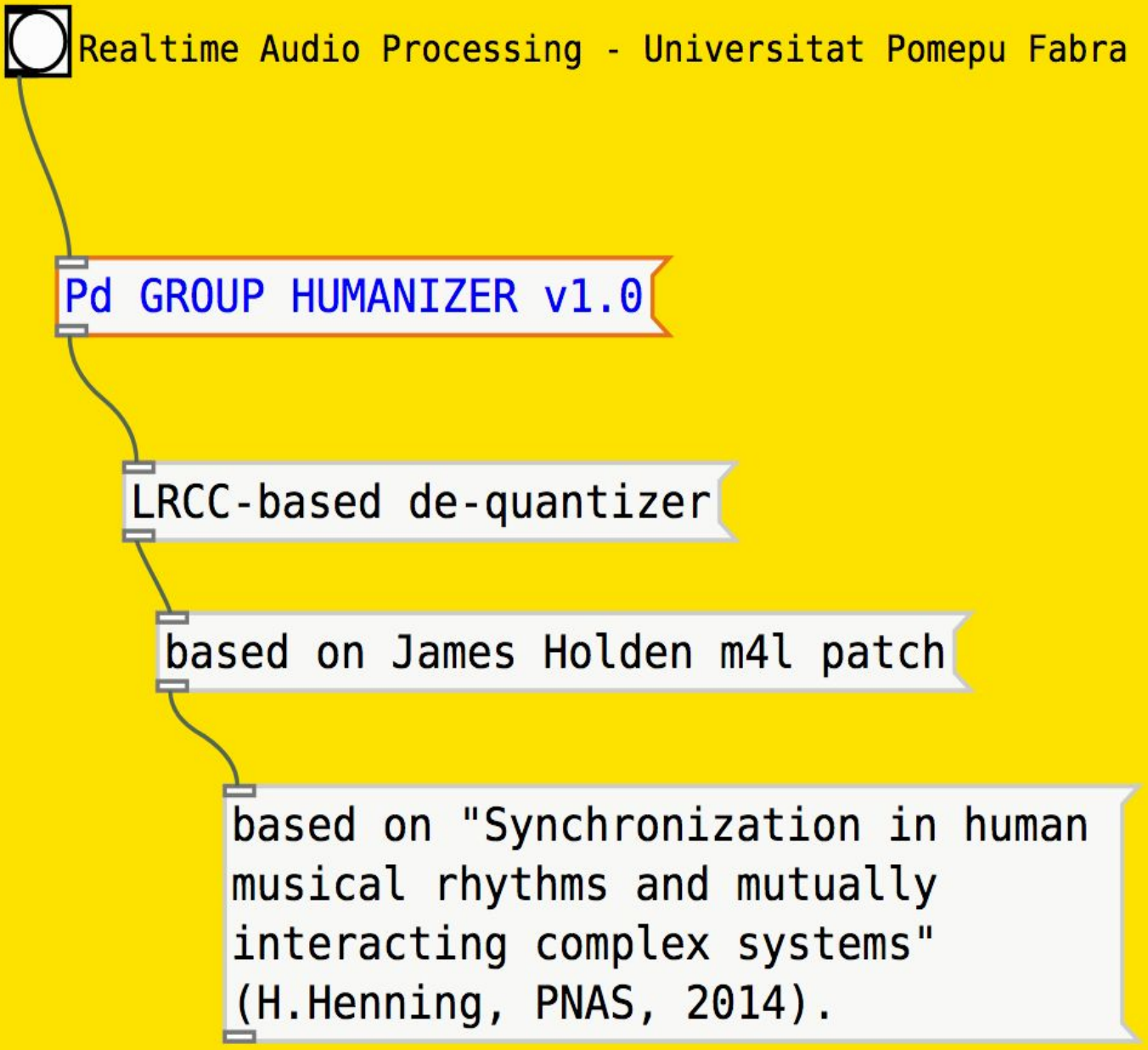


Enric Gusó
Dídac Gascón
Sergio Gor

Quick-start Guide

1.-Introduction: About Pd's Group Humanizer

The Group Humanizer is a Max for Live plug-in created by James Holden on 2015. Based on research from Harvard scientists, Holden built a Max for Live device which automatically shapes the timing of your MIDI channels, injecting the organic push-pull feel you can only get from human performance. In fact, Holden introduced the patch into his live show, allowing his modular synthesizer to follow the shifting tempo of his live drummer.

Holden's patch has two modes:

- **Nudge tempo:** it moves Live's playback in order to set it into the musicians tempo.
- **Metronome:** musicians are forced to come back to original tempo.

It is only available for Ableton Live so the idea behind this project was to make it available for any DAW. In this first version 1.0, we present one of the modes: the metronome mode.

This patch pushes and pulls MIDI notes in order to make them more "human", taking into account other channel's errors and trying to adjust their tempo to other channels' tempo. When they drift too far from original metronome, they are forced to come back smoothly.

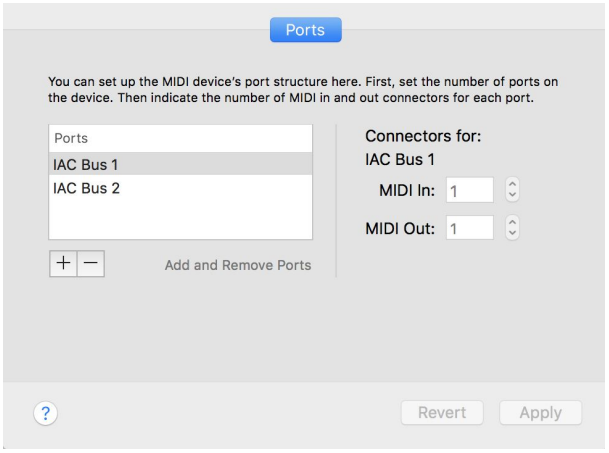
To run it, it is necessary a copy of Pure Data-Extended (or Purr Data) installed and a DAW able to receive MIDI notes from a virtual MIDI port.

2.-Setting-up IAC Driver & PureData

The patch humanizes the 16 midi channels you get into Pure Data. MIDI notes are sent from your DAW/MIDI sequencer to PureData, and then, back to Pd's MIDI out, so you can route them anywhere you want. In a basic setup, from DAW to DAW, two virtual MIDI ports are needed (send & receive).

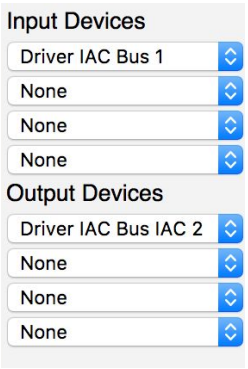
The patch humanizes the 16 midi channels you get into Pure Data. MIDI notes are sent from your DAW/MIDI sequencer to PureData, and then, back to Pd's MIDI out, so you can route them anywhere you want. In a basic setup, from DAW to DAW, two virtual MIDI ports are needed (send & receive).

- **macOS:** Utilites >> Audio MIDI Setup >> Window >> Show MIDI Studio >> IAC Driver



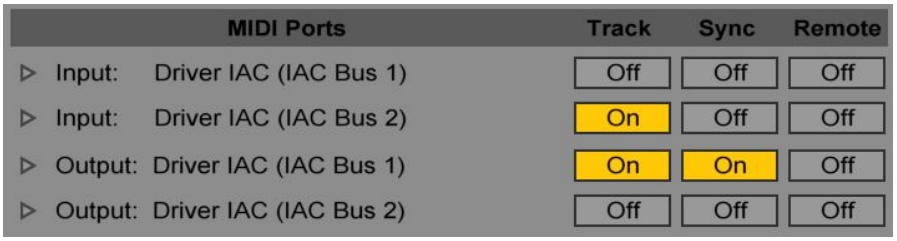
- **Windows:** it does not have own virtual MIDI ports. It is possible to install drivers which allow operate with them. For example: Virtual MIDI for Windows - Tobias Erichsen
- **Linux:** Ports can be created using: PortMidi - PortMedia

Once virtual MIDI ports are created it is necessary to set up PureData. Go to: Preferences >> MIDI and set first driver as a input, and second as a output.

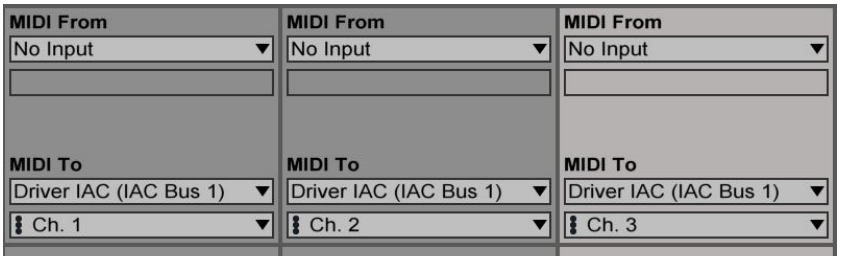


3.-Setting-up your DAW

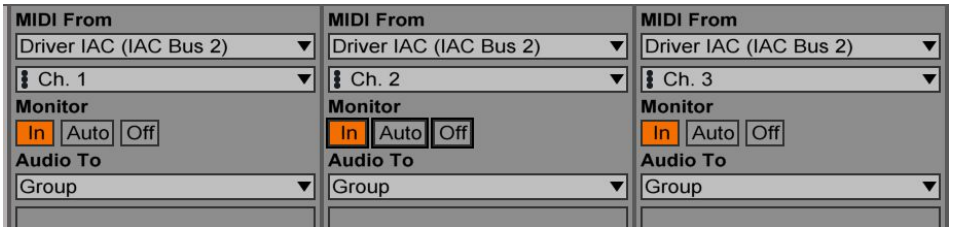
Just set one of your virtual MIDI ports as output (route them to Pd), but with MIDI SYNC enabled. Then, activate the second MIDI port as a input (receives delayed MIDI notes from PureData). To do this, go to Preferences and MIDI options of your DAW.



Finally, you have to set your MIDI tracks to output's MIDI port and the desired instruments to input's MIDI port. An example is shown:



MIDI SENDS



MIDI RECEIVES

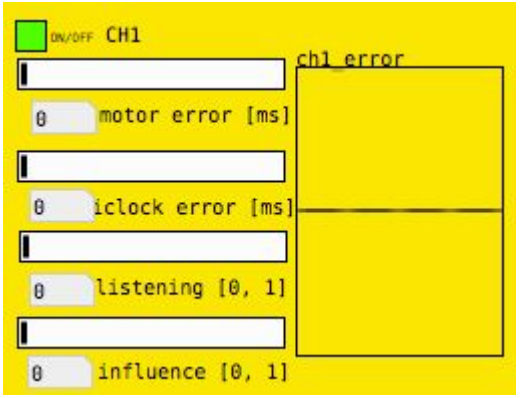
Finally, the patch uses a latency buffer in order to pull notes "to the past". You can compensate this latency setting **-85ms in both your sends and receives** in your DAW track settings, syncing with DAW's click.

4.-Operating Instructions:

1.-Open GROUP_HUMANIZER_GUI.pd
2.-Turn on the MIDI channels you want to humanize
3.-Open your DAW/sequencer and start playing
4.-Rise motor and iclock errors gradually to the desired effect

A good starting point would be:

- MOTOR ERROR of 0.5ms
- TIMING ERROR of 5.0ms
- LISTENING ratio of 0.5
- INFLUENCE ratio of 0.5
- Threshold to 20ms



Next, let's take a brief look at the different parameters:

Motor error: There is a short time since the brain gives the order to play a note, until the hand (or whatever triggers the note) plays the instrument. It is a non-correlated error with previous motor errors and other musicians errors. Its value is calculated randomly following a gaussian distribution. In the patch we can control the motor error by selecting a value in milliseconds [ms]. What we are changing is the standard deviation of the gaussian distribution, which basically tells us how good is the musician. If it has values around 0-0.5 [ms] we will be emulating an exceptional musician, while if we set the value around 5 [ms] we'll be closer to a regular one.

Iclock error: Nobody has a perfect metronome in their brain, not even professional musicians. This value represents how much can one musician rush to the next note, making all other musicians to follow him but drifting from the click track. Its value goes from 0ms to 10ms and tells us the maximum deviation of the tempo that can be reached in a given note. Low iclock error should keep the tempo pretty attached to the original one, whereas high errors will lead them to drift from the click quickier.

Listening ratio: Musicians that play together may be listening one to each other. If the drummer suddenly starts to play faster the other musicians, they immediately play fast too. Listening ratio is this amount of awareness, how much will I be affected if another musician deviates from the original tempo. Mathematically we can easily describe this fact as how much my error is correlated to other musicians error. Low value means a non listener musician while high listening value means that the musician is paying a lot of attention to the performance of the other musicians and the group will be more prone to drift from the click.

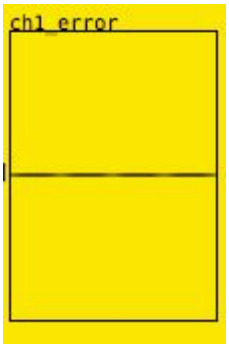
Influence ratio: Some instruments are more influential than others at keeping the tempo. A not precise drummer can make a song turn into chaos, but a atmosferic mandoline rushing might not be dramatic for the song tempo. This relation between instruments is what influence ratio intends to represent. The influence value of a given musician works like the volume at which it's send to the virtual aux send. Two same errors on an instrument with two different influence values do not affect equally, the one with the higher influence will affect more on the error of the instrument, although the two errors were equal. The ones more important like drums or bass should have higher values, while the ones less important, like a synth pad, should have lower values.

And finally, the patch has other relevant features:

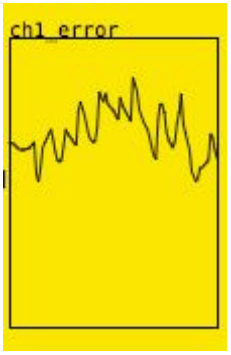
ON/OFF: When we have assigned a virtual midi port to a channel and pure data is already receiving notes, the on/off button applies the delay that the program calculates. It is important to keep in mind that the active channels must be consecutive and the first channel must be channel 1. If we do not do this, the spread between musicians won't be calculated properly. This will be solved in future releases.

Ch_error: We can see graphically the error of a single musician in the canvas next to the parameter channel. This give us relevant information about the error. A straight line tells us that there is no error on the musician, it follows perfectly the tempo. A high value error means that the musician is rushing as the values get high. And a dragging musician tends to have the error values low.

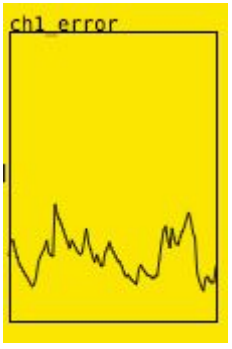
On tempo



Ahead tempo



Behind tempo



Sending Metronome: This blinks when the mean of all the offsets has reached the threshold, activating the mechanism of sending the metronome to the musicians instead of the other tracks, so they go back smoothly to the original tempo.

Metronome threshold: The default value of this threshold is 20 ms but the user can change this value between 0 and 30 ms, and represents the allowed deviation from the original click. If you're making music for DJs, you should be closer to 10ms than to the maximum, 30ms.

Force Resync: This does something similar to sending metronome but only when the user presses it. It empties the error integrators and forces all musicians to go straight to the original tempo, without a smooth transition.

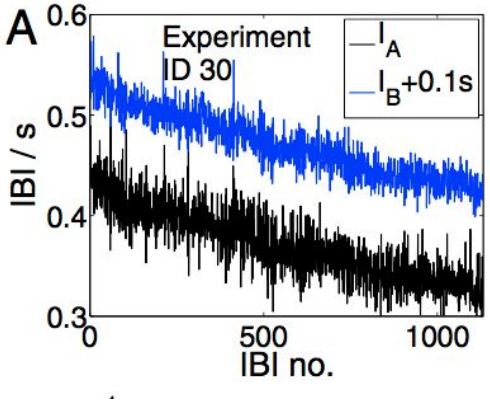
Mean: It represents the distance between the current tempo center and the original click in ms. Its value is needed for the metronome threshold calculation.

Spread: In this canvas we can see, for each note, the time difference between the musician which is more ahead from the click with the one that is dragging the most. When zero, it means that all musician have played at the exactly same time, without implying being centered to the click.

Appendix 1: Theoretical background

This patch is based on the James holden's Program alongside H.Henning's paper "Synchronization in human musical rhythms and mutually interacting complex systems" (H.Henning, PNAS, 2014). The paper introduces two concepts fundamental to understand how we have developed this patch.

MICS: It stands for mutually interacting complex systems. And basically describes the statistical approach of the ways musicians interact. When two musicians play they do not play independently. The way one play affects the other, and equally otherwise. The observation of how this complex systems interact has concluded in a type of correlation that lead us to the second concept: LRCC.



LRCC: The correlation that two musicians have playing together is called long-range cross correlation. To demonstrate this fact Henning performed an experiment with two musicians. They were both seated in a MIDI piano playing periodically a note over a thousand times. The results came up with the figure above. We can clearly see how they deviate parallelly, and that the gradient stays pretty equal in both.

The paper presents two main equations:

1) The offsets of the two musicians will follow the sum of a gaussian and a 1/f distributions, using the last value of the drift

$$I_{A,n} = \sigma_A C_{A,n} + T + \xi_{A,n} - \xi_{A,n-1} - W_A d_{n-1}$$
$$I_{B,n} = \sigma_B C_{B,n} + T + \xi_{B,n} - \xi_{B,n-1} + W_B d_{n-1},$$

2) This drift is integrated in time, producing the LRCC:

$$d_n = t_{A,n} - t_{B,n} = \sum_{j=1}^{n} (I_{A,j} - I_{B,j}),$$

Appendix 2: James Holden's approach

Hennig's early paper was based on a simple premise: two musicians trying to match their internal clocks at a given note length. We've known that they both worked together into generalizing this to many musicians, and reading Holden's patch we can synthesize the model in the next equation:

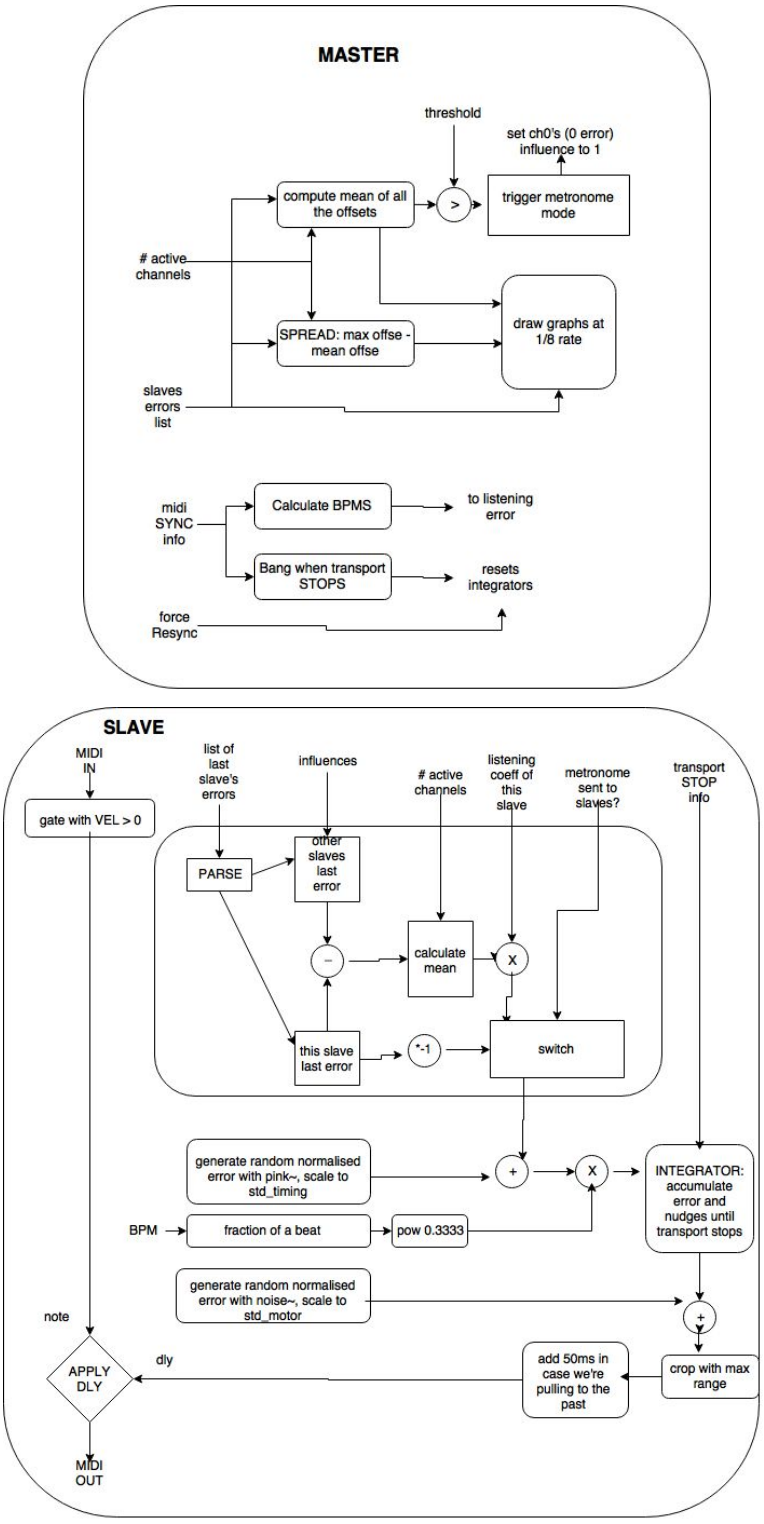
$$I = \sum_{i=1}^{m} \sum_{j=1}^{n} \left\{ \xi_{A,j} + \sqrt[3]{Fob}[\sigma_A C_{A,j} + W_A \frac{d_{i,j} - \sum_{i-1}^{m} Z_i d_{i,j-1}}{m}] \right\}$$

$\sum_{i=1}^{m}$	For each musician
$\sum_{j=1}^{n}$	For each note
$\xi_{i,j}$	Motor error
$\sqrt[3]{Fob}$	Fraction of a beat cubic rooted. Relates the amount of error to the duration of the note.
$\sigma_i C_{i,j}$	Internal clock error
W_i	Listening coefficient
$\frac{d_{i,j} - \sum_{i-1}^{m} Z_i d_{i,j-1}}{m}$	Arithmetic mean depending on the number of active channels
$d_{i,j}$	This musician last error
Z_i	Influence value of the other musicians
$d_{i,j-1}$	Last error of the musicians

Appendix 3: Implementation details

a3.1.-PATCH STRUCTURE

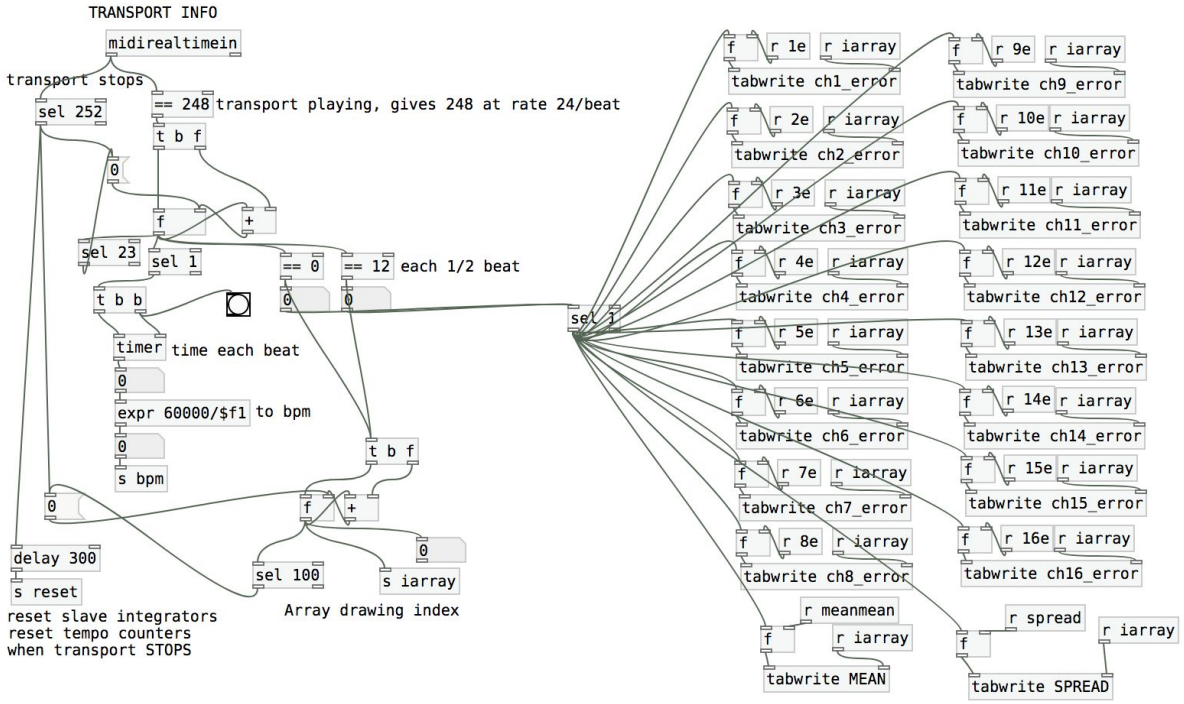
Here we attach the block diagrams of the patch:



a3.2.-MIDI SYNC

As you can see in the section above, one needs to know two things about what is happening outside of Pd: the tempo, for computing the note length/rate, and when the transport stops for resetting the integrators, which means clearing the memory or resetting the algorithm.

This has been done using basic MIDI capabilities in order to make it compatible with any DAW, through Pd's *midirealtimein*



Basically: when transport is playing, *midirealtimein* outputs 248 value at a rate of 24 times per beat, allowing us to estimate the BPMs with some accuracy limitations due to Pd's timer. In the same manner, when transport stops *midirealtimein* triggers the value 252, allowing us to empty the integrator or to send NOTE OFFs for avoiding the typical hanged note glitch. In this same section, we've also used transport information for drawing the graphs at a ⅛ note rate.

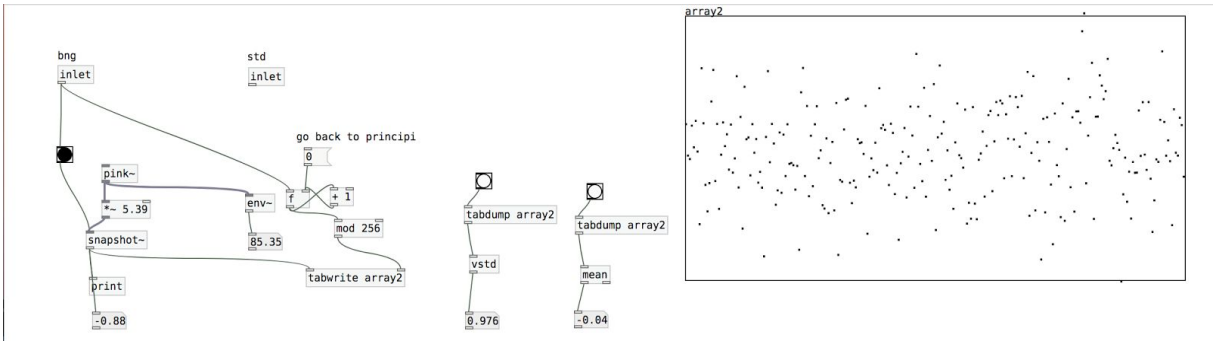
a3.3.-DATA HANDLING

In contrast with Holden's implementation, we've chosen tables as the memory for the algorithm. All fader values, and all last offsets, are stored in tables with 17 cannels, 16 for the actual virtual musicians and ch0 for the virtual metronome. You can monitor them in *create_tables* patch.

Offsets can be positive or negative, so we've added a 50ms buffer to the delay line (object *pipe*), which adds to the regular latency and gives the -85ms previously stated.

a3.4.-ERROR GENERATORS

For each slave, one has to generate pseudo-random errors for motor and internal clock series. This is done with white and pink noise generators, with an added normalization for ensuring 0 mean and standard deviation 1. Otherwise, musicians will drift from the others.



Trial and error process of normalizing pseudo-random generators to mean=0 and std=1

a3.5.-Mean, Spread, and Metronome mode

Forcing influence to be 0 when channels are OFF, we ensure that all active channels that matter will have a non-zero influence. This way, we can obtain the number of active channels which will be used in the next two calculations.

Mean: the sum of all last offsets / #active channels
Spread: we trim the offsets' list to the number of active channels, and then we compute max-min.

The active channels' list has been implemented with *bondo*, forcing the list to get updated each time one of the values change.

Finally, the current slave error calculation has two modes:

The one in regular mode, where the equation of Appendix 2 is applied:

```
expr
listenings[$f1]*(errors[1]-errors[$f1])*influences[1]+(errors[2]-errors[$f1])*influences[2]+(errors[3]-errors[$f1])*influences[3]+(errors[4]-errors[$f1])*influences[4]+(errors[5]-errors[$f1])*influences[5]+(errors[6]-errors[$f1])*influences[6]+(errors[7]-errors[$f1])*influences[7]+(errors[8]-errors[$f1])*influences[8]+(errors[9]-errors[$f1])*influences[9]+(errors[10]-errors[$f1])*influences[10]+(errors[11]-errors[$f1])*influences[11]+(errors[12]-errors[$f1])*influences[12]+(errors[13]-errors[$f1])*influences[13]+(errors[14]-errors[$f1])*influences[14]+(errors[15]-errors[$f1])*influences[15]+(errors[16]-errors[$f1])*influences[16]
```

And the Metronome mode mode, when virtual monitor send is substituted for a zero-error click.

```
expr -errors[$f1]*0.5
```