A modular tool for automatic Soundpainting query recognition and music generation in Max/MSP

Arthur Parmentier, Ken Déguernel, Constance Frei

A MODULAR TOOL FOR AUTOMATIC SOUNDPAINTING QUERY RECOGNITION AND MUSIC GENERATION IN MAX/MSP

Arthur PARMENTIER (arthur.parmentier@epfl.ch)[1], **Ken DÉGUERNEL** (ken.deguernel@epfl.ch)[1], and **Constance FREI** (constance.frei@unil.ch)[2]

[1]**Ecole Polytechnique Fédérale de Lausanne**, Switzerland
[2]**Université de Lausanne**, Switzerland

ABSTRACT

This paper presents a modular software designed for automatic recognition of Soundpainting query. Soundpainting is a musical practice and a sign language used for real-time composition with an orchestra. A series of signs is gestured by the soundpainter creating a "sentence" describing a musical idea that they want the orchestra to perform. We propose an open-source tool, for Max/MSP, able to perform every task for a computer to be part of a Soundpainting scene: motion tracking, gesture recognition, query parsing and music generation. This tool is created in a modular way so that it can be easily modified to fit the needs of the user, for instance, changing the type of inputs for the motion tracking or adapting the Soundpainting grammar. We describe the global architecture, the different components of this tool, and the currently implemented methods for each of these components. We then show examples of use for this tool, from the learning of a new sign to a performance with several virtual instruments.

1. INTRODUCTION

Soundpainting (SP) is a sign language developed in 1974 by the New York composer and saxophonist Walter Thompson for real-time composition with his orchestra [1]. Although the language was originally used for composing with musicians, it has extended to multiple artistic disciplines such as dance, theatre or visual arts and is now used worldwide by a variety of artists in diverse contexts and configurations. SP is not originally a language designed for working with electronic devices and computers. It is often reported that their use in SP is made difficult by the high reactivity requested by the soundpainter to the set of performers that forms the orchestra. However, digital tools have been used since the second half of the 20th century in new forms of compositional processes and aesthetics of music [2–4]. Modern methods of Human-Machine Interaction for learning and performing in real-time [5, 6] enables the exploration of new artistic materials with new dynamics of creativity [7, 8].

Automatic gesture recognition is an important topic of Human-Machine Interaction aiming at interpreting human gestures using cameras and/or motion sensors, providing 2D or 3D information. Machine learning techniques are used in order to recognise and analyse the gestures, usually focusing either on full-body gesture recognition or hand gesture recognition. Automatic gesture recognition is used for many applications, such as medical diagnosis [9], surgery analysis [10, 11], emotion recognition [12], sign language recognition [13], etc.

Automatic gesture recognition has also been used in many applications for computer music [14]. For instance, Fernández et al. [15] use Kinect camera, controller gloves and sound mapping for audiovisual performances inspired by percussionists movement; Dalmazzo et al. [16] use a Myo armband to detect arms and fingers movement for a pedagogical tool checking the bowing and fingering of a violinist; Cavdir et al. [17] present an interface with hand gesture recognition and haptic feedback with movement inspired by sign language to create musical performances that can be experienced both by people who are hard of hearing and by those who are not; Chandran et al. [18] uses a camera and openCV to detect facial expressions to control virtual instruments; Van Nort et al. [19, 20] proposes a gesture database, in order to analyse and map sonic and kinetic actions happening during performances according to their gestural meaning, etc.

In the last few years, automatic gesture recognition for SP started gathering some steam. Pellegrini et al. [21] proposed a proof of concept with the recognition of a few SP gestures using Kinect input classified with Hidden Markov Models. However, at the time, they only discussed the possibility of SP annotations and other use cases of the recognition system theoretically without proposing a concrete prototype for it. More recently, Gómez Jáuregui et al. [22] used SP gesture recognition for the generation of electronic music. They use a Kinect and decision tree classifier to recognise 9 different gestures. However, the gestures are considered individually and not as part of a syntactic phrase. SP gesture recognition has also been used outside of the scope of music in order to control a swarm of drones [23].

In this paper, we propose a new tool for Soundpainting Query Recognition (SPQR) with two main aspects in mind. First, the software we propose is complete, ie. it performs every necessary task for SPQR (cf. Figure 2) motion tracking, real-time classification of gesture, query parsing and music generation, and can be used by anyone with a com-

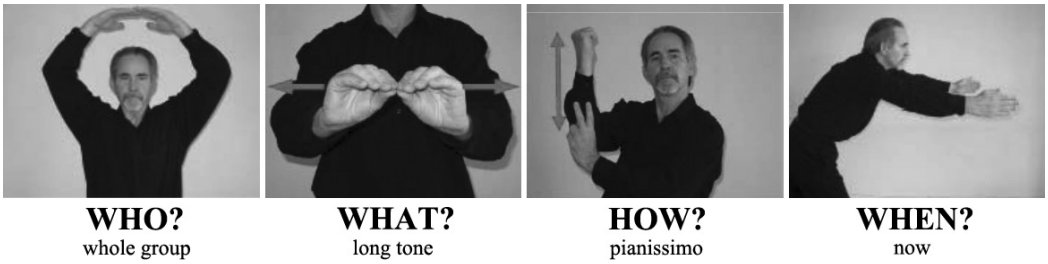

Figure 1. Example of a Soundpainting query using the Who? What? How? When? syntax (illustration from [1]).

puter and a webcam. Second, the software is built in a modular way: every component is independent, enabling an advanced user to easily change them, for instance modifying the motion censors or replacing the classification method.

The rest of the paper is structured as follows. In section 2, we introduce and discuss the key elements of SP related to our project. Then, in section 3, we describe the architecture and the different components of our SPQR software. Then in section 4, we discuss some aspects of performance and provide demonstrations for the use of the software. Finally, in section 5 we discuss the use of this tool in the SP sphere and propose some perspectives for the future of this project.

2. ELEMENTS OF SOUNDPAINTING

Soundpainting is a sign language used by a (or in some cases, several) composer, called soundpainter, as a real-time communication system with performers of an orchestra. The language is organised in "modes" of interaction between the soundpainters and the performers, each one having its own grammar and dictionary of signs. The most common mode ("default mode") uses a syntax that is commonly simplified as "Who" –indicating which performers should respond to the request–, "What" –what content should they perform–, "How" –the shape of the content– and "When" –when the request should be executed–. Fig. 1 shows an example of a SP phrase using this syntax. In response, the performers do not sign but rather provide artistic contents that build up the composition. The soundpainter can then sign in reaction or not to the performers' responses shaping the artistic material while it is being played to the audience. In the rest of the article, we call "query" a SP phrase. This is a bit of a play on word between the computer science definition that the software has to deal with, and the human language definition, since in the philosophy of SP, the soundpainter only requests things from performers without any guarantee on the performer's responses. *"The Soundpainter composes with what happens in the moment, whether expected or not."*

Most of the basic signs of SP involve only a combination of dynamic gesture, e.g. arms and hands movements and of static postures made by the soundpainter, facing the performers. For instance, in the SP workbook [1] the sign for "Whole-group" (see Fig. 1) is described as the following posture: *"Hold both arms over your head creating a circle with fingertips barely touching"*, and the sign for long-tone, is described as the following gesture: *"Holding your hands a little out in front of your body, pinch the thumb and index finger of both hand together and pull them apart along a horizontal plane"*. However, it is to note that the time and space span for each gesture and posture may vary.

Among the most common SP signs are the "multi-disciplinary" signs. These signs such as "minimalism" or "long tone" can be interpreted in several disciplines (music, dance, acting, visual arts...). Even though the concept they refer to usually come from one discipline only, their interpretation in SP has been extended to the other ones by creating relevant analogies for each one. It results in a rich and powerful language that stimulates the creativity of the composer and performers using it.

The full grammar of SP is not yet fully described and involves many other syntactic categories that are learned during practice, therefore, for the rest of this article, we are only considering the "default" mode for which the syntax is established and described in details in [1].

3. SOFTWARE ARCHITECTURE

SPQR is built with several independent layers, emulating the different structural levels of SP, from the creation of signs to the parsing of its grammar and music generation. We have identified five layers:

- **input management**, computing features from different input systems (webcam, gloves, etc.),
- **sign and dictionary managemen**t, defining and storing the SP gestures and postures,
- **real-time classification,** performing SP sign recognition,
- **parsing and request-forming automata**, creating a query out of a series of signs,
- **orchestra simulation**, performing music according to the query.

An illustration of these components and their interactions is given in Figure 2.

Each layer was conceived as a specific function that the user should easily be able to identify and interpret. Inside each layer are different processes and objects that the user interacts superficially with from the interface of the program.

At the interface level, all layers are implemented inside Max/MSP. The user can see the whole patcher in

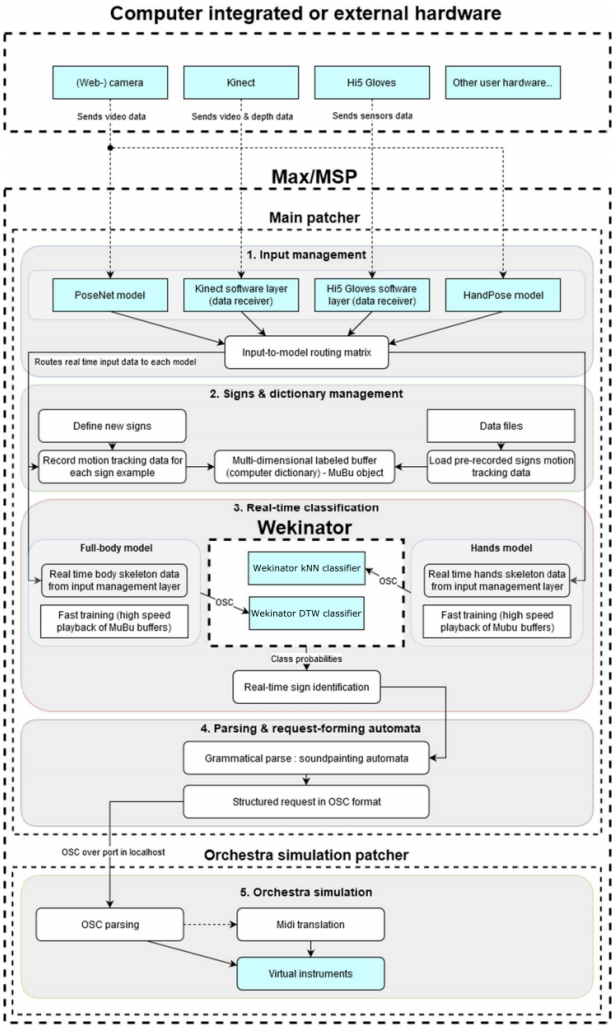

Figure 2. Schematic overview of the different modules and their interactions in SPQR. Blue components come integrally or partially from other works while white components have been designed during this project.

the main window and is also able to access specific functionalities of each layer by using tabs. At the processing level, Max/MSP itself has three different threads that it uses for processing the data passing through its compiled objects. For these threads, Max guarantees the synchronicity/ordering of events. However, Max also interprets Node.js code that is processed in external threads asynchronously to Max internal threads.

In this part, we describe the different components implemented in the current version of this software, but each module could be easily replaced by an experienced user.

3.1 Input management

The role of the motion tracking layer is to compute a set of motion features from the movement of the user. There exist several motion tracking systems with different technologies that can be used to model the human body from the position of characteristic points. These points can then be transformed into features of a classifier to identify gestural signs. In SP, some body parts such as the hands are used much more frequently to sign than others, therefore they require more precise tracking than the latter to classify amongst the signs. However, all motion tracking systems available have a finite range of operation, i.e. they can only track motion at a certain scale (just like the human cognition system). We integrated two models to respectively track signs from the full-body and from the hands.

For these scales, we use respectively PoseNet [24] –an open-source computer-vision model that can be used to estimate the pose of a person in an image or video by estimating where key body joints are in 2D space– and HandPose [25] –it's equivalent for modelling the hand–. For PoseNet, the available features are the position coordinate for ears, eyes, shoulder, elbows, wrists, hips, knees and ankles with the nose considered as a fixed point. For HandPose, the available features are the position coordinate of every finger tip and knuckles with the centre of the palm considered as a fixed point. For our experiments, we only used the positions of elbows and wrists for PoseNet, and the positions of every fingertip as well as the second knuckle for the index and middles fingers.

Their performances on modern CPUs and GPUs allow them to run in real-time using a webcam or alternative low-latency video input devices, making them usable by anyone without needing special hardware. We use the TensorFlow version of these tools which are ported into Max/MSP using a Node.js server. Therefore, the patch provides a simple user interface (cf. Figure 3). PoseNet and HandPose allow the user to choose different models and internal parameters that will affect its performance, such as the architecture of the model (MobileNet or ResNet), the input resolution of the video, the size or the depths of the model, etc. With these settings, the user can adapt the model to its hardware easily to get the best performance.

The different inputs can be attributed to the different models using the input manager. We opted for a modular approach with a design that allows the user to add its own inputs and models in from the Max patcher. Because inputs have different data rates and dimensions, it is in general not possible to route more than one input to a single model. If two inputs are compatible (typically with similar data rates) the user simply can create a new input and merge the two original ones.

3.2 Signs and dictionary management

Once the inputs and models are defined in the program, the user can start recording signs and building its sign dictionaries for each model. Users can either create their own signs or use pre-recorded signs, that would have been created or recorded by other users. A sign can be defined by two properties: its name or its category, in analogy with the syntactic model of SP (identifier, content, etc.) These properties are sufficient to allow the program to parse the sign, i.e. to construct a meaningful request from the temporal flow of signs.

In order for the sign to effectively be identified, two steps are required after the sign has been defined: record training examples and program the virtual instrument itself to interpret the sign. While the recording of training exam-

Figure 3. User interface for PoseNet in the Max/MSP patch. The left part shows the input from a webcam with the PoseNet key body joints. The right part shows the different coordinate for each key body joints. The user can select here which features to take into consideration in the system.

ples is an automated process that simply involves pushing one button, the programming of the virtual instrument or device that the sign should control is outside the scope of the program.

The user can choose to either define one sign at a time and record one or several training examples for it, then saving the training data and adding another sign, or directly define a list of signs and recording all of them in the same session. Each recording takes place in a different buffer of the Multiple Buffer (MuBu) objects (one Mubu object per model) [26] and each active input data is saved into a different track. The recordings can then be saved to build a dictionary of MuBus corresponding to different signs. It is also possible to load pre-recorded signs by dragging and dropping data files in the dedicated zone.

3.3 Real-time classification

For the system to be able to "recognise" the signs, we decided to focus on lightweight, interpretable models that can be trained fast and identify the signs that are performed in real-time. In our case, the identification is a simple classification process, in which we ask the classifier to predict the "class" of the motion sequence performed among a set of classes that have been previously learned by the model –the SP signs.

After a few experiments with the gesture follower from MuBu based on Hierarchical Hidden Markov Models, we decided to use the external software Wekinator [27] performing better by offering a very efficient Dynamic Time Warping implementation based on the FastDTW library [28] with additional improvements for real-time performance and several internal parameters for its DTW model. Wekinator is integrated into Max through communication with the Open Sound Control (OSC) protocol. Although the user must launch Wekinator separately and perform basic operations on its GUI, the most important parts of Wekinator can be controlled remotely via OSC, allowing Max to automatise certain operations, such as its training process.

Once the model is running, Max receives in real-time the set of DTW distances from the real-time sequence to each recorded sign sequence. By finding the minimal value in that set and comparing this value to a confidence threshold, we can identify when a sign is being performed in real-time.

3.4 Parsing and request-forming automata

From the models introduced in the previous section, we can recognise individual signs, forming a sequence in time, just like words form a phrase in oral languages. The next step is therefore to implement the grammar of SP with a parsing mechanism that would then allow us to create requests or commands to each device that acts as an individual performer in the system. To that purpose, we implemented an automaton inside Max using Node.js. A visualisation of the automaton is provided inside the patch for a clear description of the SP grammar and also for more experienced users to have direct feedback on their grammatical implementation, for instance when adapting the grammar to their own purpose (cf Figure 4).

The automaton that we implemented corresponds to the grammar for the "default mode" of SP explained in section 2 (the "Who–What–How–When" syntax). The automaton allows us to represent such a sequence and understand each sign as a particular function in the query. It also recognises "wrong" signs and allows for feedback for the user about the correctness of their query. Details about the grammar of SP and its implementation can be found in [29].

Once parsed, the query is structured by collecting each sign during the state transitions and assembling them into several hierarchical objects. The query is then converted in OSC format that can be used to communicate with a virtual performer.

3.5 Orchestra simulation

From the OSC commands created by the automaton, there is an unlimited panel of tools and ways to create an orchestra of virtual performers, using DAWs, OSSIA [30], etc. In order to provide a complete usable tool in Max/MSP, we decided to use *bach* [31]. *bach* implements both classical music notation and proportional music notation, with support for accidentals of arbitrary resolution, rhythmic trees and grace notes, polymetric notation, MusicXML, MIDI files, etc. The orchestra simulation with *bach* is implemented in a different patcher than the recognition tool and receives the commands from the automata with OSC. We decided to build fixed contents into the `bach.roll` and `bach.score` objects that allow the user to place notes into a score, just like any score editing program. These objects are then connected to virtual instruments through Midi.

In more general approaches, the recognition program can be connected to a variety of devices that interpret OSC. In our case, the companion program works as follows: each instrument is simulated by a vst that can receive midi notes. The midi notes are sent from the reading of `bach.score` and `bach.roll` objects that each corresponds to a given

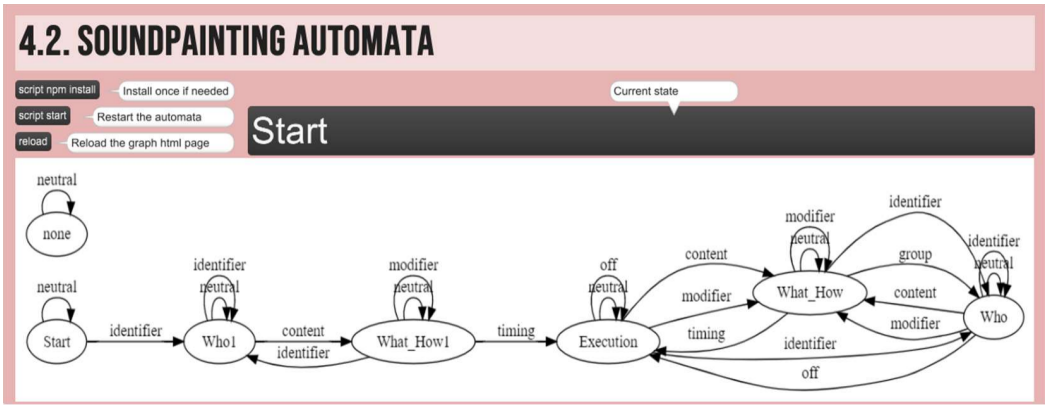

Figure 4. Finite-state automaton representing the grammar for the "default mode" of Soundpainting. The graph shows the different states of the automaton and the possible transitions between them.

musical content of a given instrument and that the user can write and edit just like a normal score. When an OSC command is received from SPQR, a simple regexp routing allow the command to be sent to a given type of musical content (for instance "long tone") of a given instrument (for instance "percussions"). If several long tones are implemented for a single instrument, one is chosen randomly when the command "start" is received. By default, a medium tempo and volume is chosen. But if the commands "tempo" or "volume" followed by a value are received, the `bach.roll` and `bach.score` objects are set to change their values. Therefore, the tempi and volumes of the different instruments can be adjusted in real-time. Once a "stop" command is received, the values of tempo and volume are set back to their default values. This implementation allows us to predefined a set of possible musical elements with Bach objects that are then chosen randomly in the play. Future approaches could rather rely on probabilistic generation models and models of interaction between the different instruments [32].

4. PERFORMANCE AND DEMONSTRATION

The Soundpainting Query Recognition tool is fully available under GPLv3 licence on its Github repository [1] . Detailed informations about the installation process of this SPQR tool can be found on the Github repository as well as in [29].

After installation, even though the number of layers is high, the end-user only manipulates high-level elements on the program: buttons and text elements. Being released by default with webcam inputs, it is easy for the user to run it on a personal computer without any setup on the input part. As for now, we only provide the companion patcher that runs the basic orchestra simulation with *bach* objects. However, the setup of the devices that are controlled at the input and output of SPQR can be modified and added by an experimented users.

The tool was tested using a relatively high-end home computer with 16GB of RAM, an i7-8750H processor and an Nvidia 1060 GTX as a dedicated GPU. For PoseNet, we used the parameters for ResNet50 at quantbytes = 1 and input size = 350, achieving 15 FPS. For Wekinator, we set the default max sequence length to 30, achieving again 15 FPS, which has proven to be more than enough for SPQR. It is to note that PoseNet is very sensitive to light conditions and contrast. Although it is difficult to describe the perfect environment in those terms, the user should pay attention to both camera settings (luminosity or ISO, contrast, saturation profiles, etc) and the position of the lights in the configuration to ensure that the models can work with the best accuracy. A rather uniform background will probably result in good recognition when in high contrast with clothes and skin colour. For our tests, we used two cameras (computer webcam and phone camera) running in parallel. The computer camera is then routed to the full-body model with PoseNet and the phone camera to the model for the hands with HandPose. In general, the choice of the inputs (number, type...) is left to the user and will greatly influence the performance of the recognition. In our case, we worked with a computer camera for simplicity of usage. However, one could reach higher performance and better gesture discrimination with gloves or full body suit 3D tracking systems for instance.

We also released a series of videos, showing the different aspects of this work accessible at `http://deguernel.discordia.fr/spqr/`. The first video explains the gesture and query recognition task using PoseNet, Wekinator DTW, and the SP automaton. We demonstrate the training part of the model using several gestures that are repeated very few times. We compare then the performance of MuBu's gesture follower using Hierarchical Hidden Markov Models and of Wekinator, using DTW, for the classification task, and finally, show how the gesture recognition is used to navigate the SP automaton to form a SP query. The second video shows how to use two cameras (here a webcam and a phone camera) to model both full-body and hand gestures simultaneously with PoseNet and HandPose. Finally, the third video shows a full demonstration of SPQR for music generation using *bach* as a virtual orchestra.

[1] `https://github.com/arthur-parmentier/soundpainting-signs-gestures-recognition`

5. DISCUSSION

From the point of view of the soundpainter, the recognition is an exploratory and creative instrument that can also push for new ways of signing and thinking SP as a direct relation to the instrument (or device) itself. Learning this sign language with a band is not offered to everyone and most orchestras are interested in practising with experimented soundpainters only. In practice, most soundpainters first learn the language as performers before signing themselves, such that they have already internalised most of the language and compositional propositions before endorsing the "role" of soundpainter as a composer.

Feedback is one of the most important aspects of learning. One will for instance learn how to correct himself from errors when he will be able to perceive those as such and understand what the cause of the error is. Even though in the SP design, nothing is considered a mistake on the side of the performer who interprets the sign, the soundpainter can make syntactic mistakes. In a learning tool implementing a grammar, it is important to let the user know why a particular sentence is wrong or what did the program recognize that is not intended. In our SPQR tool, the user can already receive these types of feedback from the automaton, which on top of outputting its actual state and how the request is created also provides error messages that indicate if an unexpected or illegal sign has been observed. For instance, if the user signs "rest of the group, long tone, whole group, minimalism, play", the automaton will output an error message when receiving "whole group, minimalism" , stating that "rest of the group" has already been requested as a content previously in the sentence and that the request of a new content is ambiguous.

Another type of feedback is the ability to hear the contents that are produced by the program and how they react to the different requests, even when the user is making mistakes or the program does not recognize the intended signs. Feedback is one form of incentive to explore more of the tool and learn with it. They can be classified into two categories: external or internal to the program. Some artists are already interested in using the recognition tool in their own installations: they have external incentives. However, some users may not be familiar with digital tools nor with SP and will not take to create something of their own if they are not pushed to it by the program's mechanics that form its internal incentives. Typically, games are good examples of programs with a lot of internal incentives. They have gamified features, such as a score, elements of competition or collaboration, rewards, etc., which push the user to explore more of the game and performing better at it. For future work, there is probably some potential to be explored with SPQR for the implementation of gamified elements or feedback through interactive designs and visualisations. Moreover, we also plan to reach testers to make a more collaborative and musician-oriented development plan and to assess the performances of SPQR more quantitatively. Finally, we also plan to release a compiled version of the system that will simply run as an executable file, simplifying the installation process a lot.

Acknowledgments

A. Parmentier and K. Déguernel share first authorship for this article. This work was made possible thanks to Sarah Kenderdine, the eM+ Lab and the DHLAB at EPFL. We would also like to thank Walter Thompson for his support and the interesting conversations.

6. REFERENCES

[1] W. Thompson, *Soundpainting: the art of live composition. Workbook 1*, 2006.

[2] Guy E. Garnett, "The Aesthetics of Interactive Computer Music," *Computer Music Journal*, vol. 25, no. 1, pp. 21–33, 2001.

[3] D. Keislar, "A historical view of computer music technology," in *The Oxford Handbook of Computer Music*, R. Dean, Ed. Oxford: Oxford University Press, 2011, ch. 2, pp. 11–43.

[4] J.-C. Risset, "Computer music: why," *Songes, Passages, Computer Suite from the Little Boy, Sud. Mainz: Wergo*, vol. 2050, 2013.

[5] D. Herremans, C.-H. Chuan, and E. Chew, "A functional taxonomy of music generation systems," *ACM Computing Surveys*, vol. 50, no. 5, pp. 1–30, 2017.

[6] J. Nika, K. Déguernel, A. Chemla-Romeu-Santos, E. Vincent, and G. Assayag, "DYCI2 agents: merging the "free", "reactive", and "scenario-based" music generation paradigms," in *Proceedings of the International Computer Music Conference*, 2017.

[7] E. A. Edmonds, "Human computer interaction, experience and art," in *Interactive experience in the digital age:evaluating new art practice*, L. Candy and S. Ferguson, Eds. London: Springer, 2014, ch. 2, pp. 11–23.

[8] P. Esling and N. Devis, "Creativity in the era of artificial intelligence," in *Proceedings of the Journées d'Informatique Musicale*, 2020.

[9] F. Negin, P. Rodriguez, M. Koperski, A. Kerboua, J. Gonzàlez, J. Bourgeois, E. Chapoulie, P. Robert, and F. Bremond, "PRAXIS: Towards automatic cognitive assessment using gesture recognition," *Expert Systems with Applications*, vol. 106, pp. 21–35, 2018.

[10] I. Funke, S. Bodenstedt, F. Oehme, F. von Bechtolsheim, J. Weitz, and S. Speidel, "Using 3D convolutional neural networks to learn spatio-temporal features for automatic surgical gesture recognition in video," in *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2019, pp. 467–475.

[11] X. Gao, Y. Jin, Q. Dou, and P.-A. Heng, "Automatic gesture recognition in robot-assisted surgery with reinforcement learning and tree search," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2020, pp. 8440–8446.

[12] F. Noroozi, D. Kaminska, C. Corneanu, T. Sapinski, S. Escalera, and G. Anbarjafari, "Survey on emotional body gesture recognition," *IEEE Transactions on Affective Computing*, 2018.

[13] M. M. Islam, S. Siddiqua, and J. Afnan, "Real time hand gesture recognition using different algorithms based on american sign language," in *Proceedings of the IEEE International Conference on Imaging, Vision Pattern Recognition*, 2017.

[14] A. R. Jensenius and M. J. Lyons, Eds., *A NIME reader : Fifteen years of New Interfaces for Musical Expression*. Springer, Cham, 2017.

[15] J.-M. Fernández, T. Köppel, G. Lorieux, A. Vert, and P. Spiesser, "GeKiPe, a gesture-based interface for audiovisual performance," in *Proceedings of the New Interfaces for Musical Expression Conference*, 2017, pp. 450–455.

[16] D. Dalmazzo and R. Ramírez, "Air violin: a machine learning approach to fingering gesture recognition," in *Proceedings of the International Workshop on Multimodal Interaction for Education*, 2017, pp. 63–66.

[17] D. Cavdir and G. Wang, "Felt Sound: a shared musical experience for the deaf and hard of hearing," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2020.

[18] D. Chandran and G. Wang, "InterFACE: new faces for musical expression," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2018.

[19] D. V. Nort, I. Jarvis, and M. Palumbo, "Towards a mappable database of emergent gestural meaning," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2016.

[20] D. V. Nort, I. Jarvis, and K. Maraj, "Performing gesture and time via an emergent database," in *Proceedings of the 2020 International Conference on Movement and Computing*, 2020.

[21] T. Pellegrini, P. Guyot, B. Angles, C. Mollaret, and C. Mangou, "Towards Soundpainting gesture recognition," in *Proceedings of the Audio Mostly: A Conference on Interaction With Sound*, 2014.

[22] D. A. G. Jáuregui, I. Dongo, and N. Couture, "Automatic recognition of Soundpainting for the generation of electronic music sounds," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2019.

[23] N. Couture, S. Bottecchia, S. Chaumette, M. Cecconello, J. Rekalde, and M. Desainte-Catherine, "Using the Soundpainting language to fly a swarm of drones," in *Advances in Intelligent Systems and Computing*, J. Chen, Ed. Springer, Cham, 2017, pp. 39–51.

[24] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. P. Murphy, "Towards accurate multi-person pose Estimation in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4903–4911.

[25] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "MediaPipe Hands: On-device real-time hand tracking," in *Proceedings of the Workshop on Computer Vision for Augmented and Virtual Reality*, 2020.

[26] N. Schnell, D. Schwarz, J. Larralde, and R. Borghesi, "PiPo, A plugin interface for afferent data stream processing modules," in *Proceedings of the International Symposium on Music Information Retrieval*, 2017.

[27] R. Fiebrink and P. R. Cook, "The Wekinator: A system for real-time, interactive machine learning in music," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2010.

[28] S. Salvador and P. Chan, "FastDTW: Toward accurate Dynamic Time Warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 70–80, 2004.

[29] A. Parmentier, "Soundpainting Language Recognition," Master's thesis, EPFL, 2020.

[30] J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier, and M. Desainte-Catherine, "OSSIA: Towards a unified interface for scoring time and interaction," in *Proceedings of the International Conference on Technologies for Music Notation and Representation*, 2015.

[31] A. Agostini, D. Ghisi, and J.-L. Giavitto, "Programming in style with bach (extended version)," in *Proceedings of the International Symposium on Computer Music Multidisciplinary Research*, 2019.

[32] K. Déguernel, E. Vincent, and G. Assayag, "Probabilistic factor oracles for multidimensional machine improvisation," *Computer Music Journal*, vol. 42, no. 2, 2018.