Interactive Systems Portfolio

Albert Madrenys Planas

Maynooth University
MU617A[A] — Interactive Systems (2023-24)

1 Introduction

This document presents the final portfolio documentation for the Interactive Systems module at Maynooth University. Throughout this writing, the various projects of the portfolio will be explained. Comprising three distinct elements, each one will be explored in terms of its intended interaction, underlying motivations, hardware and software integration, and a final assessment with potential avenues for improvement.

While this document will feature photographs highlighting key content, it is important to note that each project within the portfolio possesses its own gallery containing images, a demonstration video, and access to the source code.

2 Electrobodhrán

The bodhrán is a drum commonly played in traditional Irish music. It is typically made of goatskin, with a frame that usually ranges from 35 to 45 cm in diameter. As a one-sided drum, the bodhrán is placed vertically on the player's leg. One hand strikes the skin side with a beater, while the other hand touches the back of the skin through the open side of the instrument to control the pitch and timbre.

By capturing the vibrations of the skin, the original sound can be enhanced using digital signal processing. This can be done by attaching a piezoelectric sensor to the skin with vinyl tape.

2.1 Hardware

The electrobodhrán is designed to be an instrument that does not require the use of a computer, which is why an embedded platform is necessary. Furthermore, we need minimal latency to ensure that the percussive and electronic sounds have as little delay as possible.

Daisy Seed is the chosen option. Thanks to the DaisyDuino library, DSP programs can be easily written and compiled for the chip.

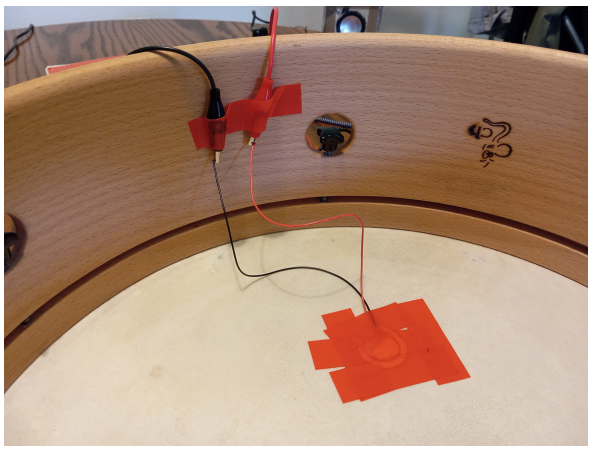

Fig. 1. Piezoelectric sensor attached to the bodhrán skin.

One of the key elements in creating the electrobodhrán is capturing the vibrations of the skin. This can be achieved with a piezoelectric sensor. The sensor consists of a metal disk that, when carefully placed and fixed on the instrument's skin, generates voltage in response to vibrations caused by drum hits. This can be done by attaching the sensor to the skin using vinyl tape (see fig. 1). The use of tape is quite intensive, as using less tape would make the sensor loose, causing unwanted high-end sounds due to movement between the skin and the sensor when hitting the drum with the beater.

An additional switch has been introduced to make turning the effect on and off very easy. The configuration also includes an amplifier a speaker and a battery. Fig. 2 shows the final configuration of the circuit, including the use of resistors for safety, as demonstrated in the slides for the Interactive Systems module.

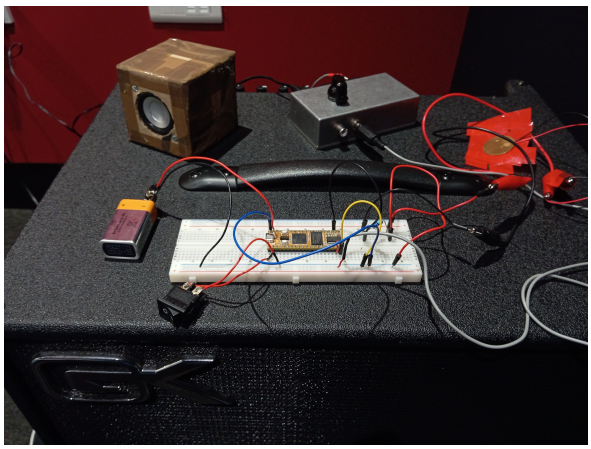

Fig. 2. Electrobodhrán final circuit configuration.

2.2 Software

The software used for creating the Daisy program is the DaisyDuino library within the Arduino IDE. We have only two inputs to consider: the piezoelectric sensor and the switch. For the sensor, we will read the input in an analog way to get a gradient. This value will then be mapped to normalized values (0 to 1), using one of the examples shown in this module. The switch can be read as a simple binary input.

The Daisy library offers a variety of DSP effects. The initial source is a sawtooth wave oscillator. The oscillator is then sent to a Moog ladder filter, followed by a balance process, and finally to a reverb. The input from the piezoelectric sensor is used to change the amplitude of the oscillator and the resonance of the Moog ladder filter. This allows both the amplitude and the timbre of the sound to change with the hits on the drum, creating an interesting sound. The balance process is used to recover the RMS level of the original oscillator output before being processed by the filter. Finally, the reverb is used to extend the length of the sound after the initial hit, functioning similarly to an envelope by making each hit on the skin last a bit longer with a satisfying spectral decay.

Additionally, the input of the piezoelectric component have been smoothed at control-rate by using a portamento filter. This has the negative effect of adding delay, but it helps with the input data not being as jittery. Furthermore, the amplitude of the oscillator is multiplied by the gate that is controlled by the switch, so we can turn on and off the sound in a simple way. A diagram of the whole processing pipeline is shown in Figure 3.

2.3 Results and Improvements

The final result is interesting and satisfactory. Making the vibrations of the skin modify both the output level and the resonance of the filter, thus changing the timbre, results in a pleasing and interesting sound. The input is quite responsive, and thanks to the reverberation, the output envelope lasts longer than the initial drum hit, since the input signal caused by the vibrations of the skin stops

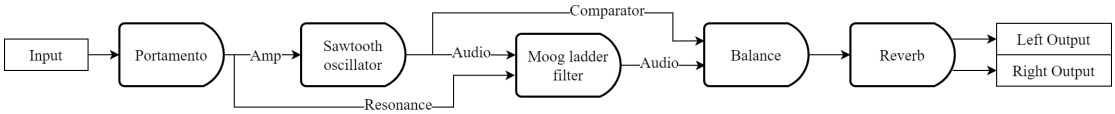

Fig. 3. Electrobodhran DSP diagram.

very quickly after the hit. The final audio almost sounds like a digital brass instrument, such as a trombone. It is a very organic and pleasing sound.

One less favorable element in the final result is the noticeable delay perceived between the physical skin hit and the response from Daisy. This delay is caused by the portamento and Moog ladder filters and is intrinsic to these solutions. Unfortunately, we cannot do much to solve this issue without changing the timbre of the sound by altering the Moog filter. While less delay would be welcome, it is not significant enough to be overly annoying.

3 Connect 4ton

Connect 4 is a classic tabletop game where two players choose a color and take turns dropping colored tokens into a suspended grid. It is a highly popular and well-known game, easily accessible for anyone to pick up and strategize. Because of its familiarity, it is an ideal candidate for adding a unique twist. Introducing an audio interactive system that responds to the evolving match could offer an interesting and enjoyable experience.

This project is named Connect 4ton. As will be explore in the next section, light will be used to manipulate the sound. Connect 4ton is a play on words, combining the original game with "photon," a subatomic particle found in light.

3.1 Hardware: Failed Attempt

The primary concept behind this aspect of the portfolio involves interrupting the passage of light through the grid when a token is dropped into a hole. This interruption can be detected using a light sensor. An Arduino chip connected to multiple light sensors can then send MIDI messages to a computer. This sensors have been implemented with safety resistors as demonstrated in the slides for the Interactive Systems module.

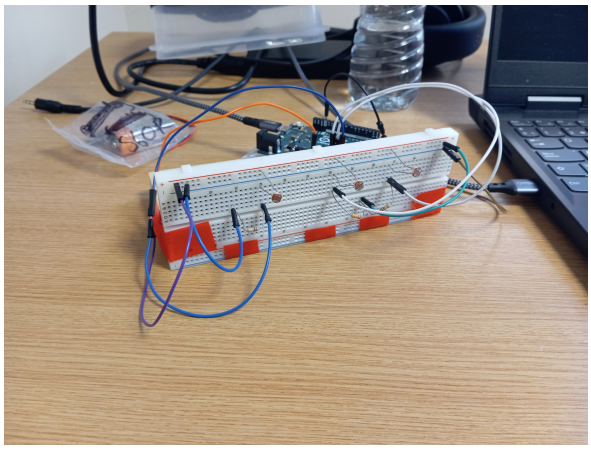

Fig. 4. Light sensors without the cardboard tubes in the failed attempt.

One of the main challenges encountered from the project's inception was the placement of the light sensors to ensure they react accurately while maintaining a clean setup. Initially, the idea was to

Fig. 5. Failed attempt setup with the cardboard tubes.

isolate the light reaching each sensor using sliced cardboard tubes. The setup involved positioning a breadboard facing the grid directly, with the light sensors connected to it and each sensor isolated by a cardboard tube extending from the base of the breadboard to the grid. Each tube was designed to cover four holes arranged in a 2x2 pattern. Fig. 4 and 5 show the light sensors positioning in relation to the game grid, with and without the cardboard tubes.

On the opposite side of the grid, a directional light source was positioned to provide more consistent lighting conditions across different environments. The light source consisted of a headlamp secured with a food plastic container.

Despite the promising start, the changes in light perceived by the sensors were too inconsistent, and the setup proved messy. It was challenging to position the tubes correctly. Consequently, an alternative setup had to be conceived.

3.2 Hardware: Final Version

The previous attempt faced two main issues: a messy setup and inconsistent results. These problems have been addressed by using only one light sensor for each hole. This approach simplifies the detection process to binary changes: either there is light passing through or a token obstructs the light. To achieve this and clean up the setup, the light sensors need to be placed closer to the hole, eliminating the need for tubes.

This is accomplished by using duct tape. Duct tape is used to cover all the holes on one side of the grid, allowing for small gaps to pass the two light sensor cables through. The light sensor cables have been extended to reach a breadboard. Fig. 6 illustrates the light sensors being secured in the holes using duct tape.

This adjustment significantly simplifies the setup and improves the precision of the light sensors. However, it comes with the drawback of requiring a larger number of light sensors to cover the entire grid, as one sensor is now needed per hole. In this proof of concept, only five light sensors will be used, placed on the lowest row. This decision is due to the limited availability of light sensors and the impracticality of scaling up the setup. Additionally, the version of Arduino used does not support a large number of analog sensors.

3.3 Software: Input Detection

The input detection will be primarily handled by Arduino. Arduino will continuously monitor the inputs from light sensors and convert this data into MIDI control messages, which will be sent to the computer functioning as a MIDI device. Messages will only be sent when changes in the light are detected. Arduino is capable of sending MIDI messages thanks to the MIDIUSB.h library, and each light sensor will be mapped to a different control channel. The Arduino code is a highly modified version of one of the examples shown at this module.

These MIDI messages will be received and processed by Csound on the computer. Csound will maintain arrays with five elements, with each element representing a MIDI channel and a light sensor.

Fig. 6. Light sensors in the final version.

These arrays will contain the current MIDI value, the previous MIDI value from the preceding Csound call at the control rate, and a boolean to indicate the state of each event. Csound will trigger a change in the sound when three conditions are met: it is not the first call, so the previous MIDI value is known, the event has not been triggered yet, and the difference between the current and previous MIDI value exceeds 25 (with 127 being the maximum MIDI value). The value of 25 has been determined empirically and consistently yields satisfactory results in this specific setup.

It is worth noting the use of sharp changes in light to trigger events, rather than a simpler threshold comparison. This approach allows for event triggering across a wider range of lighting environments. Whether the game is played in a dimly lit or brightly lit room, what matters is the sudden change in light.

3.4 Software: Audio Processing

The audio processing part of Csound will consist of two instruments: one instrument will play continuously throughout the duration and will modify the output sound based on the state of the Connect 4 game, while the other will produce a short and loud sound scheduled when a token is placed in a hole, alerting players that the system has responded to their actions in the game.

The first instrument will continuously play a tune to provide a background of engaging music, avoiding repetitive sounds that could become boring or annoying. The chosen piece of music is the Tetris Theme from the game with the same name. The version selected is the Game Boy rendition, because of its retro aesthetic, nostalgia, catchiness, and fun factor. This tune is easily recognizable, making changes to it easily noticeable. Interestingly, the Tetris Theme originates from a folk Russian song called *Korobeiniki*.

The tune is altered in three main ways: changing the tempo, adjusting the pitch, and adding a flanger effect that modulates the pitch periodically. Two light sensors will control the tempo, two will adjust the pitch, and one will introduce the flanger effect. In Csound, the tempo of the tune can be modified by adjusting one of the parameters of the *diskin* opcode, which reads the sound file. However, changing the tempo also alters the speed. To address this, when adjusting the pitch using the *pvscale* opcode, the pitch value needs to be divided by the speed value to separate the two characteristics.

The second instrument will be a short and loud sound triggered by a new token placed on the grid. It will utilize the *marimba* opcode, which implements a marimba physical model. The output of the marimba will be followed by a strong reverb effect to extend the duration of the instrument with a pleasing decay. This effect also somewhat mimics the sound of the token hitting the grid.

3.5 Results and Improvements

The final result is highly satisfactory. The chosen theme is catchy and engaging, and the acceleration of tempo and pitch as the match evolves adds a sense of tension, which was the intended effect.

Additionally, the extra instrument triggered when placing a token provides strong and satisfactory feedback to the players. The input reading is consistent and works effectively.

While the current setup serves as a successful proof of concept, there is room for improvement in the future. For instance, the current setup only accommodates five holes, yet the grid has many more. Additional effects such as distortion, filters, modulation, and noise could be incorporated. Tokens with different transparencies could be used to differentiate between players, introducing more nuanced interactions with the grid. Tracking the state of the game would allow for the triggering of a third instrument to signal the end of the match and stop the music. Furthermore, experimenting with layered music, gradually introducing more instruments as the game progresses, could enhance the experience.

Overall, the final result is positive and engaging, and the concept has significant potential for expansion.

4 SuperOSCollider

SuperOSCollider is a demonstration of the capabilities of SuperCollider, OSC messages, and the use of AI processing to modify musical elements. This demo is not intended to be artistic but will focus more on the technology and structural principles of designing a complex system. This system will includes different processes communicating with each other using OSC messages. As a result, the audio design will be kept simple to make it easier to detect and understand what is happening behind the scenes.

4.1 FM Synth and Patterns

The SuperCollider audio processing section will comprise two elements. The first is a very short, beepy sound that will be repeated with variations using a pattern. This sound will be generated with a sine oscillator and shaped by a percussive envelope. The parameters that can be modified externally include the oscillator's frequency, amplitude, sound duration, and panning.

To ensure this short sound plays immediately after the previous one ends, SuperCollider offers a very useful pattern called *Pbind*. This pattern automatically instantiates new instances of a given synth, with its arguments modified according to other patterns. *Pbind* will be used to generate short instances of the synth explained earlier, each time with different random frequency, duration, and pan values.

The other synth that will be used is one that will not be enveloped and will play throughout the entire execution of the system. It is a basic frequency modulation synth, allowing for changes in amplitude, carrier and modulator frequencies, and the index of modulation. The output is stereo, with panning modulated by a sine oscillator.

4.2 OSC Recievers

Some of the synth and pattern arguments described earlier will be modified upon receiving OSC messages. These OSC messages will be received and managed by two *OSCFunc* instances. SuperCollider will use two OSC receivers because it will receive messages from two ports: 57120 and 57121. Only the latter needs to be explicitly opened, as the former is the intrinsic SuperCollider language port.

When defining the functions to map the message data into the pattern elements, a problem was encountered. The characteristics of the *Pbind* pattern could not be changed once the pattern was instantiated. To solve this, *PatternProxy* was used in some of the *Pbind* bindings instead of setting a fixed number or another pattern. *PatternProxy* acts as a placeholder for a pattern, allowing the proxy to be changed elsewhere in the code in execution time, such as within the *OSCFunc*."

4.3 Game Controller

Up until now, the focus has been on the SuperCollider side of the project, which involves receiving messages and modifying the audio accordingly. Now its time to discuss the elements responsible for sending these messages.

The first element is a game controller, specifically a wireless Xbox 360 game controller developed and manufactured by Microsoft for their home game console. To enable the computer to recognize the

device, a wireless adapter and its respective drivers were installed. Due to the age of this controller and the fact that it was developed at a time when the Xbox and Windows system, despite both being owned by Microsoft, were not as well integrated, an online tutorial was needed to configure everything correctly, including the device port. The tutorial can be found at: `https://www.youtube.com/watch?v=yFdajgerBMc`.

Fig. 7. Wireless Xbox 360 Controller and their corresponding USB adapter.

To use the Xbox 360 controller for sending OSC messages, a script for the Processing sketchbook software is required. Originally developed by Peter Lager and Rebecca Fiebrink to map gamepad inputs to a pair of animated eyes, the script was later modified by Scott H. Hawley to send OSC messages. The author of this portfolio further modified the script to send normalized values from 0 to 1, instead of data related to the eyes positions on the screen.

The messages sent by the game controller will be received directly by SuperCollider through one of the two ports it is listening to.

The controller has multiple buttons, but only the two joysticks and the left trigger will be used in this demo. The left joystick will control frequency parameters of the SuperCollider *Pbind* pattern. The horizontal axis will set the minimum frequency at which the pattern can generate random beeps, while the vertical axis will control the range of the interval. For example, a very low vertical axis value will generate beeps with frequencies close to the minimum set by the horizontal axis, whereas a very high vertical axis value will allow a wide range of frequencies.

The right joystick and the trigger will control the FM synth. The vertical axis of the joystick will control the frequency of the modulator, and the trigger will switch between two states of the modulation index. A more comprehensive diagram showing the input mapping of the controller is shown in fig. 8.

Fig. 8. Xbox 360 Controller mapping in SuperOSCollider.

4.4 Face Detection

Another element of the system is face tracking through a camera connected to the computer. Using a script for Processing, OSC messages can be sent. The script detects a face and sends its horizontal and vertical position, as well as the width, which is directly related to the distance of the face from the camera.

This script was originally developed by Daniel Shiffman, then modified by Jordi Tost to make it compatible with the OpenCV library, and further modified by Rebecca Fiebrink to send the three OSC values mentioned.

In this case, the OSC messages will not be sent directly to SuperCollider. Instead, they will be sent to Wekinator through port 6448.

4.5 Wekinator

Wekinator is an open-source software created by Rebecca Fiebrink that enables the use of machine learning to build computer vision and listening systems. It is a simple and user-friendly AI tool designed for artists and musicians, capable of sending OSC messages. Its output messages are determined by the OSC messages it receives as inputs, combined with prior training.

In this case, Wekinator will receive three inputs corresponding to the outputs of the face tracking application. It will have only one output, which will be received by SuperCollider through the second OSC listener (recall that the first OSC listener receives messages from the game controller).

This Wekinator output will be used in SuperCollider to control the amplitude of both audio generators: the pattern and the FM synth. Since the Wekinator output message will be normalized from 0 to 1, this value can be easily mapped to the amplitude of the pattern, updating the *PatternProxy*. The amplitude of the synth will be determined by subtracting the amplitude of the pattern from one. This means that when the amplitude of the pattern is high, the amplitude of the synth will be low, and vice versa. This is known as the complement.

Due to the inherent unpredictability of AI, the author wanted this unpredictability to have some relevance in training the AI, while still maintaining some level of control over the generated output. To achieve this balance, a middle-ground approach was chosen: the AI was trained so that when a face is detected on the left, the output value will be low, and when it is on the right, the output value will be very high. For the remaining inputs, the AI was trained with random values, allowing for some uncertainty in how the AI will react to the vertical position of the face and its proximity to the camera. Although there is not full control over the AI's behavior, it is understood that a face on the left will tend to increase the synth amplitude, while a face on the right will tend to increase the pattern amplitude. This middle-ground approach demonstrates both the power of AI and the ability to influence its behavior through training.

4.6 System Structure

Up until now, the discussion has focused on each part of the puzzle individually. However, to gain a full understanding of the design of SuperOSCollider, it is necessary to examine the bigger picture. Fig. 9 offers a comprehensive map of how the entire system communicates between its various components, detailing the messages sent, the ports they originate from, and how they are utilized.

4.7 Results and Improvements

The final result is satisfactory. SuperCollider effectively receives messages from multiple sources and modifies audio elements of various natures, such as patterns and synths, in a satisfactory manner. By incorporating elements like face detection through a camera, machine learning, and a game controller, it serves as a compelling demonstration of what can be achieved with OSC messages and the power of coordinating different applications using the same protocol.

While the current setup is effective, there are areas for improvement to make SuperOSCollider even more compelling. One area is the simplicity of the audio generation units. While they utilize clever techniques like *Pbind*, incorporating more complex audio generation methods could enhance the system's capabilities.

Additionally, the use of AI could be expanded further. AI has the potential to extract valuable data from complex and difficult-to-identify elements. For instance, it could be used to identify objects

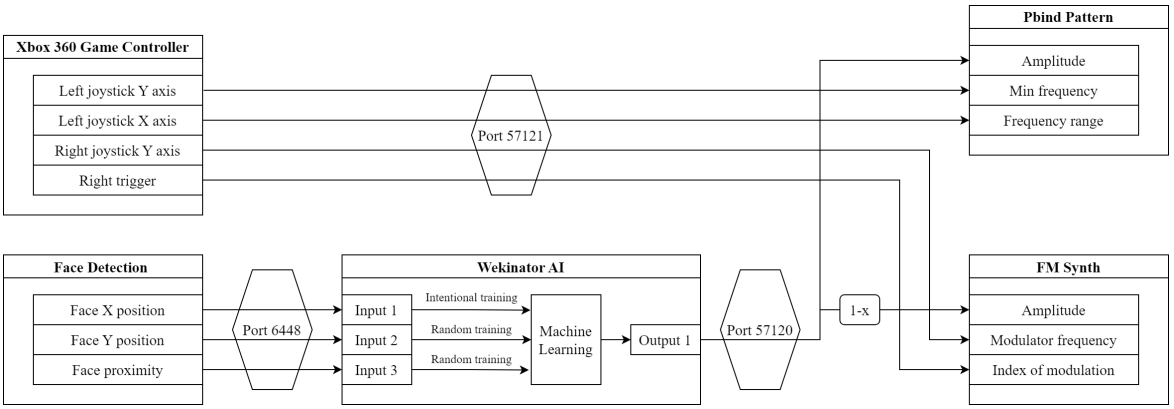

Fig. 9. Diagram of the SuperOSCollider messaging system.

in an image, providing more meaningful insights than simply detecting face positions. While the current demonstration is effective, making use of the full power of Wekinator could unlock even more potential.

5 Conclusions

In conclusion, this portfolio comprises three projects: the electrobodhrán, Connect 4ton, and SuperOSCollider. The electrobodhrán explores the integration of technology into traditional instruments, enhancing sound through the use of sensors and embedded systems. Connect 4ton introduces an interactive twist to a classic game, utilizing light and sound to enhance the gameplay experience. Lastly, SuperOSCollider delves into the realm of audio messaging protocols, employing tools like SuperCollider and machine learning to manipulate audio in interesting ways

Together, these projects demonstrate a wide range of techniques learned throughout the module. This portfolio showcases the endless possibilities at the intersection of human interaction and sound technology, encouraging new avenues of expression and innovation.

References

1. Electrosmith, DaisySP Documentation, `https://electro-smith.github.io/DaisySP/index.html` [accessed 15 May 2024]
2. Iain McCurdy, Punk Electronics, `https://mu-main-mdl-euwest1.s3.eu-west-1.amazonaws.com/0f/08/0f0890d4a0570edd4ec213bcb490fd1bbd746db6?response-content-disposition=inline%3B%20filename%3D%22MU617A%20Punk%20Electronics.pdf%22&response-content-type=application%2Fpdf&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAWRN6GJFL2EMJIMOU%2F20240608%2Feu-west-1%2Fs3%2Faws4_request&X-Amz-Date=20240608T005806Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21594&X-Amz-Signature=9ff8291e1e75fd290258c02dee29ff549873fd76a6bd92852baa5dc733f10bae` [accessed 4 June 2024]
3. Iain McCurdy, USB MIDI on Arduino, `https://moodle.maynoothuniversity.ie/mod/folder/view.php?id=692655` [accessed 4 June 2024]
4. Lazzarini, V., Yi, S., ffitch, J., Heintz, J., Brandtsegg, Ø., McCurdy, I.: Csound: A Sound and Music Computing System. Springer (2016)
5. Wikipedia contributors, Korobeiniki, `https://en.wikipedia.org/wiki/Korobeiniki` [accessed 6 June 2024]
6. Wikimedia contributors, Xbox 360 Wireless Controller White, `https://commons.wikimedia.org/wiki/File:Xbox-360-Wireless-Controller-White.png` [accessed 7 June 2024]
7. Filipe Cruz, How To: Fix Xbox 360 wireless receiver drivers, `https://commons.wikimedia.org/wiki/File:Xbox-360-Wireless-Controller-White.png` [accessed 13 September 2023]
8. Processing Foundation, Reference — Processing.org, `https://processing.org/reference` [accessed 28 May 2024]
9. Rebecca Fiebrink, Wekinator — Software for real-time, interactive machine learning, `http://www.wekinator.org/` [accessed 28 May 2024]

10. Simon Katan, Example input and output in single SuperCollider patch, https://www.doc.gold.ac.uk/~mas01rf/WekinatorDownloads/wekinator_examples/inputs/SuperCollider/wekiExample_2Inputs_5ContinuousOutputs_ReceivesOnPort57120.scd [accessed 29 May 2024]
11. Scott H. Hawley, Gamepad_Eyes_OSC.pde, https://gist.github.com/drscotthawley/dd74db91dd2f8ec3c810a87d4d26b576 [accessed 29 May 2024]
12. Rebecca Fiebrink, Face Tracking, http://www.doc.gold.ac.uk/~mas01rf/WekinatorDownloads/wekinator_examples/all_source_zips/VideoInput_FaceDetection_Processing_3Inputs.zip [accessed 29 May 2024]
13. SuperCollider Contributors, Pbind — SuperCollider 3.12.2 Help, https://doc.sccode.org/Classes/Pbind.html [accessed 5 June 2024]
14. SuperCollider Contributors, PatternProxy — SuperCollider 3.12.2 Help, https://doc.sccode.org/Classes/PatternProxy.html [accessed 5 June 2024]