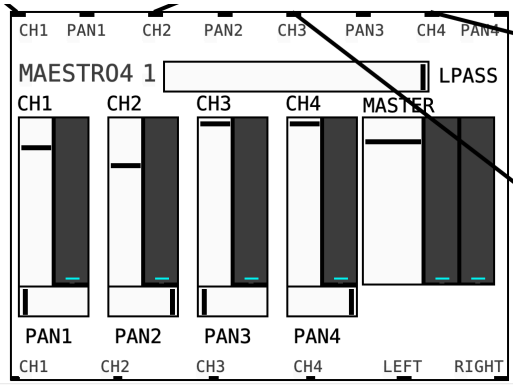
21M.370 Digital Instrument Design
Lab assignment 6 - New Capacit Synth

Assignment description:

Today we are exploring creating a new synth in automatonism and modifying our mapping in python to control it.

Automatonism notes



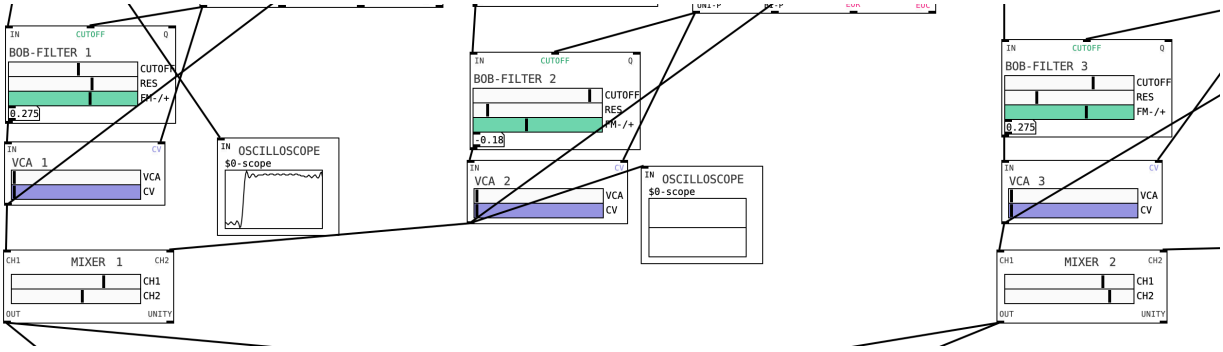
Essential Automatonism objects:

“Maestro4 1”

- sends audio to your computers DAC
- must be instance 1 to work with the mixer in capacitCtrl.pd

VCA

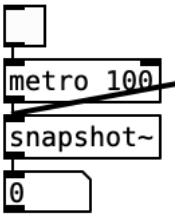
- control the amplitude of any signal
- VCA 1-4 are predefined as to control the ouput amplitude of four voices
- create a new PD object (cmd or ctrl-1 then type ‘VCA 1’ for instance 1)
- this lets you define the instance number
- creating objects using the automatonism interface gives random instance numbers



scope

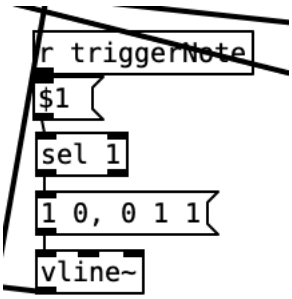
- creates an oscilloscope to let you monitor output signals
- only works for signals from -1 to 1

snapshot~



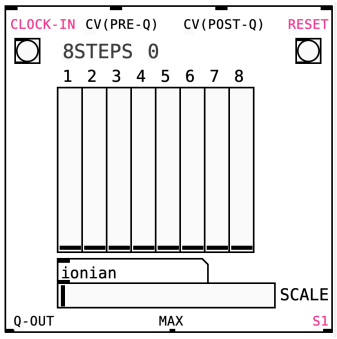
- lets you view the raw value of an output
- requires a toggle and metro object to set how often you are taking a snapshot of the signal
- connect both the metro and the signal you want to capture to the left inlet of the snapshot

Turning an OSC message into a trigger



- the ‘receive’ or ‘r’ object will receive messages sent to that address
- in Python the line
  sendOSC("triggerNote", num, num, num)
  will send the values (num,num,num) to the receive object names ‘triggerNote’. Only the first value is used here due to the message box with $1 in it.
- the output of the vline~ object is a signal that goes to:
  [1 0, 0 1 1] which we can read as ‘go to 1 in 0 ms, then go to 0 in 1ms after waiting for 1 ms’

Working with pitches



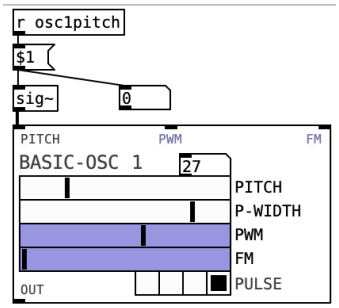
- CV signals meant for pitch information need to be quantized so they conform to the notes of a scale. Many objects already have a quantized CV output, like the Q-out of the ‘8steps’ object.
- We can also use the ‘quantize’ object to quantize any signal. This will only output signals quantized to generate the pitches of a scale.
- The normal representation of pitch is MIDI note values, which range from 0-127, where each integer represents a pitch, e.g. MIDI note 60 is middle

C (which is also C4 where the 4 represents a specific octave), 61 is C#4, 62 is D4, 63 is D#4, 72 is C5.

- In automatonism, we represent MIDI notes as a range from 0-1 with the simple formula `quantizedCV = MIDInote/127`.
- From python, we have several options:
  1. set the frequency of a VCO directly by sending a value from 0-127:

```
instanceNum = 1
midiNote = 60 #can be from 0-127
sendOSC('basic-osc', instanceNum, 'PITCH',
midiNote)
```

  2. send a frequency to an oscillators PITCH input:



     - the pitch in this case is from 0-1

```
pitch = 60/127 #from 0-1
sendOSC('osc1pitch',
pitch,pitch,pitch)
```

     - note that sendOSC requires four arguments, so we send pitch three times and then only use the first pitch value by using the message box to select $1.

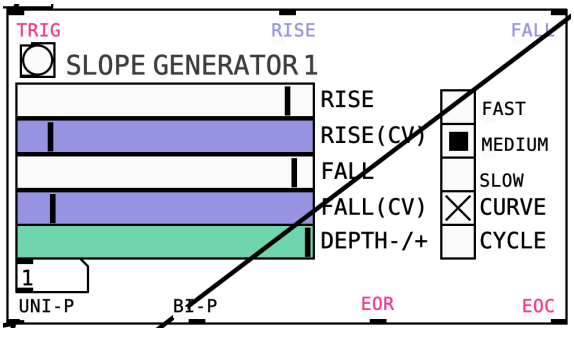
Python notes

To send data to PD you can just use the senOSC function:

sendOSC('module-name', instanceNum, 'PARAM_NAME', val)

Where:

- 'module-name' is the first symbol of the name of the module. In PD this will be in all-caps, but for sendOSC it needs to be all lower-case.
- for most objects this will be obvious but some are trickier, e.g. the slope generators name is just 'slope'.
- same for the parameters - they should



be as written, always in all-caps.

- But sometimes it isn't obvious - for example, the paramNames for slope are:
  RISE
  RISE(CV)
  FALL
  FALL(CV)
  DEPTH-/+
  CURVE
  CYCLE
  RADIO
  TRIG
- if you need help figuring it out you can:
  - right click on any module
  - select 'open'
  - look for a sub-patch called 'external-messages and double-click it
  - the module is name is in the to 'receive' or 'r' object
  - the paramNames are in the 'route' object: