] Having our code open-sourced will also help in building and customizing the application approach without being stuck behind a paywall-ed closed code.

Here are a few keywords which have referred or used in the paper as well as our systems to make it easy for musicians and computer scientists work with our system:

1.1 MIDI

Known as Musical Instrument Digital Interface. It is a communication protocol to connect various electrical musical instruments to a computer for playing, editing and recording. A single MIDI link through a MIDI cable can carry up to sixteen channels of information, each of which can be routed to a separate device or instrument. MIDI carries event messages; data that specify the instructions for music, including a note's notation, pitch, velocity (which is heard typically as loudness or softness of volume); vibrato; panning to the right or left of stereo; and clock signals (which set tempo). When a musician plays a MIDI instrument, all of the key presses, button presses, knob turns and slider changes are converted into MIDI data. From a technical perspective, this is as good as sending a number in the desired MIDI format to make or process the music. [1]

We can refer to figure 2, this diagram decomposes a grand piano into MIDI notes, and gets a better understanding on what messages can be expected from MIDI when a specific key is pressed on the instrument. This changes from different instruments and hence having an idea of how instruments interact with MIDI is becomes important.

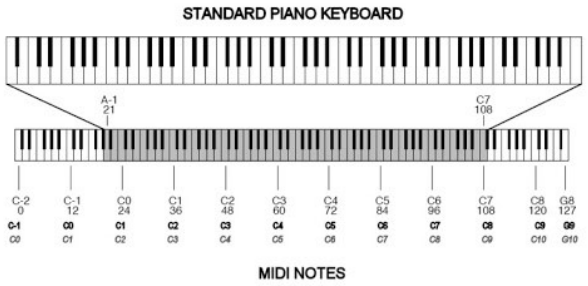

Figure 2: A standard piano keyboard decomposed into MIDI notes.[12]

1.2 MIDI Controllers

MIDI controllers are designed to just generate the MIDI data over a device which musicians are familiar with, and transmit it to some other device using a MIDI cable which can play them as sound information. MIDI controllers do not create or produce musical sounds by themselves.There are different type of MIDI Controllers like Keyboard Controllers, Wind Controllers, Drum and percussion controllers, etc. These are available in various sizes and instrumentation types that range from 25-key, 2-octave models, to full-sized 88-key instruments with real-time controllers such as sliders, knobs

and wheels to control pitch, modulation, sustain and other functions.

Figure 3: A MIDI drum-pad which takes in how hard a pad was hit and converted them to drums like cymbals, snares, toms and hats.

1.3 OSC

Also known as Open Sound Control, is a protocol for networking sound synthesizers, computers, and other multimedia devices for purposes such as musical performance or show control. [7] Few of the important applications of OSC are real-time sound and media processing environments, programming languages, etc and interaction between them. OSC uses UDP/IP protocols to transmit messages using open-ended, dynamic, URI-style symbolic naming scheme. It allows the customization of dynamic messages and let them send over ports and also over internet.[8] Since OSC works over the internet as well, our application though demonstrated on a local machine, can be configured to work on an architecture where MIDI inputs come in remotely via Internet from all over the world.

Here we have used OSC to communicate between generated MIDI messages between python (which continuously tracks for inputs from keyboard) and Max/MSP, which controls the GUI and also plays the MIDI messages as sound.

1.4 Max/MSP

Max/MSP is a visual programming language for the specialized needs of artists, educators, and researchers working with audio, visual media, and physical computing. We have used the latest version Max8 as the Graphical User Interface(GUI) to visualize the layout of MIDI controller. It acts as a client in our software and interacts with the python server via OSC. MAX has a lot of inbuilt functions which let process an audio wav file as well as MIDI prcessing, which we have not touched upon in our approach. We feel a power user can use Max at the full extent combined with out approach. Their drag and drop approach to programming makes it easy for musicians to work and develop low code for their applications. From figure 4, we can see the MIDI keyboard controller layout on the Max8 software.

2 RELATED WORK

Previously, different software has been designed to replace the MIDI keyboard controllers with QWERTY Keyboard. for example, Studio

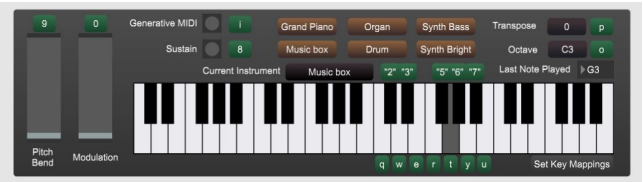

Figure 4: Max/MSP: A GUI to visualize the MIDI Keyboard controller.

One is a software developed by PreSonus. It is a digital audio workstation application used to create, record, mix and master music with extra features. It allows user to switch the MIDI keyboard controller by single handed,2-octaves modeled QWERTY keyboard. But it does not provide two-handed layout on the QWERTY keyboard which restricts the user from that genuine experience.[3]

There are other alternatives of Max8, one of them is Ableton Live. It is also digital audio workstation, designed to be an instrument for live playing and also have editing, composing, arranging and other features.

3 ARCHITECTURE

In figure 5, we can see the overview architecture of our software. There are three major parts of this whole software i.e. Max8 as Graphical user interface, OSC Module and midiKeyboard module. Max8 provides a GUI for MIDI controller incorporating the status of different features like pitch bend, modulation, last key played, transpose, octave, etc. It also shows the assigned key-notes mapping, and is editable for assigning new keys.

The midiKeyboard module is a python class and responsible for mapping the keys of the QWERTY keyboard to different notes and features. It is also responsible for figuring out what key is pressed or released, and sending back the required response for that key. It is the back end of the software.

OSC module is our python class and acts as the middleware which establishes an asynchronous channel for the communication between the Max8 and the midiKeyboard module. It uses the communication protocols and a dynamic, URI based messages to send the information from one end to another. Those message addresses have been designed or named for specific tasks which makes communication system easier. These messaging nomenclature can be found in our documentation available with the code.

3.1 User Interface Module

We are using Max8 or Max or MSP as the graphical user interface in our system. It receives the processed keyboard inputs and produces the corresponding MIDI sound output. It also offers the control to convert the output sound in different type of instrumental sound like Grand Piano, Organ, Synthesizer Bass, Music Box, Drum and Synthesizer Bright. It also provides the feedback of the current instrument playing and the last note played. It keeps showing the user about the different configuration settings of the whole MIDI controller. for example, it shows the key mapping of the QWERTY keyboard and the corresponding 12-notes and the features like Transpose, Octave, Sustain, Pitch, Modulation and Generative MIDI. Once the user is satisfied with the key assignments, they can set the

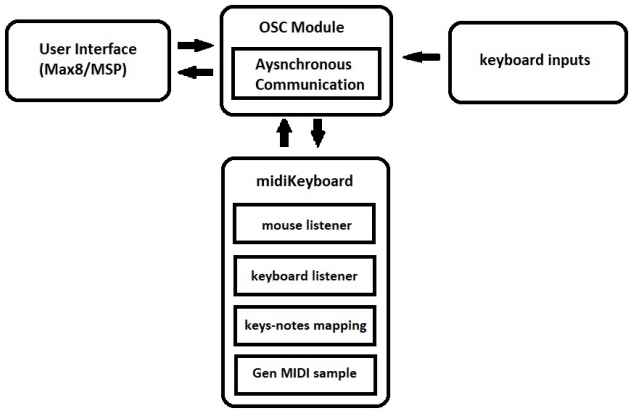

Figure 5: Architecture Overview of the software

configuration by pressing the the *Set Key Mappings* button which will send this information back to the server.

3.2 OSC Module

Open Source Control module is responsible for the communication between the keyboard inputs, User interface Max8 and the midiKeyboard module. It creates an asynchronous communication channel which starts listening between Max8 and the midiKeyboard module. When an action is performed like pressing a key or releasing a key, the input message in the form of address get transmitted using UDP/IP protocol through the asynchronous communication channel, as we can see in table 1. This module is written in python script using pythonOSC library.

The communicateOSC class is responsible for all of the functions of this module. It takes an IP address, port in and port out values to establish the asynchronous communication channel.

3.3 midiKeyboard Module

This module is responsible for taking the inputs from the keyboard or Max8 module through OSC module and return back the response in MIDI format. After OSC module reads the input from the asynchronous communication channel, it calls the mapped functions of midiKeyboard module. The functions have applications like updating the Octaves and Transpose, returning the note value in MIDI format,etc. It also takes movement of mouse or touch pad and keyboard activity to update the pitch bend value and modulation value by using the mouse and keyboard listener of python libraries. The module is also written in python script.

The keys and notes mapping is declared in this module. So, when a key is pressed or released, corresponding notes value get processed with other arithmetic operations and returned back in MIDI format. It is also responsible for playing randomly stored generative MIDI samples.

4 APPROACH

We purchased an AKM320 by MIDIPLUS MIDI controller (shown in Figure 6) to understand how MIDI inputs can be expected from a device and how they can be recreated over keyboards. It is a 32

Table 1: Dispatcher Mapping of Addresses and functions

Address	Function	Explanation
"/inputs/key_mapping/*"	set_keyMappings	set key-mapping
"/inputs/key_on/gen_MIDI"	set_genMIDI	MIDI samples
"/inputs/key_on"	keyOn	key pressed
"/inputs/key_off"	keyOff	key released

key keyboard, which spans over two and a half octaves, and has buttons to change octave up or down, along with transposition of the keyboard keys. Along with this, it has two knobs, one for Pitch Bend and other for Modulation, which can be turned up or down for their effects to affect the music being played. The know for Pitch bend once left automatically resets back to the original position. It also has an input to connect a sustain pad, but we did not have one, so we have implemented sustain similar how how a piano sustain works. [11]

Figure 6: AKM320 by MIDIPLUS, a MIDI controller we used for reference while building our library

4.1 Class *communicateOSC*

In this class, we have used python library python-OSC to establish the asynchronous communication channel between the front end(GUI on Max8) and back end(python server which processes the corresponding MIDI output using class *midiKeyboard*). It communicates with a specific input/output port and an IP,

4.1.1 Dispatcher. OSC receives the messages from front end in URI-style with unique addresses like *"/inputs/key_mapping/*"* and arguments.The Dispatcher maps and calls these addresses with corresponding functions of different classes. We can see the mapping in Table 1.

4.1.2 keyOn: Note on. When any key is pressed this function is called. It takes two parameters, the address and array of arguments. The argument args[0] contains ASCII value of the key pressed. The other argument, args[1] keeps the track of modifier keys. It calls the function of class *midiKeyboard* and returns the output list of length two where the second element of the list is 127, indicating the key was pressed i.e. Note on.

4.1.3 keyOff: Note off. When the key is released, this function is called with ASCII value of key in args[0]. Similarly, this function also takes two parameters, the address and the array of arguments. It calls the same function of class *midiKeyboard* and returns the output list of length two but here the second element of the list is 0, indicating the key was released i.e. Note off.

4.1.4 set_genMIDI: play samples. It calls a function from class *midiKeyboard*, which is responsible for playing the generated samples of MIDI stored in the directory /gen_midi_samples.

4.2 Class *midiKeyboard*

This class performs various MIDI effects and operations according to the key pressed on the keyboard. It performs different calculations, modifies different real-time controllers, sets and updates the key mappings and returns back the output in MIDI format to the *communicateOSC* class.

4.2.1 MIDI data format. When a key is pressed on the MIDI controller, it sends two values to the Max8 i.e. [X, Note status]. The value X varies from 0 to 127 and the Note status takes value 127, when the key is pressed or 0, when the key is released. Rest of the controller functions like Transpose, Octave, etc and real-time controller functions like Pitch Bend and Modulation, modifies the value X, according to their functionality. The value X constitutes of note played from the 12 keys[0-12], the Transpose factor and the Octave factor.

4.2.2 Static Controllers. There are some controllers or functionalities available like Transpose update, Octave update, etc, which user need to set or update to modify the coming notes by a logical factor. These functionalities are modified in this class with different functions when the specific key is pressed. For example, the Transpose factor varies with ±1 and the Octave factor varies with ±12.

4.2.3 Real-time Controllers. There are other functionalities which are modified during the note is being played. These methods are called real-time controllers or functionalities. For example, Pitch Bend and Modulation. To handle these factors, we have used touchpad or mouse movements to control them. As the corresponding mapped key is pressed, our mouse and keyboard listeners starts listening the coordinates of the mouse pointer and starts returning the values on some logical change in the coordinates. As the key is released, the listeners stops listening. We have used *pyautogui* and *pynput* libraries of python to handle this. That's how we manage the real-time controllers.

By running our communication python script of the software, it establishes the two way asynchronous communication between the python server and the client Max8. As user provide any input by pressing a key or touch-pad movement, it takes the input to *communicateOSC* class using the OSC protocol and corresponding functions are called of class *midiKeyboard*.The *midiKeyboard* class identifies the key and performs different MIDI effects and operations on the received input. Once the effects are applied on the note, the function sends back the MIDI output to the *communicateOSC* class, which returns it back to the Max8 client and corresponding MIDI sound is generated.

5 EVALUATION

5.1 Heuristic Evaluation

Considering the front end part of our software project, we decided to perform Qualitative Evaluation on the system i.e. Heuristic Evaluation. While applying the Heuristic evaluation, we have realised that our software's user interface violates few heuristic principles. We can observe our software user interface layout before applying heuristic evaluation from figure 7.

5.1.1 H2-1: Visibility of system status. User is not able to see the status of the key pressed. User cannot deduce the type of instrument playing by looking the user interface.

5.1.2 H2-6: Recognition rather than recall. User has to remember all the mapping of the keys, which is prone to wrong key press. It also leads to user's memory overload.

5.1.3 H2-10: Help and documentation. New user will face drastic issues in understanding the software and its all the functions. Many functionalities are hidden to user.

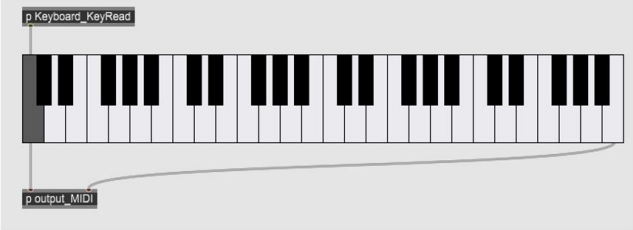

Figure 7: Before Heuristic Evaluation

5.1.4 After Heuristics. From figure 8, we can see the improved user interface for our software after rectifying the violations.

5.1.5 H2-1: Visibility of system status. We have added the continuous feedback section in our User Interface, which tells the user about the current key pressed and also about the current instrument selected. We have also added the bars to show the current intensity of Pitch bend and Modulation.

5.1.6 H2-6: Recognition rather than recall. A user can identify the key mapping by looking the User Interface. And adjust them according to his comfort.

5.1.7 H2-10: Help and documentation. We have added the help documentation for our software. So, user can understand all the features of the software and can modify them according to his required application.

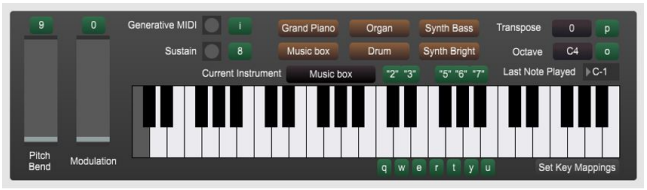

Figure 8: After Heuristic Evaluation

5.2 User Evaluation

We circulated out application among some users, while some were adept with music knowledge (including MIDI as well as some musicians who don't use MIDI), others were normal users who have

used a piano before but do not understand music theory. Our test group consisted gave us a perspective about the application from a power user's perspective as well as a lay man. We also had it tested with a few musicians can code and understand how MIDI and OSC work with each other.

Users without any music theory knowledge found the application as a fun way to interact with a musical instruments without having to make any investment. They also said this would motivate them to learn and understand music as their investment was minimal but gave a good idea on how a synthesizer works. Their favorite feature was being able to switch between different musical instruments. The generative MIDI also given them a good understanding of learning a piano.

On the other hand, Musicians did not prefer this method, mainly because performers are already adept to an actual keyboard and it's keys and found it a bit difficult to adjust to clunky keyboard keys. Though, musicians who have worked with MIDI before really appreciated this technique, as this intrigued them on the machine learning applications. They still preferred MIDI keyboards to compose and perform most of their music, but the idea of having a keyboard add value as an extra instrument or sampler was appreciated. The best feedback we received came from a Music Theory teacher, who felt this was a cheap and intuitive tool for new students to understand, and listen to Music theory concepts, and they are planning to use this in their class next semester.

6 FUTURE WORK

We have developed a modular code base for our software. Currently we are using it to play two-handed layout of keyboard on QWERTY keyboard, but one can modify the controllers to be a wind controller, Drum and percussion controllers as we have discussed before. Our most important offering is the OSC library which would make it easy to configure any application working on OSC to communicate with our system to make new musical interfaces.

We are using a midi python library which also has features to play the midi via python itself. This would be nice integration to the system than sending out messages to other MIDI player. With integration of other machine learning techniques for MIDI, it would also be a nice addition to play music with time signatures. [9] this could also help build a small library for MIDI and ML interaction.

7 CONCLUSION

Our approach lets users use keyboards as a musical instruments, while also giving freedom to make new musical instruments from any devices which can connect and communicate with our python server. The heuristic evaluations have helped us understand musical interfaces better for further development, while the user evaluations have provided with new ideas to develop, and help students and teacher learn music remotely without expenses. The most important takeaway would be the importance of MIDI in machine learning techniques and how our application can be integrated easily into such systems.

REFERENCES

[1] The MIDI Association. 2000. MIDI History:Chapter 6-MIDI Is Born 1980-1983. (2000).

[2] Andrew R. Brown. 2009. Generative Music Editorial. *Contemporary Music Review* (February 2009), 1–4.

[3] PreSonus Commercial Divison. [n.d.]. *Studio One by PreSonus.* https://www.presonus.com/products/studio-one/

[4] F. Vico J.D. Fernandez. 2013. AI Methods in Algorithmic Composition: A Comprehensive Survey. *Journal of Artificial Intelligence Research* 48 (November 2013).

[5] Nathan Keil, David A. Dahlbom, Jeremy Stewart, Matthew Goodheart, Curtis Bahn, Mary Simoni, Michael Perrone, and Jonas Braasch. 2019. Polyphonic pitch perception in rooms using deep learning networks with data rendered in auditory virtual environments. *The Journal of the Acoustical Society of America* 145, 3 (2019), 1784–1784. https://doi.org/10.1121/1.5101527 arXiv:https://doi.org/10.1121/1.5101527

[6] Ann McCutchan. 1999. The Muse That Sings: Composers Speak about the Creative Process. *New York: Oxford University Press* 72 (1999), 67–68.

[7] opensoundcontrol.org. [n.d.]. *Introduction to OSC.* http://opensoundcontrol.org/introduction-osc

[8] opensoundcontrol.org. [n.d.]. *The Open Sound Control 1.0 Specification.* http://opensoundcontrol.org/spec-1_0

[9] Stony Brook Univeristy Perry Goldstein. 2019. *Rudiments of Music: A Concise Guide to Music Theory.* Kendall Hunt Publishing.

[10] dave smith and chet wood. 1981. the 'usi', or universal synthesizer interface. *journal of the audio engineering society* (october 1981).

[11] Barbara Tillmann, W. Jay Dowling, Philippe Lalitte, Paul Molin, Katrin Schulze, Bénédicte Poulin-Charronnat, Daniele Schoen, and Emmanuel Bigand. 2013. Influence of Expressive Versus Mechanical Musical Performance on Short-term Memory for Musical Excerpts. *Music Perception: An Interdisciplinary Journal* 30, 4 (2013), 419–425. http://www.jstor.org/stable/10.1525/mp.2013.30.4.419

[12] Wikipedia. [n.d.]. *MIDI.* https://en.wikipedia.org/wiki/MIDI

A LINKS TO OUR WORK

A.1 GitHub

All our code is open-sourced and available here. It currently runs with the MAX GUI for one handed keyboard layout and also a two-handed keyboard layout. It also includes the documentation for for understanding the OSC messages for creating a custom application over it.

A.2 Demo Video

A demo video has been created showcasing how the Max GUI looks while interacting with keyboard. This helps to understand the different features we have added to the system as well as how to use them. The video can be found here.