FP7-ICT-2013-C FET-Future Emerging Technologies-618067

SkAT-VG: Sketching Audio Technologies using Vocalizations and Gestures

D6.6.2
Front-end application for interactive sound prototyping

First Author	Stefano Baldan
Responsible Partner	IUAV
Status-Version:	Final-1.0
Date:	June 28, 2016
EC Distribution:	Consortium
Project Number:	618067
Project Title:	Sketching Audio Technologies using Vocalizations and Gestures

Title of Deliverable:	Front-end application for interactive sound prototyping
Date of delivery to the EC:	30/06/2016

Workpackage responsible for the Deliverable	WP6
Editor(s):	Stefano Baldan, Stefano Delle Monache, Davide Rocchesso
Contributor(s):	Stefano Baldan, Stefano Delle Monache, Davide Rocchesso, Hélène Lachambre, Frédéric Bevilacqua
Reviewer(s):	
Approved by:	All Partners

Abstract	The current deliverable presents the final results of WP6
Keyword List:	Imitation-driven, sound design, application, SDT, SkAT-Studio, miMic, MIMES

The SkAT-VG Consortium consists of the following entities:

#	Participant Name	Short-Name	Role	Country
1	Università Iuav di Venezia	IUAV	Co-ordinator	Italy
2	Institut de Recherche et de Coordination Acoustique/Musique	IRCAM	Contractor	France
3	Kungliga Tekniska Högskolan	KTH	Contractor	Sweden
4	Genesis SA	GENESIS	Contractor	France

Document Revision History

Version	Date	Description	Author
Skeleton	05/31/2016	Start	SteB
Draft	06/07/2016	Section 3: SkAT-Studio	Hélène
Draft	06/27/2016	Section 5: Mimes	Fred

Table of Contents

Index of Figures

List of Acronyms and Abbreviations

DoW Description of Work

EC European Commission

PM Person Months

WP Work Package

1 Executive summary

This document presents the final achievements of work package 6 (WP6), which covers the sound synthesis part of the SkAT-VG project and the control of sound generation through vocalizations and gestures, to achieve imitation-driven sonic sketching and prototyping. In particular, WP6 is responsible for the development of:

1. Sound synthesis tools able to simulate the sound sources represented by imitations;
2. High-level control strategies and layers to combine the sound models and to manipulate their parameters using vocalization and gestures;
3. Software architectures and user interface modules (UI) to facilitate the activity of the sound designer.

The first two goals of WP6 are effectively achieved by the interoperation of two software prototypes, developed during the first two years of the SkAT-VG project and already presented in deliverable D6.6.1: The *Sound Design Toolkit* (SDT), a collection of physically-informed interactive sound models simulating a wide variety of acoustic phenomena, and *SkAT-Studio*, a modular framework developed in Max which allows to define custom audio processing workflows.

The synthesis models available in the SDT follow a perceptually-founded taxonomy of everyday sound events, simulating the physical behavior of a wide variety of basic mechanical interactions (between solids, liquids and gasses) and machines (electric motors and combustion engines). These sound models are organized in *timbral families*, collections of synthesizers and relative parameter spaces, designed to accurately represent the sound categories recognizable by vocal imitations, as defined and experimentally assessed by psychoacoustics experiments in WP4.

Imitation-driven control of the synthesizers is made possible by the analysis algorithms recently included in the SDT, specifically designed to extract salient features and timbral descriptors from the audio signal of a digitized vocal imitation. Signal analysis routines and sound synthesizers are embedded as modules in the SkAT-Studio framework, which is used to build audio processing workflows that integrate acquisition and analysis of the input, generation and playback of the output, and mapping between the audio descriptors extracted by the vocal signal and the synthesis parameters exposed by the various timbral families.

This report will focus on the WP6 activities done in the last year of the SkAT-VG project, and will present the progress towards the achievement of the third objective, namely the development of a front-end application for interactive sound prototyping using vocalizations and gestures.

In this respect, the SkAT-VG tools described above have been used to develop another application called miMic, an augmented microphone for vocal and gestural sonic sketching. The prototype is composed of a physical device, based on a modified microphone with embedded inertial sensors and buttons, and a software part developed in Max. In addition to the imitation-driven real-time control of sound synthesis parameters, already presented in D6.6.1, this prototype also integrates part of the work of WP5 implementing an automatic classifier of vocal imitations, and the consequent automatic selection of the most appropriate sound

synthesis models to render the imitated sound. MiMic can be considered as a first attempt of front-end application enabling sound designers to sketch and prototype sounds with their voice and gestures.

Another prototype, called MIMES, has been developed with a greater attention towards gestural control. This application offers a complementary approach to sound design with respect to miMic, allowing the use of vocal input not only as a control signal, but also as raw sound material directly available for manipulation by means of a series of sensorized objects and tangibles interfaces. Integration plans and guidelines to merge the possibilities offered by miMic with those offered by MIMES have already been devised, and will guide the development of the SkAT-VG tools during the remaining months of the project.

2 The Sound Design Toolkit

2.1 Changes since D6.6.1

Since the release deployed for deliverable D6.6.1, the Sound Design Toolkit was subjected to a number of minor changes and updates, presented in the following paragraphs.

2.1.1 Basic solid interactions

The discretization scheme used to compute all kinds of basic mechanical interactions between solids changed since the last release of the Sound Design Toolkit. Since their first implementation, legacy SDT resonators used the bilinear transform, a method with the nice property of always leading to stable discretizations whenever the continuous-time system is also stable. The bilinear transform, however, suffers from two main disadvantages:

- The induction of *frequency warping*, a non-linear relation between the continuous and discrete time frequency axes that needs to be compensated;
- The presence of *delay-free loops* in updating the state of the system, due to the implicit nature of the approximation scheme, that require specific methods to be computed.

To solve the delay-free loops in basic solid interactions, legacy SDT interactors used a non-linear solver based on Newton-Rhapson approximation. This technique adds a considerable amount of conceptual and computational complexity to the algorithm, requiring analytic derivation of the functions governing the system and multiple iterations that are not always guaranteed to converge to the desired solution.

The new resonator algorithms rely on explicit discretization strategies, namely the forward Euler method for inertial masses and the impulse invariant method for modal resonators. These numerical approximation strategies do not exhibit delay-free loops and therefore do not need a non-linear solver when computing interactions such as impacts or frictions. Both the impact and friction interactors have been therefore reimplemented in their direct form, without the previously needed Newton-Rhapson approximation step.

2.1.2 Spectral analysis externals

The `[sdt.spectralfeats~]` object has been slightly modified to allow easier access and routing of the extracted audio descriptors. Instead of having one outlet per feature, The external now outputs all the extracted parameters from a single outlet.

Each output message comes in the form of a qualified pair, in which the first element is the name of the descriptor and the second element is the corresponding extracted value. Using a `[route]` object, part of the core library of Max, it is possible to select and output only the values which are actually needed by the control layer of a specific timbral family, discarding all the others and reducing clutter in the patch.

2.1.3 Porting to PureData

As already stated in D6.6.1, the Sound Design Toolkit is built using a modular architecture, composed of a low-level core library written in ANSI C and a set of wrappers for different platforms built on top of that library.

In the last release of the Sound Design Toolkit, the only supported set of wrappers was for the Cycling '74 Max environment. In the current release, wrappers and externals for PureData have also been included. PureData is another visual programming environment, conceptually very similar to Max (it is actually its "ancestor"), but released under an open source license. The availability of the SDT algorithms under this platform should favor the adoption of the framework in educational contexts, low cost projects, DIY hacks and inside the open source ecosystem in general.

The support of the additional platform required an update of the building system and a slight modification of the installation procedure, which will be described in detail in the next section.

2.2 Downloading and installing

2.2.1 System requirements

The Sound Design Toolkit runs on Microsoft Windows and Apple Mac OS X, and it comes in different flavors:

- Package for Cycling '74 Max (version 6 or above);
- Collection of externals and patches for PureData (version 0.43.4 or above);
- Shared library with ANSI C API.

Max can be downloaded and installed from the official Cycling '74 website (`http://www.cycing74.com`), while PureData can be downloaded and installed from the official website of the project (`http://puredata.info`).

2.2.2 Precompiled binaries

Ready-to-install packages for all platforms and flavors are available on the SDT project website:

`http://soundobject.org/SDT`

To install the Sound Design Toolkit, simply download the desired archive and unpack it in the appropriate location:

Max – In the `Packages/` folder of your Max installation;

PureData – In a directory included in the PureData search path;

C library – In a directory included in the library search path of your operating system.

2.2.3 Compiling from source

In alternative, the Sound Design Toolkit source code can be obtained cloning the main SDT repository, available on GitHub:

```
https://github.com/SkAT-VG/SDT.git
```

The compilation process is extremely simple and fully automated thanks to GNU Make[1] and custom Makefiles for each operating system. To build the software package under Mac OS X, the default terminal application can be used. To compile under Windows, a distribution of the GNU C Compiler and a UNIX style shell are needed. Cygwin is the recommended option, available at `http://www.cygwin.org`.

Once your compilation environment is correctly installed and configured, open its shell. From the repository root, navigate to the `build/` folder and then to the subfolder containing the building scripts for your operating system (`macosx/` or `windows/`).

Before compiling the repository, be sure to remove any already compiled object:

```
make clean
```

Now the SDT framework and the Max externals can be compiled with:

```
make
```

Finally, install the Sound Design Toolkit for the desired platforms:

```
make install_max DSTDIR=<max_installation_path>
make install_pd DSTDIR=<pd_installation_path>
make install_core DSTDIR=<lib_installation_path>
```

As for the precompiled binaries, it is recommended to choose the `Packages` folder of your Max installation as the destination path for the Max version, a directory included in the PureData search path for the PureData version, and a directory included in the library search path of your operating system for the C library version.

[1] `https://www.gnu.org/software/make/`

3 SkAT-Studio

SkAT Studio is a prototype demonstration framework developed by Genesis, designed to facilitate the integration with the technologies offered by the other partners. A SkAT Studio configuration is composed of five groups of modules as presented on Figure 1: 1) Inputs, 2) Analysis, 3) Mapping, 4) Synthesis and 5) Outputs. A model is a combination of a mapping module and a synthesis module that gives pertinent voice and / or gestures control possibilities.

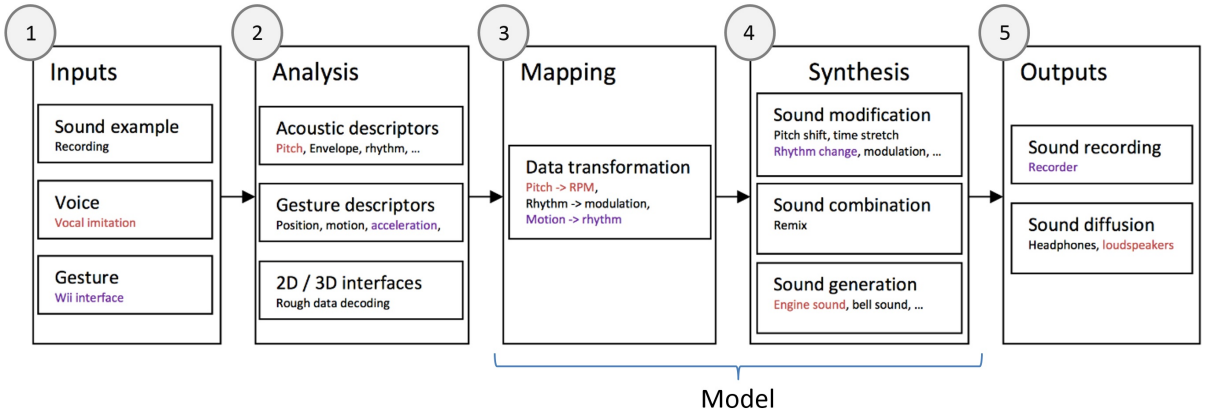


Figure 1: SkAT Studio Workflow

3.1 Changes since D6.6.1

3.1.1 Max 7

The whole development of SkAT Studio moved to Max 7, in order to use the latest improvements of Max. This also guarantees the compability with other pieces of software provided by SkAT-VG partners.

3.1.2 New GUI design

The main GUI of *SkAT Studio* was re-designed (Figure 2) to improve the ergonomy:

- The GUI has been enlarged, in order to be able to display correctly all necessary information for the user;
- The configuration is now made of 5 columns (4 previously), in order to have a completely linear process, as the workflow presented in Figure 1.

3.1.3 SkAT Studio available models and control scenarios from SDT

Eight SDT models are implemented in SkAT Studio: Blowing (Figure 4), Car engine (Figure 5), Crumpling (Figure 6), DC motor (Figure 7), Hitting (Figure 8), Liquid (Figure 9), Rubbing (Figure 10), Shooting (Figure 11). Other SDT models will be integrated during the next months of the project. The user can choose to control either manually or by voice the

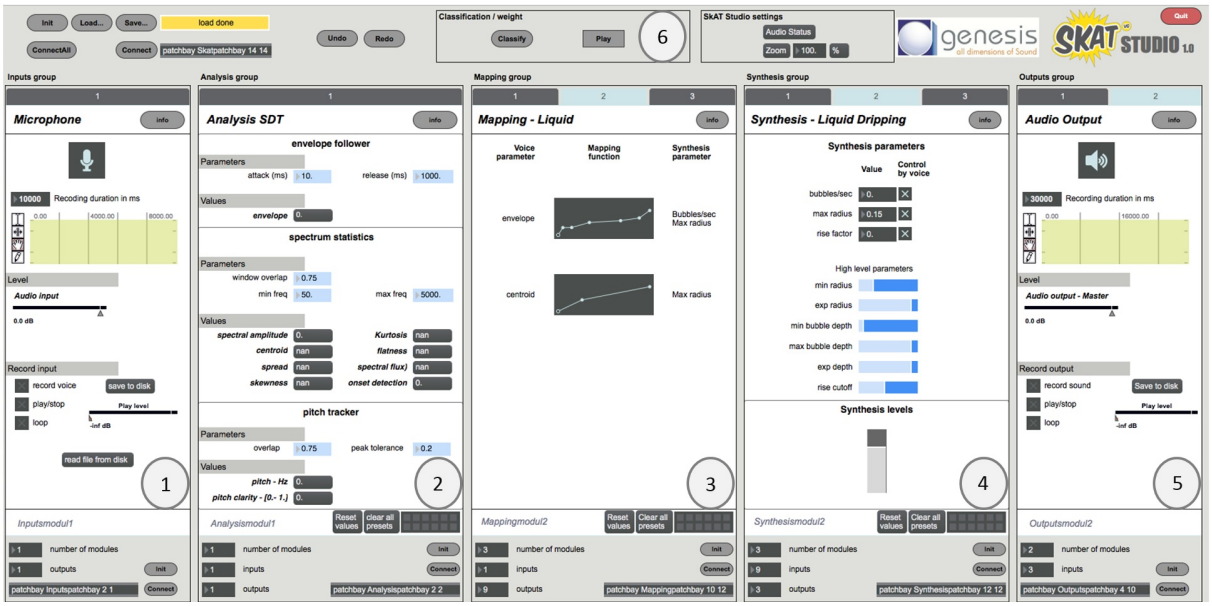
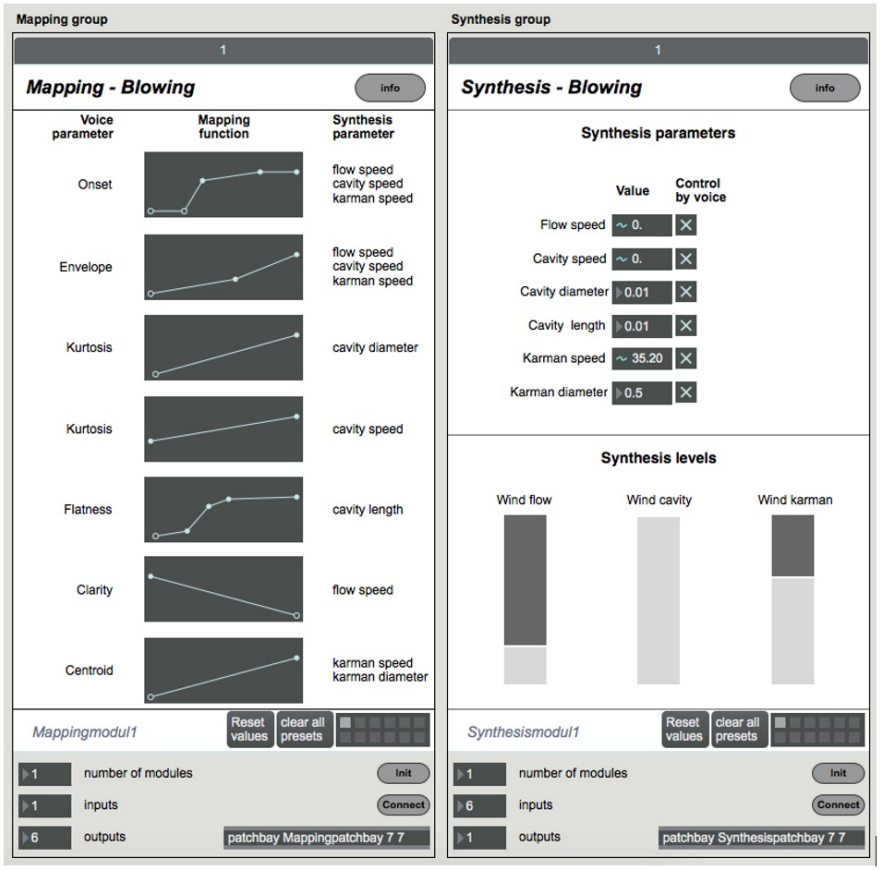


Figure 2: SkAT Studio main GUI

synthesis parameters. Along with each model, a control scenario has been developed by IUAV, which has also been implemented as a mapping module in SkAT Studio. Therefore, for each integrated SDT model, a mapping module and a synthesis module are available and loadable as predefined sessions. The mapping and the synthesis modules are presented for each model on the following figures, along with a brief description of the synthesis parameters (from Figure 4 to Figure 11). The user also has the possibility to set presets by (ctrl + left clicking) on one button of the preset pad. This functionality is available for the analysis module, the mapping module, and the synthesis module (Figure 3). It can then access to the presets by clicking on one preset pad button.

Figure 3: SkAT Studio preset pads

Blowing The blowing model covers a wide range of turbulence noises caused by air or other gas blowing against obstacles of different size and shape. Explosions are not included, as they perceptually belong to a sound category of their own. The model includes the following SDT synthesis externals: [sdt.windflow~] simulating turbulence against surfaces, [sdt.windcavity~] simulating howling cavities, and [sdt.windkarman~] simulating whistling wires or other thin objects. Vocal control is used to manipulate the intensity of the wind flow, as well as the diameter and length of the cavity model (influencing the amount and pitch of howling) and the diameter of the thin object model (influencing the amount and pitch of whistling).



Synthesis parameters controlled by voice:

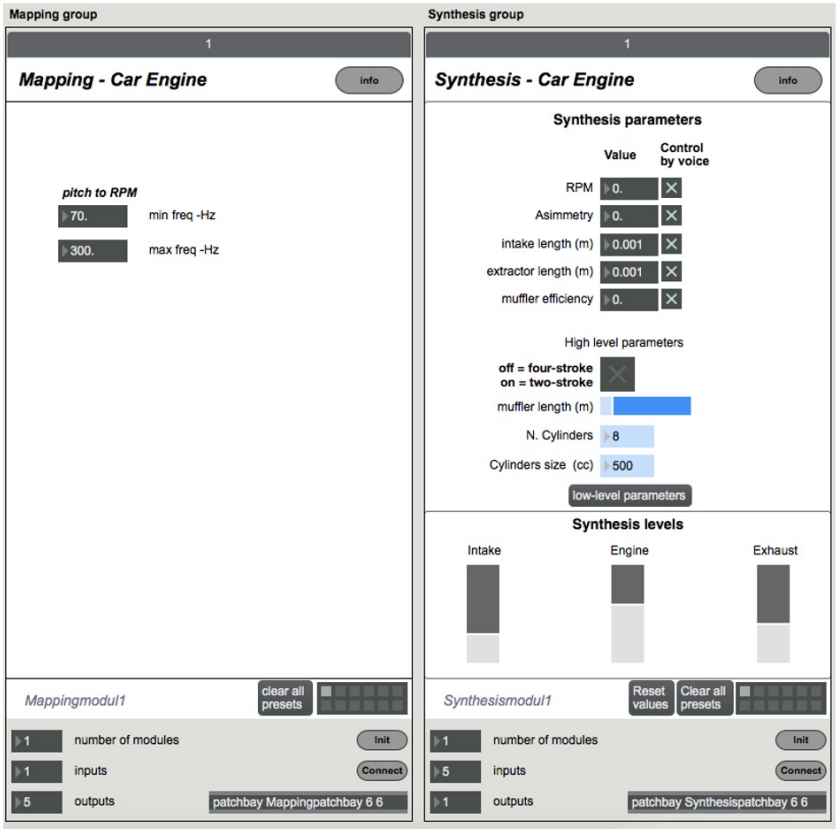
- Flow speed
- Cavity speed
- Cavity diameter
- Cavity length
- Karman speed
- Karman diameter

Synthesis levels controlled manually:

- Wind flow
- Wind cavity
- Wind karman

Figure 4: Blowing model implemented in SkAT Studio

Car engine The car engine model reproduces the characteristic voice of combusion engines, and it is based on the `[sdt.motor~]` SDT synthesis external. Although the physical model is quite complex, with many tweakable synthesis parameters, some of them have only marginal effects on the resulting timbre, and have therefore been collected in a hidden subsection to avoid clutter in the interface. Vocal control is used to drive the Revolutions Per Minute (RPM) of the engine (directly related to the sensation of pitch), the asymmetry of the engine cycle (related to its "growl"), and the size and resonance of intakes, extractors and muffler, often used also in the physical world to tune the voice of real engines. Manual controls are mostly used to tweak parameters related to the engine design, such as the number of cylinders or the type of operation cycle (four-stroke vs. two-stroke), allowing to simulate a wide variety of vehicles, ranging from small mopeds to big road trucks or racing cars.



Synthesis parameters controlled by voice:

- RPM
- Asimmetry
- Intake length (m)
- Extractor length (m)
- Muffler efficiency

Synthesis parameters controlled manually:

- Muffler length (m)
- Cylinders number
- Cylinders size (cc)

Synthesis levels controlled manually:

- Intake
- Engine
- Exhaust

Figure 5: Car engine model implemented in SkAT Studio

Crumpling The crumpling model simulates a series of micro impacts, with a time-frequency distribution similar to the one generally found in sound events involving multiple fractures of a solid object: crushing, breaking, ripping and so on. The model is based on a SDT basic solid interaction composed of a `[sdt.inertial]` mass and a `[sdt.modal]` resonator, put in mutual relation by a `[sdt.impact~]` interactor. On top of this system, a `[sdt.crumpling]` external controls the distribution of micro impacts giving a series of `strike` messages to the inertial mass and manipulating the `fragmentSize` of both resonators. Vocal controls are used to tweak the micro impact distribution of the crumpling event, while manual controls allow to modify the resonating properties of the solid objects.

Synthesis parameters controlled by voice:

- Mass
- Crushing energy
- Crumpling granularity
- Decay scaling factor
- Contact shape

Synthesis parameters controlled manually:

- Frequency
- Decay
- Pickup
- Contact stiffness

Synthesis levels controlled manually:

- Synthesis levels

Figure 6: Crumpling model implemented in SkAT Studio

DC motor The DC motor model simulates the sound of electric motors, and it is based on the [sdt.dcmotor~] SDT synthesis external. Vocal controls are used to manipulate the engine RPM (related to the sensation of pitch), the mechanical stress on the engine (related to the asymmetry of its cycle and therefore to its "growl"), the gear ratio (influencing the brightness of the timbre) and the amount of harmonics (harshness) present in the sound. Like in the combusion engine model, manual parameters are mostly related to the design of the motor: number of coils in the rotor, chassis size and resonance and so on.

Synthesis parameters controlled by voice:

- RPM
- Motor load
- Fear ratio
- harshness

Synthesis parameters controlled manually:

- Coils number
- Air noise
- Brush noise
- Gear noise
- Body resonance
- Rotor noise
- Size of the chassis

Figure 7: DC motor model implemented in SkAT Studio

Hitting The hitting model simulates a single impact between two solid objects. Similarly to the crumpling model, the hitting model is based on a SDT basic solid interaction composed of a `[sdt.inertial]` mass and a `[sdt.modal]` resonator, put in mutual relation by a `[sdt.impact~]` interactor. Vocal controls are used to decide when to strike the object and to control the decay time of the resonating modes of the struck object after the impact. Manual controls allow to edit the structural properties of the simulated mechanical system: hammer mass, impact stiffness and so on.

Synthesis parameters controlled by voice:

- Strike
- Decay global factor

Synthesis parameters controlled manually:

- Hammer mass
- Contact stiffness
- Contact shape
- Dissipation coefficient
- External force on hammer
- Frequency
- Frequency global factor
- Decays
- Pickup
- Extra gain

Figure 8: Hitting model implemented in SkAT Studio

Liquid The liquid model simulates all kinds of sounds related to mechanical interactions involving liquids, such as dripping, splashing, gurgling, gushing or sloshing. The model is based on the [sdt.fluidflow~] SDT synthesis external, which generates spherical bubbles (exponentially decaying sinusoids) according to a stochastic model. Vocal controls are used to manipulate the average number of bubbles per second (affecting the density of the sound event), the maximum bubble radius (affecting brightness) and the frequency rise factor (the amount of "blooping" for each bubble). Manual controls allow to edit other properties of the statistical process, affecting the timbre of the resulting sound texture.

Synthesis parameters controlled by voice:

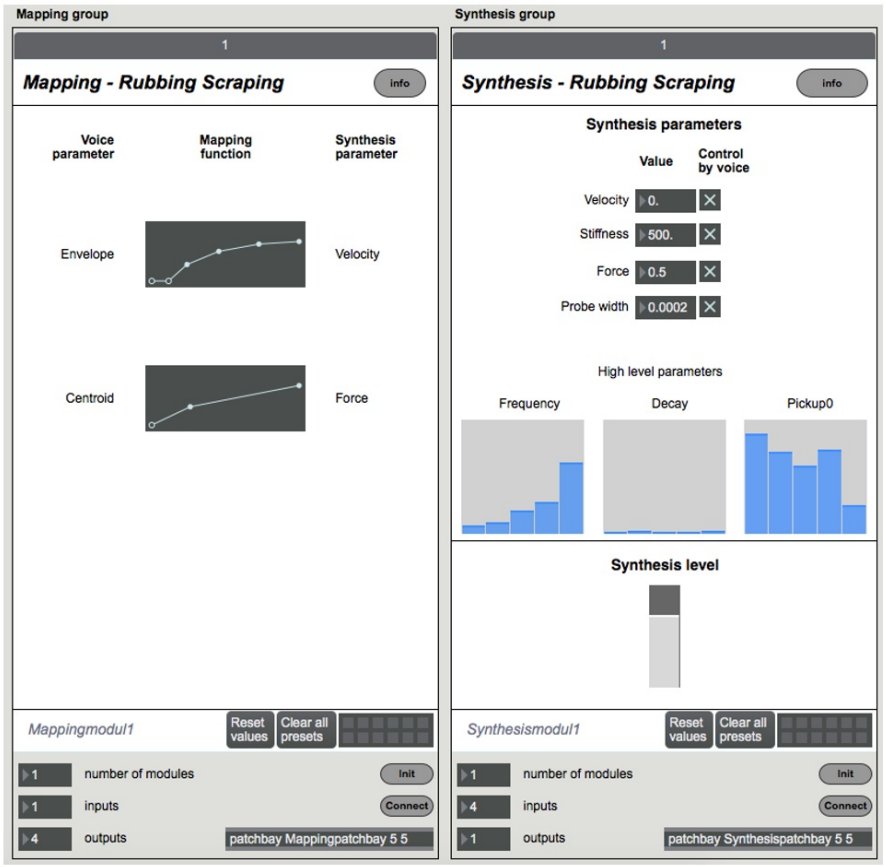
- RPM
- Motor load
- Fear ratio
- harshness

Synthesis parameters controlled manually:

- Coils number
- Air noise
- Brush noise
- Gear noise
- Body resonance
- Rotor noise
- Size of the chassis

Figure 9: Liquid model implemented in SkAT Studio

Rubbing and scraping The rubbing and scraping model, similarly to the crumpling model, simulates a compound mechanical interaction between solids by means of a carefully tuned distribution of micro events. This time, the underlying basic solid interaction implements a `[sdt.friction~]` object as interactor. An additional `[sdt.scraping~]` external directly applies a variable force to the modal resonator, generating the characteristic scraping noise. Vocal controls are used to tweak the variables of the stochastic model behind the scraping event, while manual controls allow to modify the resonating properties of the solid objects.



Synthesis parameters controlled by voice:

- Velocity
- Stiffness
- Force
- Probe width

Synthesis parameters controlled manually:

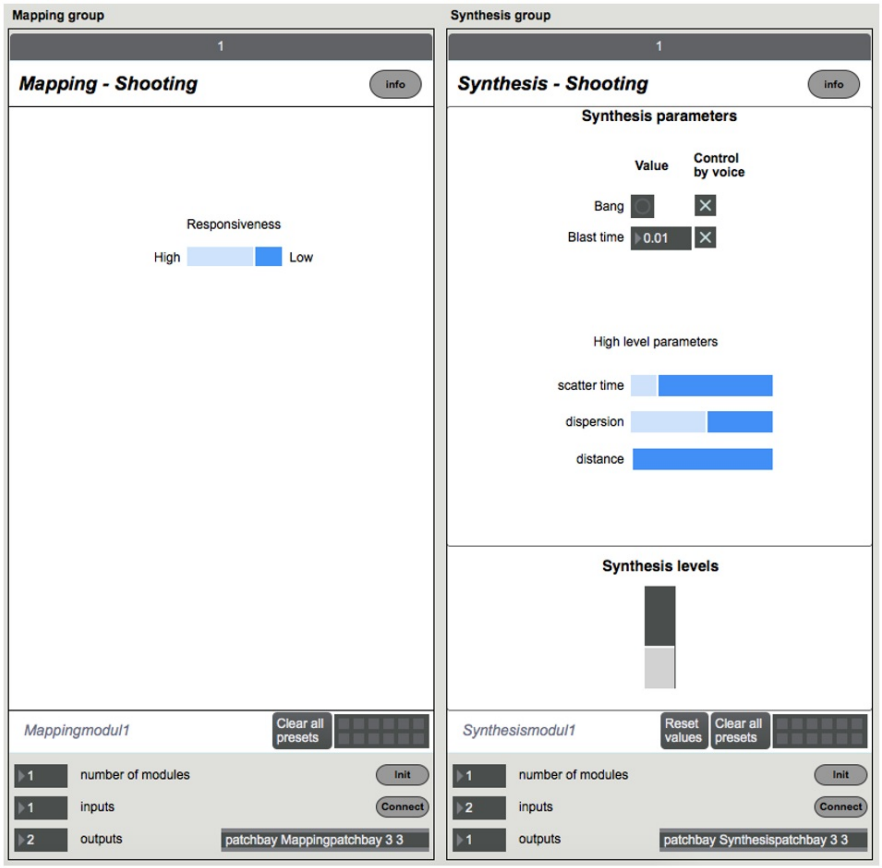
- Frequency
- Decay
- Pickup

Synthesis levels controlled manually:

- Synthesis levels

Figure 10: Rubbing scraping model implemented in SkAT Studio

Shooting The shooting model allows to simulate a wide variety of explosive sounds, and it is based on the `[sdt.explosion~]` SDT synthesis external. Vocal controls are used to trigger an explosion, and to manipulate its power. Manual controls allow to modify other variables with major effects on the explosion timbre, like the amount and duration of scattering phenomena and the distance of the listener from the explosion source.



Synthesis parameters controlled by voice:

- Bang
- Blast time

Synthesis parameters controlled manually:

- Scatter time
- Dispersion
- Distance

Synthesis levels controlled manually:

- Synthesis levels

Figure 11: Shooting model implemented in SkAT Studio

3.1.4 Automatic selection step implemented

The *automatic selection step* is implemented and is designed to integrate the work of IRCAM WP5. At the time of writing, the classifier used to select the categories is the same as the one used for Mimic. This classifier is based on GMM and trained on personal imitations. IRCAM WP5 classifier will be integrated during the next months of the project.

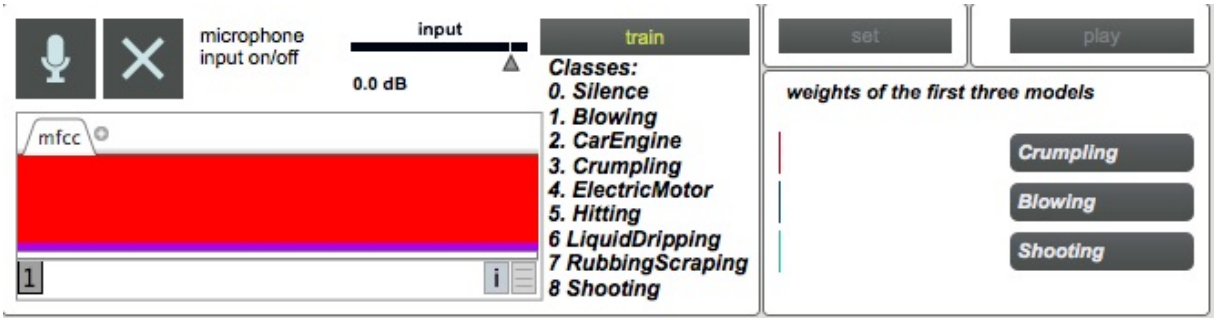


Figure 12: SkAT Studio selection interface - Training step

This selection step is available by clicking on the *classify* button on the top of the main GUI (Figure 2). With the current classifier, the first step is to train the classifier. This is done by asking the user to imitate each sound category (blowing, crumpling, ...) during 30 seconds. With the implementation of the final classifier from WP5, this training step will be skipped. Then, the selection step (button *set*) accepts a sound signal as input, and outputs a *configuration* which defines the behavior of the *play* step. This configuration is composed by the 3 most probable models output by the classifier, weighted by their probability. The SkAT Studio session is dynamically built so that these 3 models can be simultaneously controlled by voice: the needed input, analysis, mapping, synthesis and ouput modules modules are automatically loaded and connected. The weights are displayed in the selection step interface, and are reminded in the main GUI through the weighting module (outputs group, see Figure 13). The 3rd step is to produce sound. This is done by clicking on the play button, available in the selection step interface and in the main GUI.

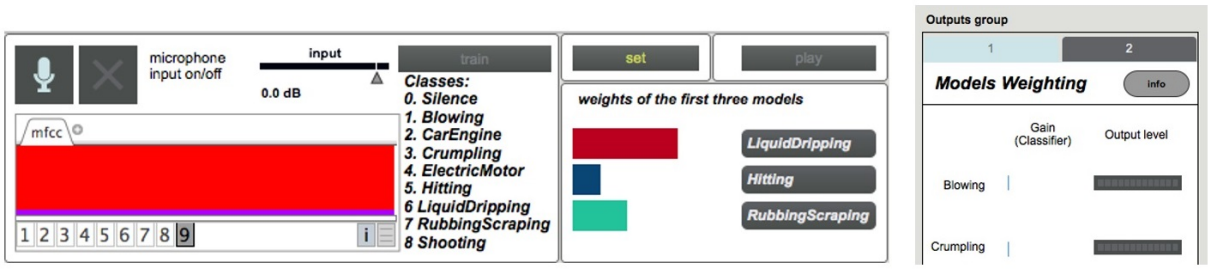


Figure 13: SkAT Studio selection interface - Setting step (left) ; Models weighting output (right)

3.2 Downloading and installing

3.2.1 System requirements

SkAT Studio runs on Mac OS environments as Max application. The integration of IRCAM WP5 classifier should allow SkAT Studio to tun on Windows environments.

3.2.2 Installation procedure

There is no need to install any software on your computer. The application can be downloaded on SkAT-VG website in the "Products" section (http://skatvg.iuav.it) The application can run after unpacking the archive in the desired location. Take care of using the right audio device by clicking on the *Audio Status* button at the top of the application.

4 Sketching sound with voice: miMic

miMic is a prototype hardware and software application for embodied sound design. The physical device is based on a modified microphone, with embedded inertial sensors and buttons. Vocal imitations of referent sounds are automatically classified into sound categories, and subsequently rendered by sound synthesis models controllable in real time by vocal and gestural input.

The main idea behind the development of miMic is to create a tool for sketching sound with voice and gesture, as immediate as a pencil is for sketching on paper. By just using voice and gesture, the sound designer can explore a vast sonic space and quickly produce expressive sonic sketches. The results can then be refined and turned into sound prototypes by further manual adjustment of model parameters through a more conventional graphical user interface.

4.1 Design

4.1.1 Workshops on Vocal Sketching

The design rationale for miMic emerged out of several experiences in the community of sonic interaction design, derived from workshops combining vocal sketching, Foley artistry, interactive-object manipulation, and sound synthesis [CAPT15, MBMR14].

In these workshops, participants are initially exposed to some introductory exercises, to have them warmed up and acquainted with the basic concepts and techniques used in vocal sketching. The following assignments (free exploration of vocal techniques [New04], representation of sonic memories [CAPT15, MBMR14], design of displays and product sounds) are aimed at the creation of a playful and collaborative playground, that would encourage improvisation, engagement, and immediacy in the production of voice-driven displays [HK14].

The system architecture of miMic is designed taking inspiration from the aforementioned experiences, aiming to provide a computational tool capable of seamlessly supporting the ubiquity of sonic sketching.

4.1.2 Bodystorming

A bodystorming session was organized and performed to define the affordances of the system and to come up with a well thought out design. Two actors performed vocal sketching while manipulating a fake microphone and a fake iron: One played the role of the sound designer, while the other played the role of the sound synthesizer.

The observed process could be roughly split into four stages:

Select: The designer performs a vocal imitation of the desired sound. The system classifies the imitation into a sound category and returns its corresponding interactive sound synthesis model;

Mimic: The designer controls the synthesis model in real-time, mimicking the desired sound with voice and gesture.

Explore: The designer further explores the timbral space of the model, using different vocal and gestural control strategies;

Figure 14: Bodystorming session for the design of miMic

Refine: The designer refines the sketch into a prototype, manually tweaking the synthesis parameters through conventional input devices.

These four stages of sound design can be loosely related to the four iterations of the *design funnel* [Bux07, GCMB11]: concepts, exploration, clarification, resolution. In the *Select* stage of our sound design funnel, by selecting one sound model or a mixture of sound models, the designer is effectively defining a sonic concept [OvEJ14]. In the miMic stage, by vocally mimicking a selected sound mechanism, the user gets acquainted with the available sonic space by mirroring its main timbral features with voice and gesture. Exploration is extended in the *Explore* stage, as soon as the user realizes that sound models can be controlled by any kind of vocalization and gesture, and that new neighborhoods of the timbral space can be reached by using creative control strategies that go beyond the boundaries of mere imitation. Finally, the *Refine* stage is the resolving iteration [Bux07], where a rough sound sketch is hard-lined into a more refined sound prototype.

The first three stages can be performed eyes-free, as they rely only on vocal and gestural input. The *Refine* stage, on the contrary, requires a visual interface to explicitly tweak every single synthesis parameter made available by the system. Voice and gesture can still be supportive in this stage, to control only a limited set of parameters while manually tweaking the others.

4.1.3 System architecture

The observations collected during the bodystorming session guided the design process towards the adoption of a classic microphone, augmented with an IMU and two buttons, interfaced to a software application with GUI developed in Max. The buttons allow the selection of two

modes of use emerged during the session, *Select* and *Play*, the first for choosing among the available palette of sound models, and the second for playing with the selected models.

To distinguish between the two different modes, a single button would have been sufficient, and to avoid modal errors, a quasi-modal interface could have been obtained by using a spring-loaded button. We opted for two distinct latching buttons instead, with embedded LEDs for light feedback, so that the current mode is in the locus of attention of the user, and two other states are available for other modes of use not envisioned by the preliminary observations.

The *Select* mode overlaps perfectly with the *Select* stage of the sound design process. The *Play* mode is used as exclusive tool in the miMic and *Explore* sound design stages, and it complements the GUI elements in the *Refine* stage.

4.2 Hardware

4.2.1 Ergonomy

The bodystorming session elicited also questions of ergonomic nature: What kind of microphone shape is the best fit for our application? What is the best place to put the buttons?

We decided that the microphone should be graspable with a single hand, similarly to a pencil, and actuated with at most two fingers. Given these requirements, the attention focused on stage microphones, thus excluding studio configurations which are not supposed to be manipulated.

In the initial idea and sketch, the two buttons were put on the microphone stand or on a separate button pad, similarly to what happens in the E-mic augmented microphone stand presented at Nime [HS03]. However, we soon realized that adding the buttons to the microphone body offered a better feeling and more freedom in gestural manipulation.

The two most common configurations for stage microphones are the "gelato" shape (e.g. Shure SM58) and the classic rounded case (e.g. Shure 55H). For the gelato shape, there is ample possibility to accomodate buttons on the stick, as in the Sennheiser prototype described in [Sch12]. However, the classic roundish shape was preferred as it compactly fits one hand, it is a sort of visual icon, it can sit vertically on a plane and it offers plenty of space for two large visible buttons on top and for the additional electronics inside.

4.2.2 Gesture acquisition

The very first idea for the acquisition of gestural information from the augmented microphone involved the use of a camera, mounted on the microphone shell. The bodystorming session revealed several issues associated with this solution:

- The camera would point to the face of the sketcher, thus capturing only facial gestures;
- It would be difficult to deduce gestural manipulations of the microphone by analyzing a video stream;
- There would be a great amount of data to process, increasing system complexity and response latency.

Figure 15: Holding and using miMic

On the contrary, an Inertial Measurement Unit (IMU) would allow the detection of movement and orientation of the microphone with minimal effort. This solution, while limiting the acquisition only to the gestures achievable by handling the microphone with one hand, allows to efficiently capture acceleration and orientation signals that can be either directly coupled to sound model parameters or used to extract continuous gesture dynamic qualities, such as energy, smoothness or timing of major strokes [ACSB12].

Manual gestures often accompany, complement and reinforce vocal imitations [SLF+15], conveying sound concepts not immediately or intuitively controllable by vocal imitation alone (e.g. noisy, shaky or revolving patterns). Moreover, sensor-augmented microphones are well represented in literature. In particular, the current implementation of the internal miMic circuitry was inspired by the Voicon interactive gestural microphone [PHL12].

4.2.3 Building the prototype

The physical realization of the prototype required the following components:

- 1 Soundsation TA-54D microphone
- 2 pushbuttons, latching, with light (one white, one blue)[2];
- 1 Adafruit LSM9DS0 Inertial Measurement Unit[3];
- 1 Arduino Nano[4];

[2] `https://www.sparkfun.com/products/11975`
[3] `http://www.adafruit.com/products/2021`
[4] `http://arduino.cc/en/Main/arduinoBoardNano`

- 2 220Ω resistors;
- Jumpers and wires;
- Metal tube segments.

Before actually hacking the microphone, the combination of microcontroller, IMU and buttons was temporarily put together and tested on a prototyping breadboard, wiring the circuit with removable jumpers and clips.

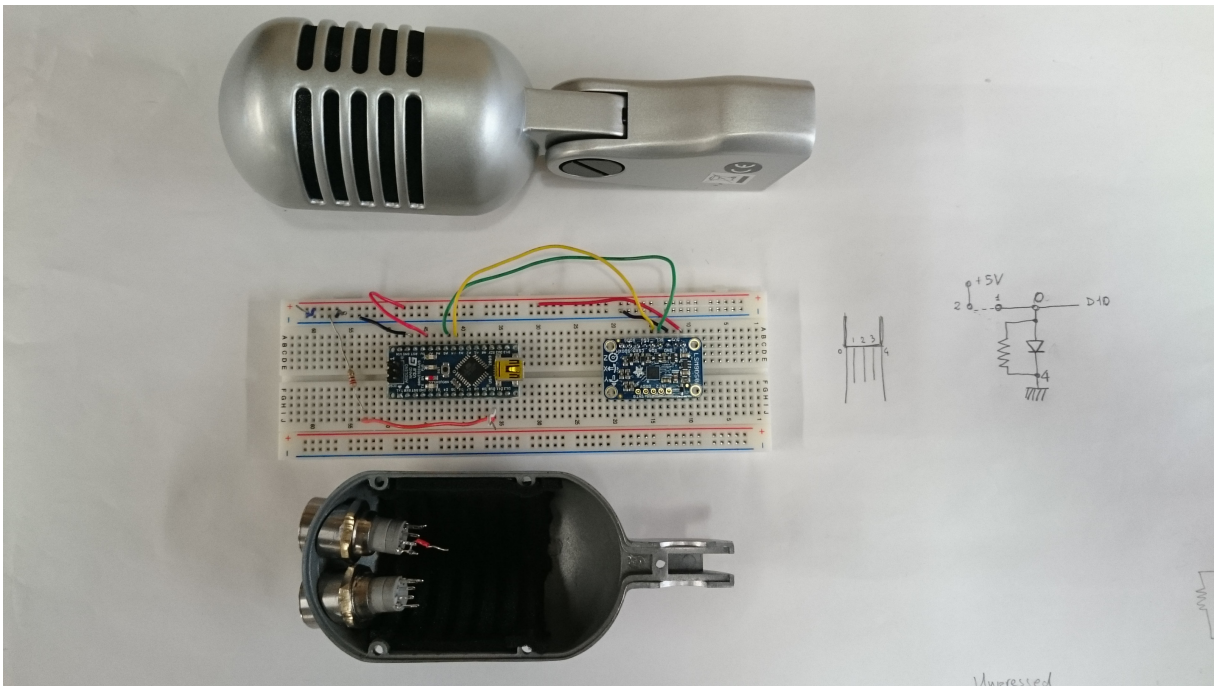


Figure 16: Internal circuitry of miMic, laid down on a prototyping breadboard

Wiring information to connect the IMU breakout board to the Arduino, as well as the code to read data from its sensors, are available on the Adafruit tutorial[5].

The two buttons have five pins each, numbered from 0 to 4 in the schematics depicted on the right side of Figure 16. Pins 0 and 1 are shortcircuited and connected to a digital input of the microcontroller, pin 2 is connected to +5V and pin 0 is grounded. Input from the buttons was handled by making trivial modifications to the Arduino code.

In the microphone shell there is just enough space to host the two buttons, the IMU, and the Arduino Nano. For the latter, it is convenient to use one of the two holes of the plastic board that keeps the microphone capsule suspended. The Nano was embedded in such hole, perpendicular to the plastic board, as shown in Figure 17.

The two buttons were placed on top of the frontal shell. Two holes were drilled and pieces of metal tube were used to bring the buttons over the level of the microphone case. It was also necessary to drill an additional rectangular hole on the bottom part of the back shell to plug the USB cable.

[5] `https://learn.adafruit.com/adafruit-lsm9ds0-accelerometer-gyro-magnetometer-9-dof-breakouts/overview`

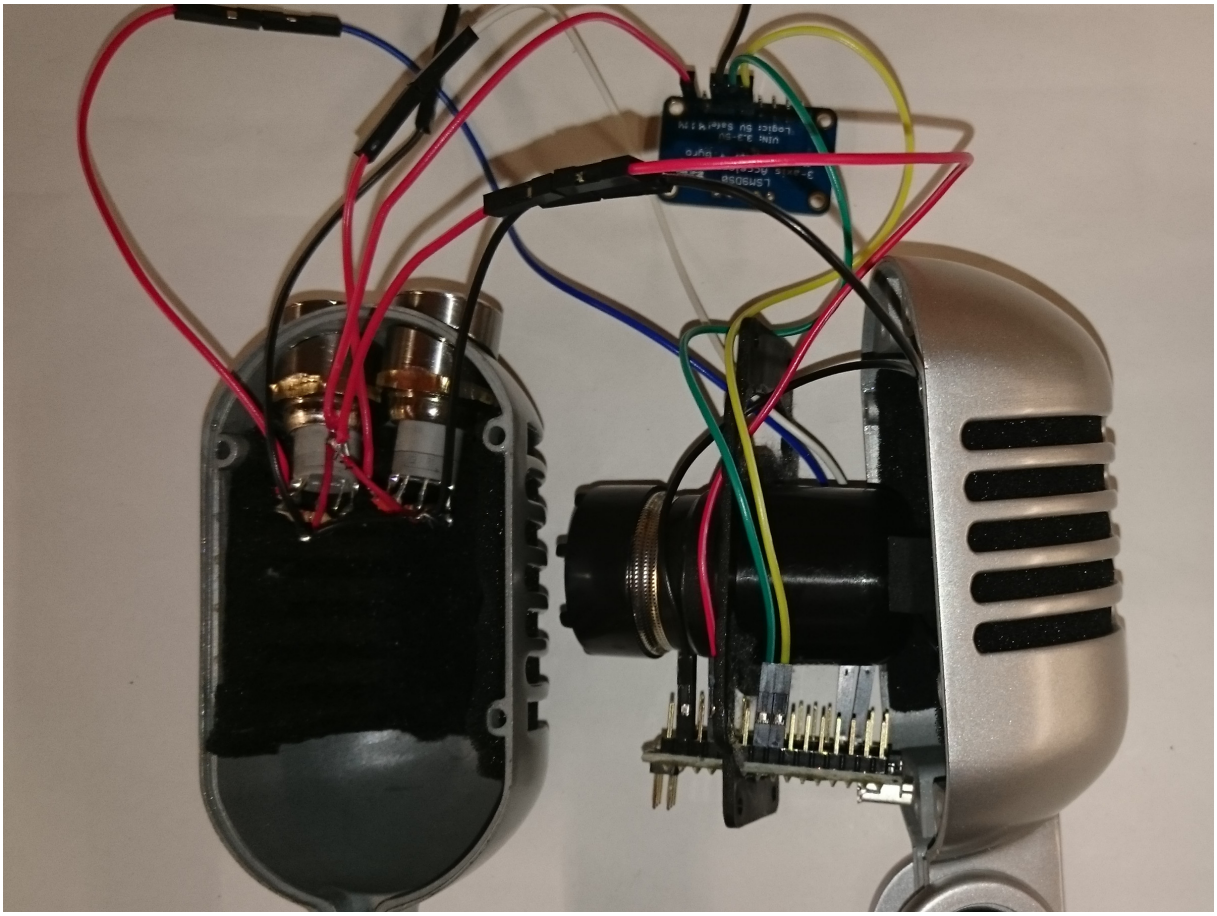


Figure 17: The hacked microphone is fully assembled and ready to be closed

Soldering was limited to a minimum, using jumper wires wherever possible. This choice led to some clutter in the circuitry, but keeps all the parts easily removable.

4.3 Select mode

The selection of one or more sound models in the *Select* mode is operated by an automatic classification of vocal imitations. This function can be designed following two different approaches:

People centered: The classifier is trained on a large set of vocal imitation provided by a large pool of subjects, trying to make a tool which is supposed to work for the casual user;

Individual centered: The classifier is trained on a small set of vocal imitations provided by the user, resulting in a personalized recognition system.

The automatic classification of imitations is the main objective of WP5, and therefore outside of the scope of this deliverable. Effective classification strategies are described in detail in deliverable D5.5.2, and a working classifier informed by this work is expected for month 31

as final outcome of the whole work package. Nevertheless, an incomplete and temporary solution to the problem was needed to demonstrate the application workflow while waiting for more definitive answers. The following paragraphs describe such temporary placeholders, whose only purpose is to be a proof of concept for the two different approaches cited above.

4.3.1 People centered classification

To demonstrate how a general purpose classifier would work in miMic, we implemented a simple model selector based on a classification tree. The construction of such classifier is based on the following steps:

1. Collecting a large set of examples;
2. Using the collected examples to build a classification tree;
3. Implementing an online recognition system, based on the classification tree.

Collection of the examples As the system was meant to be just a proof of concept to demonstrate how such a tool could work, we focused on a small subset of all the 26 timbral families defined in WP4. In particular, the following categories were chosen:

- Wind (261 examples);
- Liquid (275 examples);
- Sawing (271 examples);
- DC motors (265 examples);
- Combustion engines (257 examples);

The 1329 examples were extracted from the large imitation database collected by IRCAM for deliverable D4.4.1, and conditioned to be normalized at -1 dB FS with a length of at least 4 seconds.

Offline training The people centered classifier is trained offline. The results of the training procedure, statically embedded into the system, are then used in the online recognition phase. For this reason, the adoption of this approach in miMic does not require the initial *Train* step.

The collected examples are analyzed using a patch developed in the Cycling '74 *Max* visual programming environment, containing several feature extraction externals (plugins) which are part of the Sound Design Toolkit: `[sdt.spectralfeats~]` (spectral information, see Section 2.1.2), `[sdt.envelope~]` (envelope follower) and `[sdt.pitch~]` (fundamental frequency estimator). These objects allow the extraction of the following audio descriptors:

- Statistical moments of the spectrum (centroid, spread, skewness, kurtosis);
- Spectral flatness;

- Spectral flux;
- Whitened spectral flux (onset detection);
- Fundamental frequency (pitch);
- Amplitude envelope;
- RMS power;

Audio analysis is carried out on windows of 4096 samples with an overlap of 75%, and the resulting descriptors are sampled at an interval of 20 milliseconds over a length of 4 seconds. Spectral information is computed with a lower frequency bound of 50Hz and an upper frequency bound of 5000Hz, to reduce noisy contributions outside of the vocal range. The envelope follower has an attack time of 10 ms and a release time of 1000 ms, while the pitch estimator has a tolerance parameter of 0.2. For an explanation of the algorithms implemented by the externals and the exact meaning of their parameters, please refer to Sections 2.6 and 2.7 of deliverable D6.6.1.

The median value and interquartile range (IQR) is computed for each of the extracted descriptors. A Ratio between IQR and median is computed for the *envelope* and *RMS power* features, since they are dependent on the signal level. The Max patch produces a line of text for each imitation example, which includes a label and the sequence of feature median and IQR values. These tags are used to derive a classification tree, computed in Matlab.

Online model selection The classification tree is imported in a Max patch for online recognition. The vocal input undergoes the same processing adopted for the offline training: each new imitation is analyzed to extract median value and IQR from the same audio descriptors used for classification. The tree is visited according to the computed data, selecting the class that best matches the new vocal imitation and playing back a synthetic example of that class to the user.

Preliminary tests and observations immediately highlighted the poor performance of this simple classifier. Formal evaluation and performance optimization were not carried out because, as already mentioned above, this classifier is just a placeholder, whose only purpose is to demonstrate the people centered approach to classification in miMic while waiting for the final results of WP5. However, it was also noticed how the user rapidly adapts to the classifier, learning to produce vocalizations that trigger the desired sound class.

4.3.2 Individual centered classification

The opposite approach consists in training a classifier with just a few examples provided by a specific user, to configure a personalized recognition system. We tested this approach by using a library called *MuBu*, a set of Max externals developed by IRCAM for content-based real-time interactive audio processing.

The `[mubu.gmm]` object extracts Mel-Frequency Cepstral Coefficients (MFCC) from the audio examples, and models each sound class as a mixture of Gaussian distributions. At least one example per sound class is needed for training the classifier. During the recognition phase, the likelihood for each class is estimated. The selection of sound models can return a single

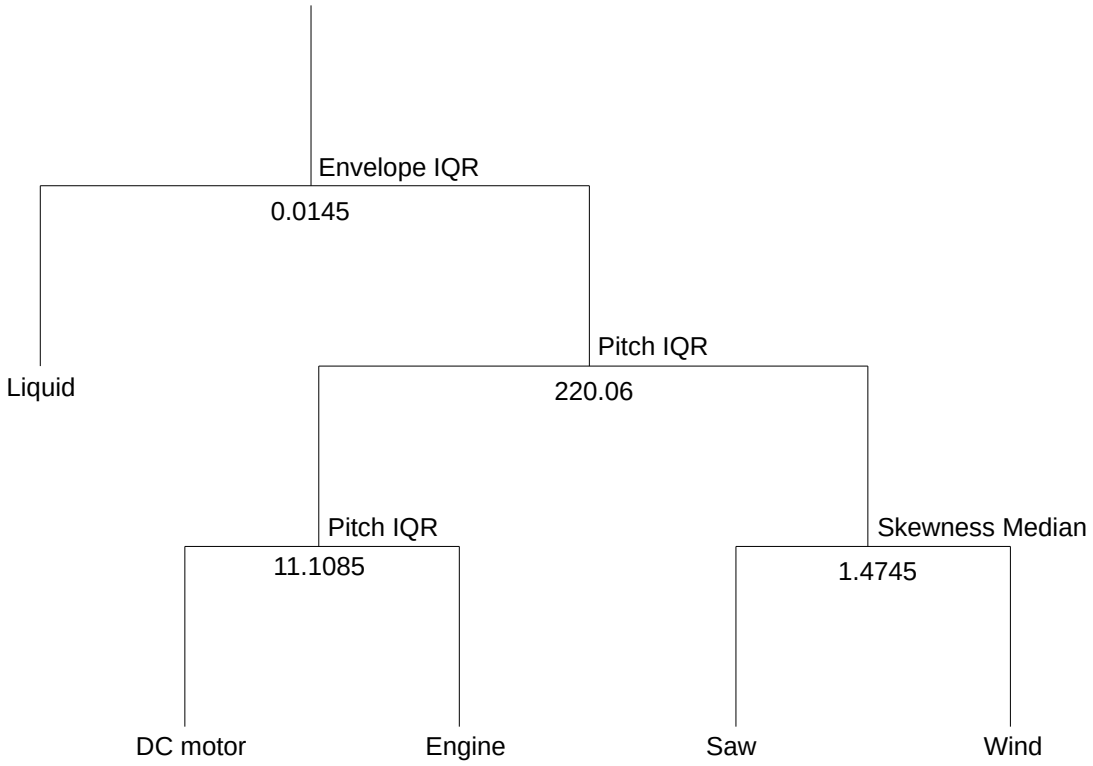


Figure 18: Decision tree obtained by training the classifier with the collected examples

result, based on the best match, or a weighted mixture using the likelihoods as mixing weights for the corresponding sound models.

The individual-centered selection of sound models requires a training phase, in which the user provides his or her own examples to the classifier. This additional *Train* mode is activated when both the buttons of miMic are pushed at the same time. An audible feedback instructs the user on when to provide the required sound examples.

The personalized classifier achieved much better results compared to the generic one, and it was therefore the only one actually included in the final prototype of the system. Nevertheless, it also represents a temporary solution, as the final classifiers are being developed, tested, evaluated and will be provided by WP5.

4.4 Play mode

The sound synthesis engine of miMic is composed by a subset of the sound models palette available in the Sound Design Toolkit (SDT), already cited in Section 2 and presented in detail in deliverable D6.6.1. In particular, each of the five sound categories recognizable by the classifiers is rendered by its corresponding *timbral family*. The concept of timbral family, already introduced in D6.6.1, can be summarized as a specific configuration of one or more sound models representing unambiguously discriminable sound categories in terms of

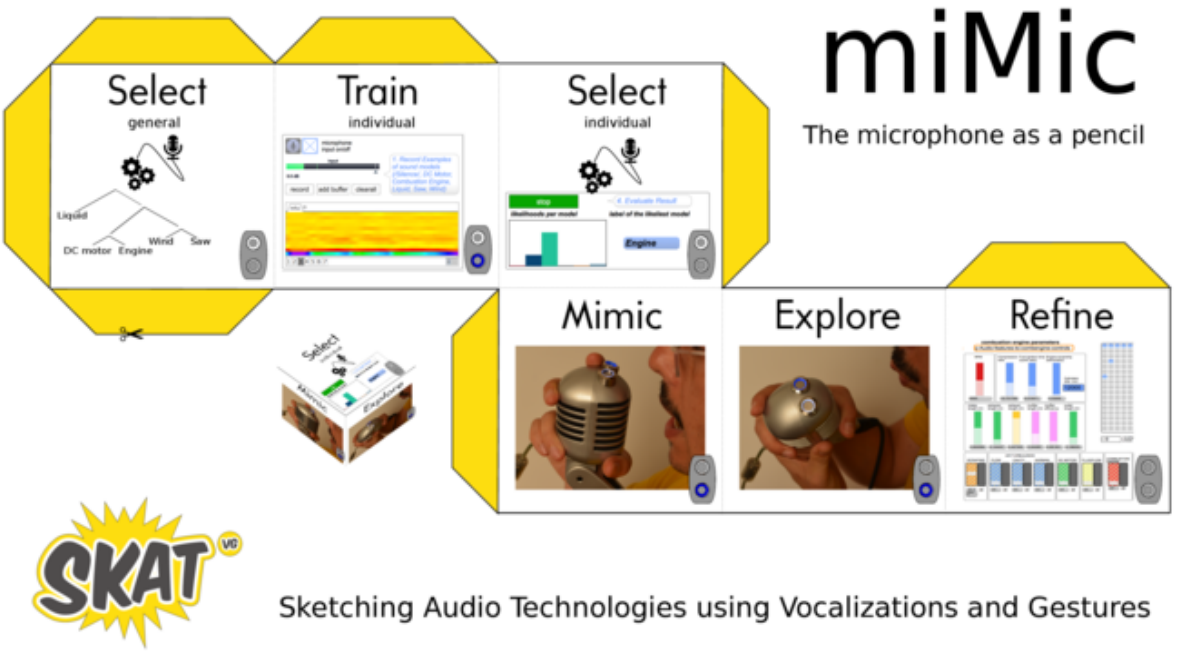


Figure 19: The different stages of sound design, as proposed in miMic, with the additional *Train* step required by individual-centered classification.

interaction, temporal and timbral properties.

Timbral families are implemented as Max patches, in which one or more physically-informed algorithms are selected and combined in order to simulate sounds belonging to the different categories. The synthesis parameters are partially fixed and/or limited in range, defining the timbral space of the simulation. For each of them, a control layer has been built and tuned to immediately interpret vocalizations and gestures as control signals.

The five timbral families selectable by the miMic classifiers are all embedded in another Max patch, collecting the available models in a single application and providing a graphical interface. The GUI allows to draw detailed maps between vocal features and parameters, and to manually set each individual model parameter.

The main assumption behind this design is that if a user selects a sound model (e.g., wind) then he/she will start controlling the model by producing wind-like sounds with the voice, so the control layer associated with the model should be ready to interpret such kind of control sounds. Only at a later stage, the user might want to explore different vocal emissions or to tune parameters by hand, to transform the rough sound material produced by using the augmented microphone into a more refined prototype.

4.4.1 Vocal and gestural control

Similarly to what happens in the classification step, the use of vocal input to control in real-time and explore the timbral space of the sound models requires the extraction of higher level features and descriptors, conveying meaningful information about the contents of the signal. To accomplish this task, the same SDT analysis externals used in the people centered classifier have been adopted.

In the construction of the control layer, the limits of humans in controlling the dimensions of timbre must be considered. Research conducted in the early stage of WP4 investigated the mechanisms of effective vocal imitations by studying if speakers could accurately reproduce basic auditory features, such as pitch, tempo, sharpness, and onset. The results of this research, published in [LJH+15] and in deliverable D4.4.2, show that speakers are able to discriminate these basic dimensions of timbre while listening, but cannot always accurately reproduce them with their voice, especially if they are requested to articulate many parameters at the same time.

The main practical implication of these results is that only a tiny subset of all the available features is actually useful to describe a timbral family at any given time. Different timbral families, however, require different imitation strategies which in turn are best represented by different features and descriptors. For instance, onset detection conveys useful information about impulsive sounds, like *hitting* or *shooting*, while fundamental frequency estimation is more relevant in pitched sounds, like those made by electric motors and combustion engines.

Each timbral family is provided with a customizable control layer that allows to connect the vocal and gestural descriptors to the interactive sound synthesis parameters. After chosing the small subset of descriptors best suited to describe and control the timbral properties of the timbral family, each descriptor is scaled, combined and assigned to one or more synthesis parameters exposed by the sound models belonging to the timbral family.

4.4.2 Mapping strategies

The association between the audio descriptors extracted from the vocal/gestural signals and the synthesis parameters made available by the timbral families is defined by hand, striving to meet the expectations of the listeners about the sonic behavior of the synthesis models with respect to the imitations provided by the users. Andrea Cera, a professional sound designer, was specifically hired to accomplish this task. In many cases, it is possible to exploit almost directly the common relations between timbral features and physical parameters. For instance, as the energy of an impact is expected to affect the amplitude and the spectral bandwidth of the resulting sound, similarly the timbral characteristics of its imitation will produce the same effect. Other notable examples include:

- The *pitch* of a vocal signal, which can be directly mapped to the revolutions per minute of both combustion engines and electric motors;
- The spectral *centroid*, which can be related to the concept of size (for instance, the size of bubbles in liquid sounds);
- The spectral *spread*, associable to the concept of hollow body resonance, as found in many timbral families (e.g cavities in an air flow, the chassis of an electric motor, the

exhaust system of a combustion engine, a container filled with a liquid, etc.);

- The temporal and spectral *onset* information, usable to trigger discrete events, like single impacts or explosions;
- The *zero crossing rate* of a vocal imitation, which can be put in relation with the graininess in higher level textures such as rolling, rubbing, scraping and crumpling, to the harshness of machine sounds, and in general to all the synthesis parameters related to the concept of noisiness.

As previously stated, humans often accompany vocal imitations with manual gestures, which may mimic the sound production mechanism or communicate some aspects of the sound morphology [CBB+14]. In particular, recent research [SLF+15] has shown that humans consistently use shaky gestures to highlight the noisy quality of sounds they are imitating.

In the current realization of miMic the following movement qualities are extracted as continuous values:

- energy;
- shake;
- spin;
- kick (rapid changes in acceleration).

The dynamic behaviors, represented by continuously varying signals, are mapped to sound synthesis parameters similarly to what happens for the audio features, to reinforce or complement the control actions exerted by the voice.

In miMic, a manually defined mapping strategy based on the knowledge of an expert was preferred over other approaches, such as the use of machine learning techniques, partly because it is more consistent with the concept of ecological listening on which the whole project is built, and partly because the added flexibility provided by a supervised learning algorithm is not needed in a contol layer which is so tightly bound to a very specific application.

While one-to-one and one-to-many mapping between descriptors and parameters is a trivial task to accomplish, many-to-one associations are achieved through specific Javascript functions, that can be recalled and edited in the control layer, directly. Finally, a control module per parameter allows to smooth, scale, and eventually distort the audio feature value into the range meaningful for the control parameter.

4.4.3 Manual refinement

Due to the limited number of timbral properties which can be articulated at the same time during vocalization, not all the timbral possibilities provided by the synthesis modules are controllable by vocal imitations. It is nevertheless possible to produce convincing and recognizable sonic sketches by mimicking a few salient, perceptually-relevant features for their identification. In general, voice and gesture are used for a coarse control of the synthesis models, leaving further timbral refinement to manual operation on a graphical user interface or on other external devices [RMM16]. The more subtle nuances, not directly controllable by

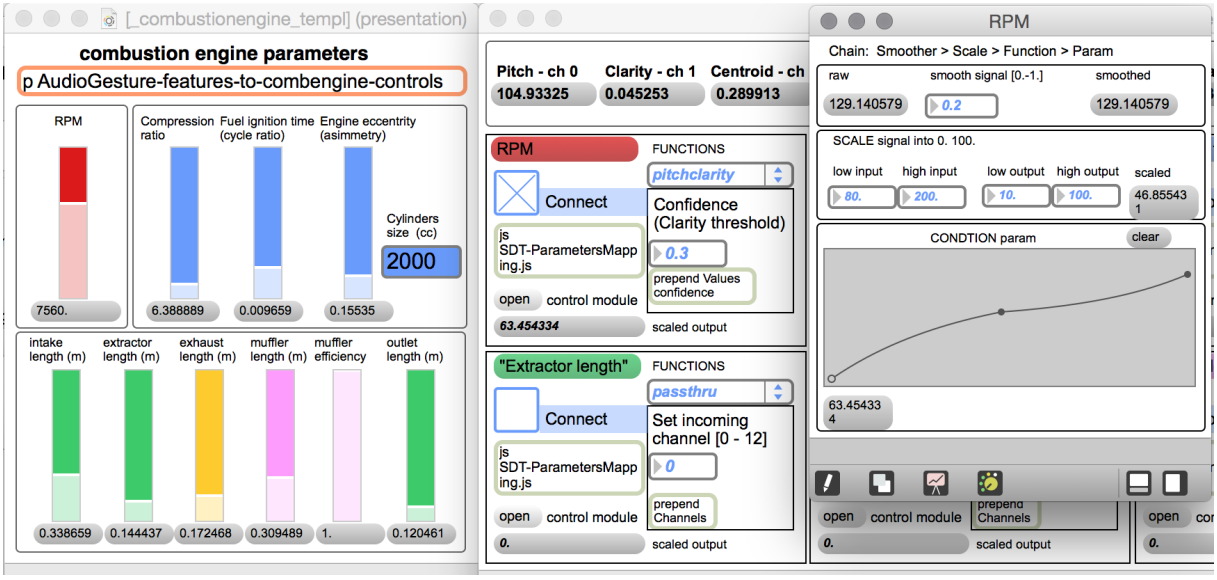


Figure 20: Mapping between input descriptors and synthesis parameters can be edited in the control layer

vocal input, can be tweaked on the GUI of each module using traditional input methods such as virtual sliders and knobs.

5 Sketching sound with recorded vocalisation and live gestures: MIMES

MIMES is a prototype that uses vocalization and movements for interactively designing expressive sounds. MIMES follows a similar approach to miMic, by implementing modes of use suitable for *Exploring* and *Refining* vocal sketches. In particular, the first version of MIMES implements two modes of operation.

1. **Record:** The user records a vocalization to propose an initial sound morphology. The user can then replace this initial vocal sketch with a synthesized sound sketch sharing similar characteristics.

2. **Control:** The user can play and further modify the sound sketch through gestures and the manipulation of tangibles objects. In other words, the user can interactively sculpt the initial vocalization.

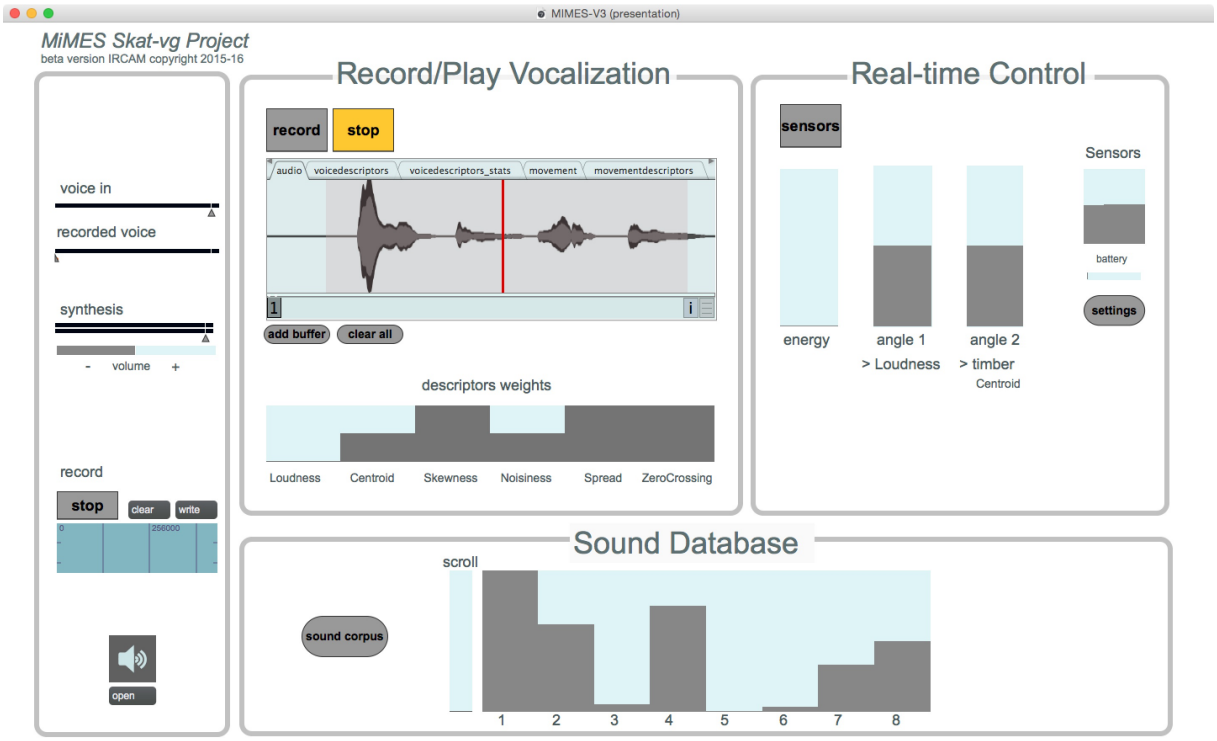


Figure 21: Screenshot of the first version of the MIMES software (June 2016), using concatenative synthesis for sound generation.

Figure 21 shows a screenshot of the software. The two modes of operation correspond of the two columns identified as *Record/Play Vocalization* and *Real-time Control*. In this version, the sound synthesis is implemented using concatenative sysnthesis where different sound databases can be mixed (using the graphical interfaces referred to *Sound Database* in Figure 21).

5.1 MIMES Architecture

The basic architecture of MIMES is shown in Figure 22, where the two modes of operation described in the previous section are also indicated. We describe below the different components.

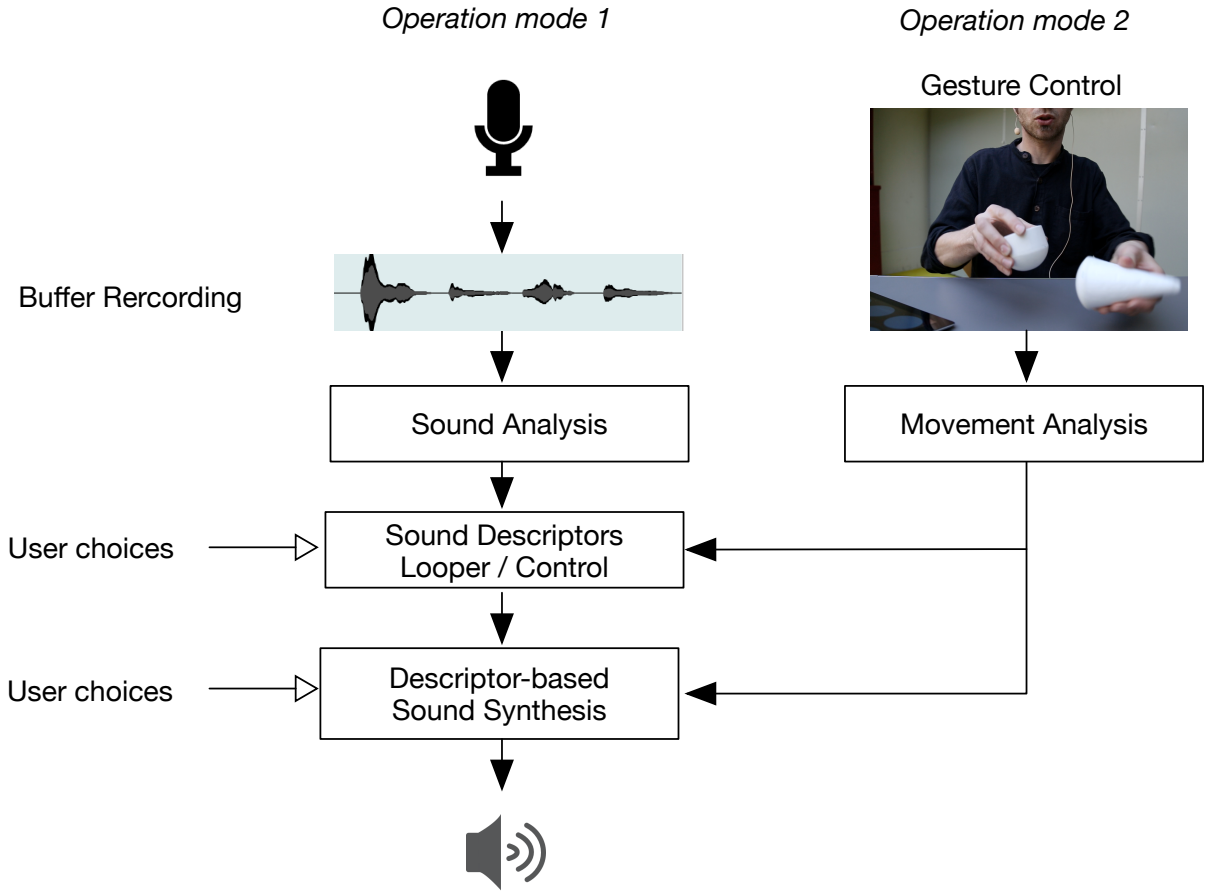


Figure 22: Architecture of the MIMES prototypes

5.1.1 Voice Recording and Sound Analysis

In the first mode of operation, a vocal sketch is recorded in a buffer, adjustable in size by the user. This audio buffer is then analyzed to obtain a time series of audio descriptors. The currently available audio descriptors are *loudness*, *spectral centroid*, *spectral skewness*, *spectral spread*,*noisiness*, and *zero crossing*. Additional descriptors may be added in future versions of the software.

5.1.2 Sound Descriptors Looper

The sound descriptors are played in a loop, and their respective weights can be adjusted by the user using a graphical interface. The sound descriptors are sampled and fed to the sound synthesis engine at a fixed rate.

5.1.3 Descriptor-based Sound Synthesis

The sound synthesis engine generates sound that matches the sound descriptors. Different sound synthesis methods can be used. Currently, MIMES implements corpus-based concatenative sound synthesis [Sch07] which in our case is also similar to audio mosaicing [SSCL10]. For each frame of sound descriptor values, the synthesis engine plays a small sound fragment that is selected to match as closely as possible the timbral features of the input. This sound synthesis engine relies on specific sound databases, previously segmented and analysed, referred as the *corpuses*. The user can explore different choices for changing the synthesis behavior. The two main options involve the manipulation of:

- Corpus weights (e.g. based on synthetic or concrete sounds), which can be mixed as seperate audio channels.
- Sound descriptors weights, putting different emphasis on the timbral aspects used to select the sound samples.

5.1.4 Gesture Control

In the second mode of operation, gesture capturing devices (see Section 5.2 for a detailed description) are used to further modify the sound sketches. First, movement analysis is performed to obtain higher level gesture descriptors from the raw input coming from the sensors, such as the *orientation* (typically quaternions or Euler angles) or the *movement intensity*. These movement descriptors can alter the original sound descriptor profiles or directly control the sound synthesis parameters (corpus weights, sound descriptor weights).

More complex control paradigms can also be used using the *mapping by demonstration* method [Fra15]. In this case, an example gesture is first recorded while listening to the sound sketch obtained through the first mode of operation. The relationship between the gesture parameters and the sound descriptors is learned by a Hidden Markov Model. This relation can then be used to regenerate the sound descriptors while replaying the gesture. Performing the gesture with some variations will generate variations in the sound descriptors, and consequently in the final sound sketch.

5.2 Implementation

The current implemenation of the MIMES prototype is currently made of a software package and a set of tangible interfaces.

5.2.1 Software

The software is built using Cycling '74 Max (version 7), and makes use of the MuBu library [SRS+09] freely available from the IRCAM Forum[6]. The sound descriptors are obtained by the external `[pipo.ircamdescriptors]`, available in the MuBu package.

[6] `http://forumnet.ircam.fr/product/mubu-en/`

5.2.2 Tangible Interfaces

The tangible interfaces of MIMES are built using 3D printing with ABS plastic material, and equipped with different wireless sensors (a triple-axis gyro, a 3D accelerometer and a triple-axis magnetometer). Examples of the objects are shown in Figure 23. The shapes were designed to exhibit specific affordances. Some objects also contain force-sensitive-resistors (FSR) and piezo sensors, connected to the main microcontroller through I2C using a Teensy 3.0 development board. Some of these objects were designed during a previous project (ANR Legos) to study sensori-motor learning. These objects have been further developed, adding interaction with the vocal component, and placed within another research context with completely different goals.

Figure 23: Examples of 3D printed objects used for MIMES

6 Integration

During the Month 25 project meeting, held at IRCAM inParis at the end of January 2016, an integration plan to merge the features offered by miMic with those offered by MIMES into one single application was devised. The following guidelines are meant to support the development of the SkAT-VG tools during the remaining months of the project, towards the realization of an integrated environment for designing sound with vocalizations and gestures.

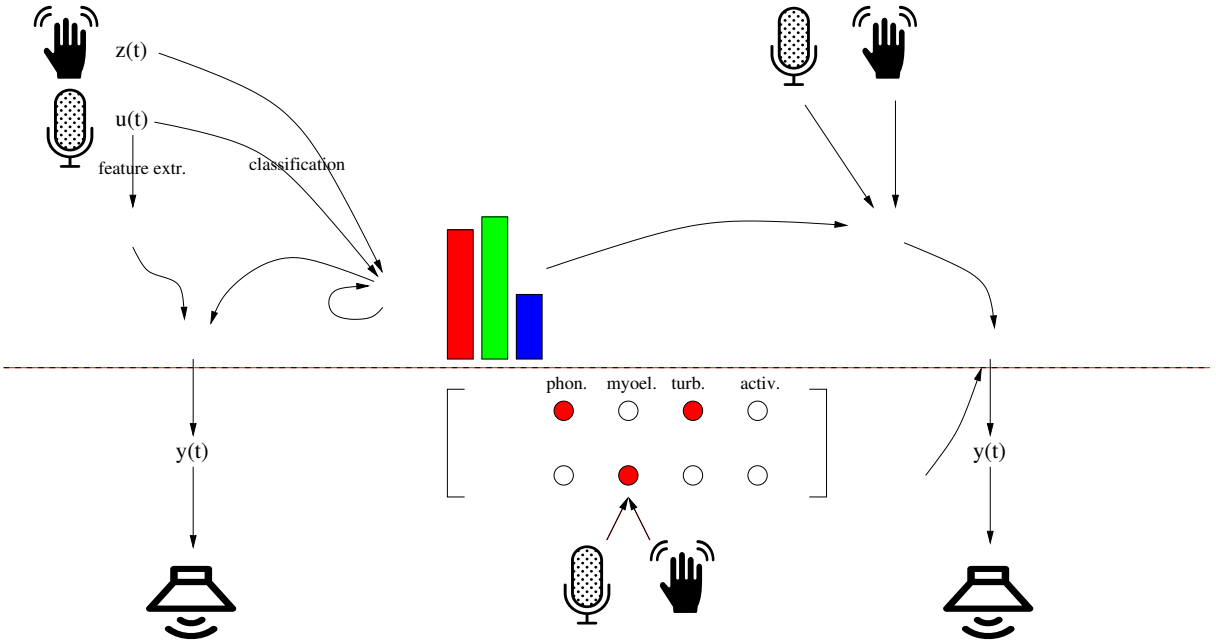


Figure 24: Sketch of miMic and MIMES integration

6.1 General system architecture

From the user's perspective, the system is structured into two main modes: *set* and *play* (see Figure 24). In the *set* mode, the microphone captures a signal $u(t)$ of a vocal utterance and the inertial measurement unit (IMU) captures movement signals (acceleration, orientation, etc.) $z(t)$. Based on these signals, the system automatically proposes:

1. a vector of weights $\vec{\gamma}$, giving the relative importance of different sound synthesis models (classification),
2. a vector of functions of time $\vec{\pi}$, with each element representing the temporal evolution of signal features (feature extraction).

6.2 Set mode

Given the model weights and the parametric control trajectories, the system is able to choose a subset of its sound models and control them to generate a sound similar to the input

utterance. The output $y(t)$ is an audio signal that goes directly to the speakers and gives immediate feedback to the user, a sort of synthetic echo to the proposed vocalization.

Users can listen to such feedback and are given the possibility to change the model weights $\vec{\gamma}$, to give more or less importance to one sound model rather than another. This change of weights can be performed using physical or virtual sliders. Each time the user changes one of the model weights, a new feedback sound is produced. To make the system relatively simple and usable even with a very limited graphical interface, it is not possible to change values in $\vec{\pi}$ or to select a model which has not been included in the $\vec{\gamma}$ vector.

In fact, even though several sound models are available, only the three that are ranked as most likely by the classifier are retained. Their relative weights γ_i get visualized through colored sliders, for potential modification. Labels indicate to which model each slider corresponds to but, in order to help the user with more stable configurations, the relative order of the sliders and the color associated to the models is preserved.

The gestural part is based on motion signals $z(t)$ coming from the IMU, which affect the choice of $\vec{\gamma}$. Section 3 of deliverable D4.4.2 points out a major discrimination between stable and shaky gestures. A shaky gesture can give more weight to a noisy model, whereas a continuous gesture generally indicates a steady tonal sound. The system works regardless of the implemented synthesis models, which can be both physically informed (as in miMic) and concatenative (as in MIMES).

6.3 Play (and loop) mode

In *play* mode, $\vec{\gamma}$ is used to mix the output of sound models that are controllable in real-time through voice and gesture, according to the control layer of each model. This modality doesn't require any visual interface. In the play mode there is another submode called *loop*, which requires both weights $\vec{\gamma}$ and feature functions $\vec{\pi}$.

For the sake of simplicity and effectiveness of control, the feature functions are indeed reduced to a set of four articulatory controls $\alpha(t)$: *Phonation*, *myoelasticity*, *turbulence* and *activity*. These three salient components in vocal imitations can be activated independently, and therefore be present simultaneously.

The synthetic sound generation is looped, and through the $\vec{\lambda}$ selectors some elements of $\vec{\alpha}$ can be replaced by live vocal and gestural control. Through the $\vec{\phi}$ selectors, some articulatory controls can be frozen during looping.

As an example, if α_i is the turbulent component of the vocal utterance, controlling α_i live allows the user to influence the turbulence-related synthesis parameters of the model with his or her voice and gesture. More than one feature can be controlled live at any given time, while keeping the others as they were recorded in the *set* mode, or frozen. Gestures might also be used to temporally unfold the loop, controlling the reading of the loop table through mapping by demonstration, as it happens in MIMES.

References

[ACSB12] Sarah Fdili Alaoui, Baptiste Caramiaux, Marcos Serrano, and Frédéric Bevilacqua. Movement qualities as interaction modality. In *Proceedings of the Designing Interactive Systems Conference*, DIS '12, pages 761–769, New York, NY, USA, 2012. ACM.

[Bux07] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[CAPT15] Baptiste Caramiaux, Alessandro Altavilla, Scott G. Pobiner, and Atau Tanaka. Form follows sound: Designing interactions from sonic memories. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3943–3952, New York, NY, USA, 2015. ACM.

[CBB+14] B. Caramiaux, F. Bevilacqua, T. Bianco, N. Schnell, O. Houix, and P. Susini. The role of sound source perception in gestural sound description. *ACM Trans. Appl. Percept.*, 11(1):1:1–1:19, April 2014.

[Fra15] Jules Françoise. *Motion-sound Mapping By Demonstration*. Theses, Université Pierre et Marie Curie - Paris VI, March 2015.

[GCMB11] Saul Greenberg, Sheelagh Carpendale, Nicolai Marquardt, and Bill Buxton. *Sketching User Experiences: The Workbook*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

[HK14] Daniel Hug and Moritz Kemper. From foley to function: A pedagogical approach to sound design for novel interactions. *Journal of Sonic Studies*, 6(1), 2014.

[HS03] Donna Hewitt and Ian Stevenson. E-mic: extended mic-stand interface controller. In *Proceedings of the 2003 conference on New interfaces for musical expression*, pages 122–128. National University of Singapore, 2003.

[LJH+15] Guillaume Lemaitre, Ali Jabbari, Olivier Houix, Nicolas Misdariis, and Patrick Susini. Vocal imitations of basic auditory features. *The Journal of the Acoustical Society of America*, 137(4):2268–2268, 2015.

[MBMR14] Stefano Delle Monache, Stefano Baldan, Davide A. Mauro, and Davide Rocchesso. A design exploration on the effectiveness of vocal imitations. In *Proceedings Intern. Computer Music Conf. / Conf. on Sound and Music Computing*, pages 1642–1648, Athens, Greece, 2014.

[New04] Fred Newman. *MouthSounds: How to Whistle, Pop, Boing, and Honk... for All Occasions and Then Some*. Workman Publishing, 2004.

[OvEJ14] Elif Özcan, René van Egmond, and Jan J. Jacobs. Product sounds: Basic concepts and categories. *International Journal of Design*, 8(3):97–111, 2014.

[PHL12] Yongki Park, Hoon Heo, and Kyogu Lee. Voicon: An interactive gestural microphone for vocal performance. In *NIME*, 2012.

[RMM16] Davide Rocchesso, Davide A Mauro, and Stefano Delle Monache. mimic: The microphone as a pencil. In *Proceedings of the TEI'16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 357–364. ACM, 2016.

[Sch07] Diemo Schwarz. Corpus-based concatenative synthesis. *IEEE Signal Processing Magazine*, 24(2):92–104, 2007.

[Sch12] Daniel Schlessinger. Concept tahoe: Microphone midi control. In *NIME*, 2012.

[SLF+15] Hugo Scurto, Guillaume Lemaitre, Jules Françoise, Frédéric Voisin, Frédéric Bevilacqua, and Patrick Susini. Combining gestures and vocalizations to imitate sounds. *The Journal of the Acoustical Society of America*, 138(3):1780–1780, 2015.

[SRS+09] Norbert Schnell, Axel Röbel, Diemo Schwarz, Geoffroy Peeters, and Riccardo Borghesi. MuBu and Friends: Assembling Tools for Content Based Real-Time Interactive Audio Processing in Max/MSP. In *Proceedings of the International Computer Music Conference (ICMC)*, Montreal, Canada, August 2009.

[SSCL10] Norbert Schnell, Marco Antonio Suarez-Cifuentes, and Jean-Philippe Lambert. First Steps in Relaxed Real-Time Typo-Morphological Audio Analysis/Synthesis. In *Proceedings of the Sound and Music Computing conference*, Barcelone, Spain, 2010.