How to play with

ALMA

Version 1.1, October 2018

Joachim Heintz

What ALMA does

ALMA receives audio input via a microphone. The input is analyzed for sounding units of four different sizes. Four modes are implemented to play back these units in different ways by a player using a MIDI keyboard.

As extension to the live input, up to seven pre-analyzed sound files can be loaded into buffers, and played back during the performance, too.

The program has been originally designed for spoken word, but it turned out that many other input sounds, in particular percussive instruments, can be nice partners for ALMA.

Needed to play with ALMA

- A **MIDI keyboard** with two octaves, eight pads, sixteen controllers, one ribbon controller. I use Arturia MiniLab, but any other device, or a combination of devices, can be used. About the assignments of key and controller numbers, see the next section *MIDI assigments*. About changing the assignments, see the section *Change MIDI or ASCII assignments*.
- One small **speaker** for the Output. I prefer to use a studio monitor, for instance Yamaha MSP-7. It should be put close to the player.
- One **microphone** for the live audio input. It is also possible to use two microphones and mix them. See *Modify ALMA -> Change input or output assignments.*
- **Audio interface** for 1 (or 2) mic inputs and 1 speaker output, for instance Tascam US-1x2.
- A **Computer** with **CsoundQt** and the ALMA program.

Start playing

1. Set up your real-time audio and select your MIDI device in CsoundQt's *Configure* panel, if necessary.
2. Touch all MIDI controllers once and adjust the numbers to reasonable values, for instance -6 dB for volume.

3. Start the ALMA program in CsoundQt. This does not yet start the performance, but you should see the live input now.
4. Push the START LIVE button to start the performance. Now the live input is being recorded, and analyzed continuously. You can pause and resume this process by pushing the PAUSE / RESUME button.
5. Play with ALMA in pressing the MIDI keys and changing the control parameters. The next sections describe the meaning of MIDI keys, controllers, and ASCII keys.

MIDI assignments

MIDI keys

The four modes are divided each in four regions. Imagine a buffer has recorded 40 seconds, then region=1 means the last (most recent) quarter (sec 30-40 in recording time). Region=2 means recording time 20-30 seconds, region=3 means recording time 10-20 seconds, and region=4 means 0-10 seconds recording time, so the oldest quarter.

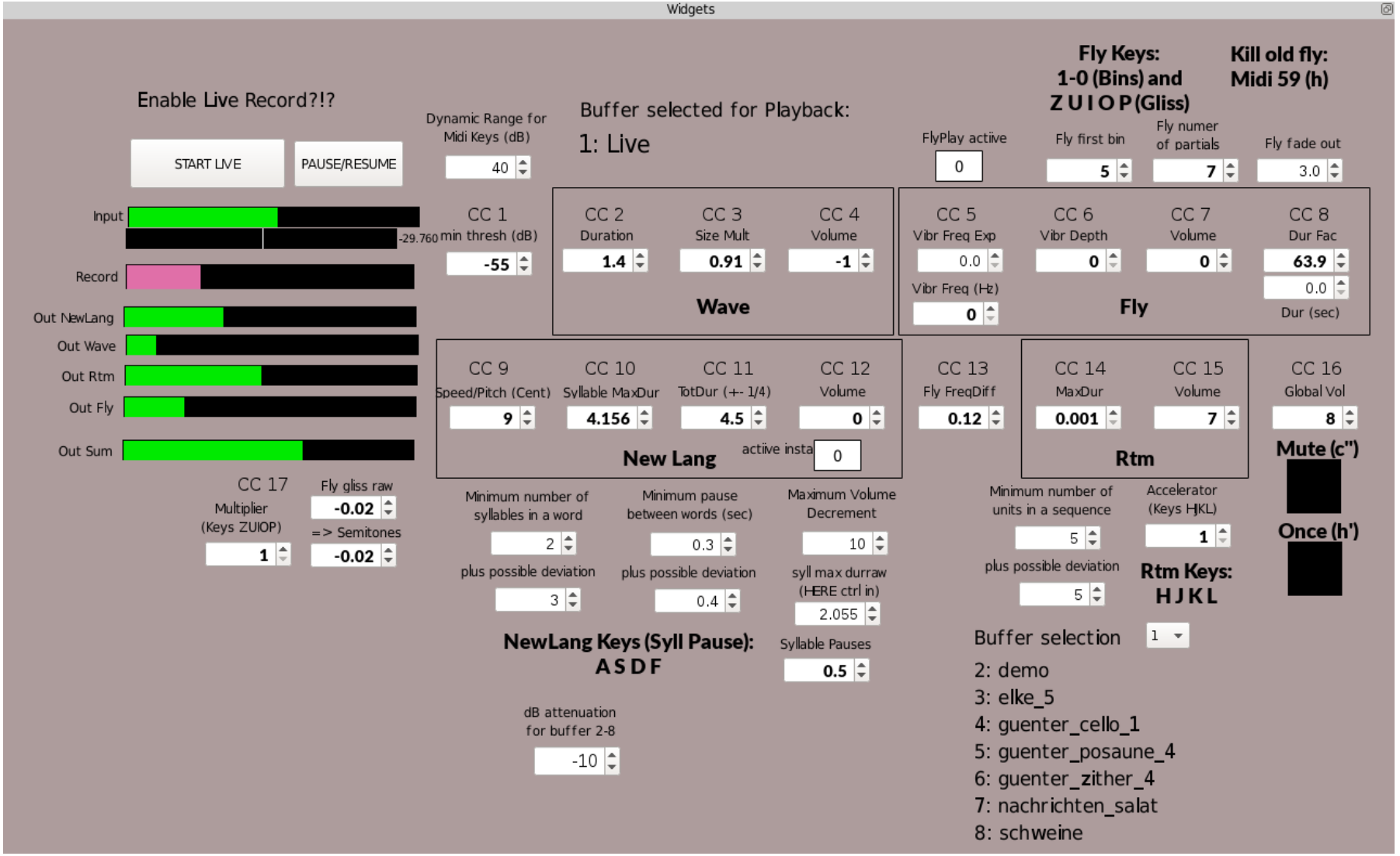
variable name	**note number (default)**	**meaning**
giMidiKeyNewLangReg1	48 (C3)	triggers a sequence of the *NEWLANG* mode, region=1
giMidiKeyNewLangReg2	50 (D3)	triggers a sequence of the *NEWLANG* mode, region=2
giMidiKeyNewLangReg3	52 (E3)	triggers a sequence of the *NEWLANG* mode, region=3
giMidiKeyNewLangReg4	53 (F3)	triggers a sequence of the *NEWLANG* mode, region=4
giMidiKeyRepeatNewLang	57 (A3)	repeats the last sequence of the *NEWLANG* mode
giMidiKeyRepeatNewLang2	55 (G3)	repeats the next-to-last sequence of the *NEWLANG* mode
giMidiKeyWaveReg1	49 (C#3)	triggers a *WAVE*, region=1
giMidiKeyWaveReg2	51 (D#3)	triggers a *WAVE*, region=2
giMidiKeyWaveReg3	54 (F#3)	triggers a *WAVE*, region=3
giMidiKeyWaveReg4	56 (G#3)	triggers a *WAVE*, region=4

giMidiKeyRepeatWave	58 (A#3)	repeats the last *WAVE*
giMidiKeyRtmReg1	60 (C4)	triggers a sequence of the *RTM* mode, region=1
giMidiKeyRtmReg2	62 (D4)	triggers a sequence of the *RTM* mode, region=2
giMidiKeyRtmReg3	64 (E4)	triggers a sequence of the *RTM* mode, region=3
giMidiKeyRtmReg4	65 (F4)	triggers a sequence of the *RTM* mode, region=4
giMidiKeyRepeatRtm	69 (A4)	repeats the last *RTM* sequence
giMidiKeyRepeatRtm2	67 (G4)	repeats the next-to-last *RTM* sequence
giMidiKeyFlyReg1	61 (C#4)	triggers a group of *FLY* sinusoids, region=1
giMidiKeyFlyReg2	63 (D#4)	triggers a group of *FLY* sinusoids, region=2
giMidiKeyFlyReg3	66 (F#4)	triggers a group of *FLY* sinusoids, region=3
giMidiKeyFlyReg4	68 (G#4)	triggers a group of *FLY* sinusoids, region=4
giMidiKeyRepeatFly	70 (A#4)	repeats the last *FLY* sinusoids
giMidiKeyTuornoffOldFly	59 (B3)	removes the oldest playing *FLY* instance (with fade out)
giMidiKeyOnce	71 (B4)	rather than playing a sequence, play only one sound in *NEWLANG* or *RTM*
giMidiKeyMute	72 (C5)	mute / unmute all audio output
giMidiKeySelBuffer1	1 (Pad 1)	select buffer 1 (= live)
giMidiKeySelBuffer2	2 (Pad 2)	select buffer 2
giMidiKeySelBuffer3	3 (Pad 3)	select buffer 3
giMidiKeySelBuffer4	4 (Pad 4)	select buffer 4
giMidiKeySelBuffer5	5 (Pad 5)	select buffer 5
giMidiKeySelBuffer6	6 (Pad 6)	select buffer 6
giMidiKeySelBuffer7	7 (Pad 7)	select buffer 7
giMidiKeySelBuffer8	8 (Pad 8)	select buffer 8

MIDI controller

Controller are directly assigned to CsoundQt widgets (spin boxes). As default, MIDI channel 1 is used.

The assignment of the controller numbers (1-17) can be seen in the widgets panel:



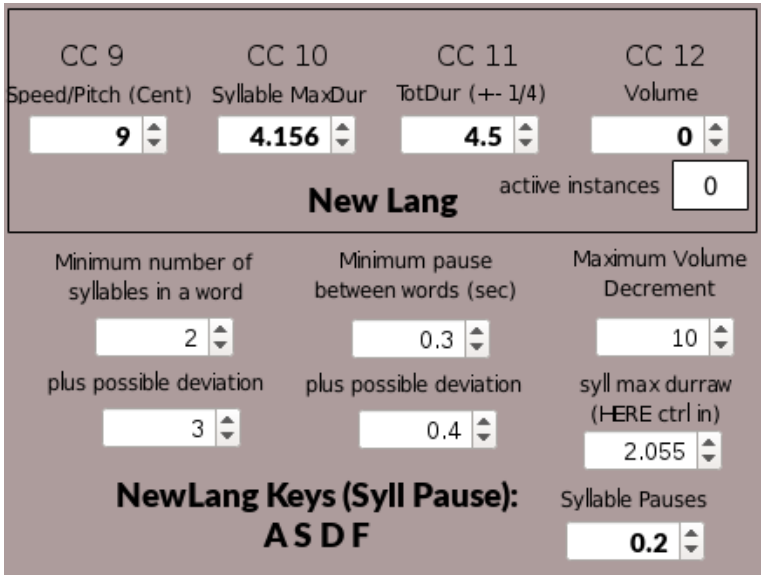
ASCII key assignments

ASCII key	usual key character	meaning
97, 115, 100, 102	a, s, d, f	sets inter syllable pause for *NEWLANG* to 0, 0.2, 0.5, 1 seconds
104, 106, 107, 108	h, j, k, l	sets accelerator for *RTM* to 1 , 2, 3, 5
49, 50, 51, …, 57, 48	1, 2, 3, … 0 (number keys)	sets first bin for *FLY* (1 .. 10)
33, 34, 36, 37, 38, 47, 40, 41, 61, 63	shift + number key	sets number of partials for *FLY* to 1 .. 10
122, 117, 105, 111, 112	z, u, i, o, p	sets multiplier for *FLY* glissando range to 1, 3, 7, 12, 24 semitones

ALMA modes

NewLang

NEWLANG takes syllable-like units and combines them to new "words" and groups of words.



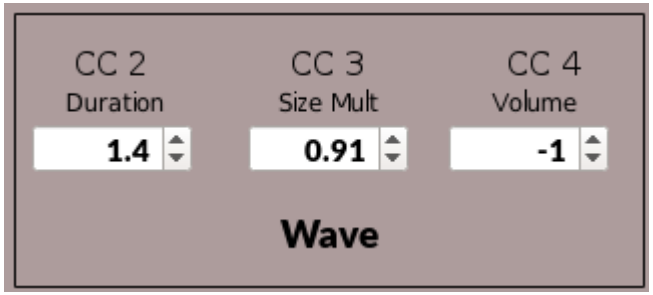
Most parameters should be self-explanatory; here some words about the others:

- **TotDur** sets the total duration of a NewLang sequence. A random deviation is applied so that in maximum the duration can be 1/4 shorter or longer than specified.

  If a **negative** duration is specified, only *one* unit is played back in a way, grains of size *Syllable MaxDur* are created in a way that the overall duration reaches the absolute duration of the given value. For example, -5 seconds as *TotDur* would result in 5 seconds duration. (Usually, values around 0.1 for *Syllable MaxDur* will be chosen to make this negative duration versatile.)

- **Syllable MaxDur** offers the possibility to shorten the "natural" length of a syllable.
- **Maximum Volume Decrement**: if set to 10, a syllable will be randomly attentuated up to 10 dB.

Wave

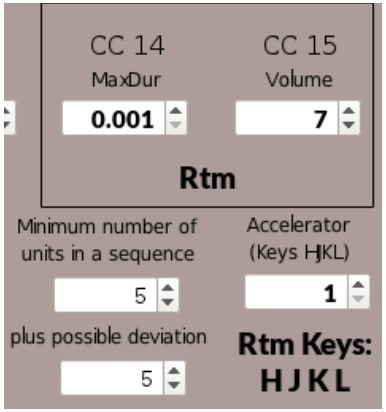


This mode produces seashore-like waves. Internally a special kind of scratching is applied. If the scratching speed (expressed as frequency) is high, the sound is more noisy and higher in pitch; if the scratching speed is lower, the source sound can be recognized better.

Two parameters affect the speed of scratching:

- Region 1 (key c#) has the highest speed (default is 400-800 Hz), region 4 (a#) the lowest (50-100 Hz).
- A **Size Multiplier** (in the range -5 .. 5) modifies these values by a factor of 2^*SizeMult*.

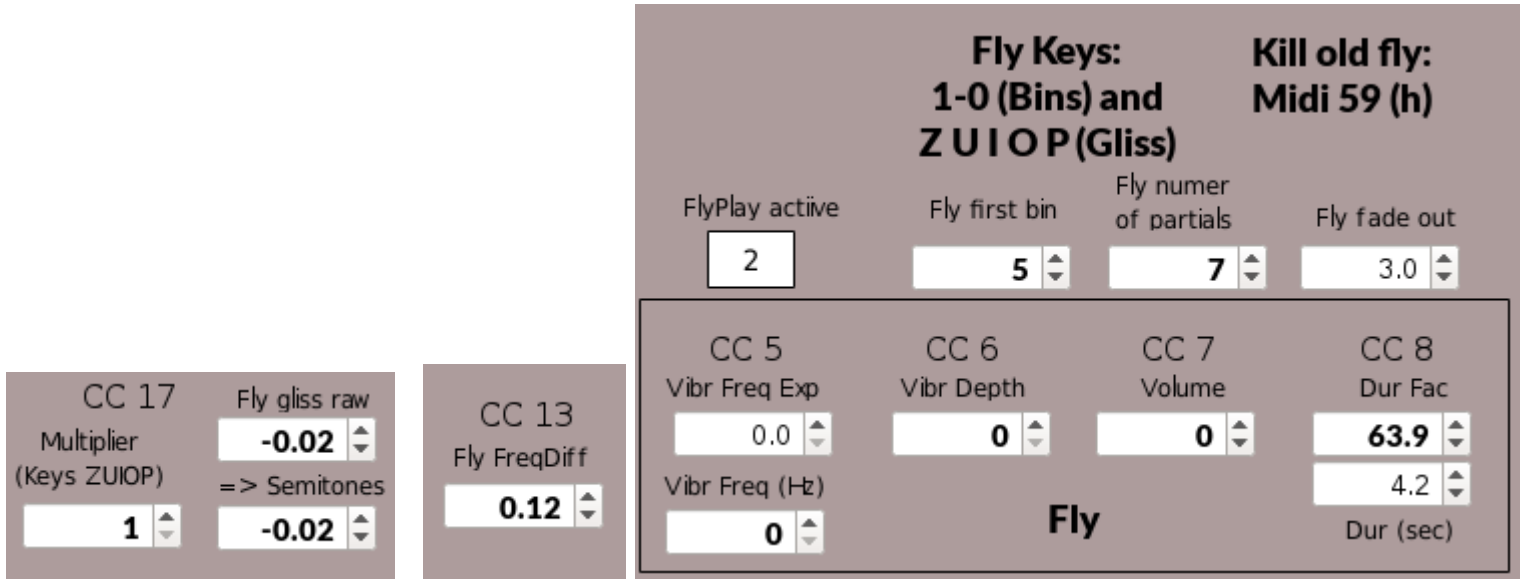
Rtm



This mode reads the proportions in an array — by default [1/2, 2/3, 1/3, 1/4, 3/4, 1], whereas 1 means one second — and first multiplies these values by a randomly chosen number from a second array — by default [1, 3/2, 2]. In the example above, a sequence between 5 and 10 notes will be generated, choosing of of the proportions times the (constant) multiplier for the sequence.

The player can decide to apply an accelerator (1=normal, 2=twice as fast, etc.) via the ASCII keyboard.

Fly



This mode takes one unit, analyzes via FFT the N strongest partials (default = 20), and resynthesize a number of them, set in **Fly number of partials**. The "**Fly first bin**" parameter gives an offset in the resynthesis; the setting which is showed above would resynthesize 7 partials, starting from the fith loudest bin. These are the other parameters:

- **Dur Fac** (CC 8): A multiplier which results in massive time stretching. The default range is 1 (= original duration of the unit) up to 1000. The box below shows the real time of the selected unit.
- **Fly FreqDiff** (CC 13): A factor which is applied to the frequency deviations of the resynthesis. For instance 0.1 would mean that the frequency deviations are only 1/10 of the natural deviations.
- **Fly gliss raw** (CC 17 = ribbon): Applies a continuous pitch shift (glissando). The multiplier results in the range of semitones for the maximum frequency shift.
- It is possible to apply a simple FM. The **Vibr Freq** and **Vibr Depth** are related to this feature.

As sounds which are triggered by this mode can be very long, it can be necessary to turn off running instances. Once MIDI key 59 is pressed, the oldest instance is turned off with a fade out which is set as **Fly fade out**. The box **Fly play active** shows the number of active instances of this instrument.

Modify ALMA

Change input or output assignments

Other input or output channels than the dafault channel 1 can be assigned in the code at the very beginning (USER SETTINGS). If two microphones should be mixed as input, look around line 423 and remove the semicolon for these two lines:

```
;aIn_R inch 2
;gaIn += aIn_R
```

Change MIDI or ASCII assignments

MIDI keys can be changed in the section MIDI KEYS (around line 90 of the code). MIDI controllers can be changed in the GUI (in the properties of a widget). ASCII keys can be changed in the instr ReceiveAsciiKey (around line 750).

Change MIDI device / Add or reduce parameters

If you have a device with less controllers, you must decide which parameters not to control in real time. Then just change the controller assignment in the GUI. Adding new parameters should be straightforward.

The ALMA program

Implementation

ALMA is written in Csound and uses CsoundQt as frontend (for the GUI). It basically needs two files for minimal usage: A **.csd file** (called for instance 181003_alma.csd, and a collection of user defined opcodes called **alma2.udos**.

If prerecorded sound will be used, they must analyzed in advance, and put in a certain folder structure.

Folder structure

In the main ALMA directory the program is organized like this (+ means folder):

main program file (xxx_alma.csd)
alma2.udos
+ buffer
  + marker (containing 4 text files for each sound file in *wav*)
  + wav (mono sound files which can be loaded into the buffers for an ALMA performance)
+ buffer_selection (containing up to 8 text files with selections of buffers for a performance)

+ *import_to buffers* (with a shell script which imports and analyzes samples and writes wav files and markers in the *buffer* folder)
+ *select_buffers_for_performance* (containing a shell script which writes a file to the folder *buffer_selection*)

Import new sound files

Run the script *import.sh* in the folder *import_to_buffers*.

Select buffers for performance

Run the script *select_for_performance.sh* in the folder *select_buffers_for_performance*.

Thanks

ALMA is a project which depends on her partners for playing. Just to name some of the important partners: Laureline, Elke, Günter, Michael, Shaghayegh. Special thanks to Amin who motivated me to write this documentation ...