Mobile Music Source Documentation

SOPI Research Group. March 2012

Coding Practice

Workbenches

- each module should have a MWE (Minimum Working Example) Workbench, to demonstrate and test functionality during integration and debugging.
- The workbench should allow debugging by manipulating the publicly available methods, sends and receives mentioned in the API/documentation.

Naming conventions for integration

- module names should be ‘camelized’ e.g. ‘ThisNameIsCamelized’.
- publicly available ‘methods’, values and sends should have camelized, descriptive identifiers. e.g. ‘thisMethodOutputsZeroWhenGivenOne’
- module folders names should end with the word `module’ and contain a Pd patch inside with the same name for use in the masterControlPatch.
- documentation should include both the variable names, their ranges and a short description of their use mentioning any cautionary usage issues.
- receivers for the master parameter file are prepended with ‘#’, are capatalized and take the name of the destination module. e.g. ‘#SYNTHESIZER’

Definitions

- BOOL is considered to be ‘zero’ and ‘not zero’

Comments

- Comments should be used to clarify working logic withing patches where needed and when the patch is not self documenting or a MWE is not sufficient to demonstrate full functionality.

‘Source Management’

- module backups are placed in the ‘xbackups’ folder within each module folder and are zipped and appended with the date of their creation e.g. ‘synthesizerModule010312.zip’

Sensor Module

Analysis Module

Adaptive System Module

This module uses rescaled probabilities to direct or adapt the state selection within a Markov chain. The rescaling is occurs in relation to social parameters. But this can be mapped to any sensor value that is normalised to the range 0-1(float).

Output is to #STATE.

Each chain sends a separate message in the following form:

```
<Markov chain Name(unique Symobl)> <Next State (int)>
```

Markov Chain state transition probabilities are

State Interpretation Mapping Layer [SIMpL]

What this does:

- Receives state information from the Markov Chains.
- Interprets the required action to take from the state information
- Outputs this information to the Synth Stuff layer.

Information from Markov Chains

Markov chains will send a message to SIMpL that contains the chain that the message came from and the state value.
Message structure:

```
<Markov chain Name(unique Symobl)> <Next State (int)>
```

Interpretation of States

This uses the information on static and adaptive system state behaviour and updates the synthesis parameters through out the system so that they correspond to the markov state. It is also used to set static behaviour.

This retrieves and interprets messages in 2 forms. The first is for updating the parameter to a constant value which is done by the following message:

```
<Markov chain Name(unique Symobl)> <Next State (int)> …
          … <synth module (unique Symbol)> <synth parameter (symbol)> <value
       (float)>
```

This message specifies the corresponding system state that this parameter change should occur in. The synth module that is to be updated. The parameter that is to be updated and the value it should be updated to.

The other form is to change the mapping of a sensor to the variable which is done with the following message:

```
<Markov chain Name(unique Symobl)> <Next State (int)> …
     … <synth module (unique Symbol)> <synth parameter (symbol)> …
        … <minimum value (float)> <maximum value (float)> …
           … <sensor value to map (symbol)> <min value (float)> …
             … <max value (float)> scaling <curve exponent (float or symbol dB)>
```

This message specifies the corresponding system state that this parameter change should occur in. The synth module that is to be updated. The parameter that is to be updated the minimum and maximum values for the parameter. The sensor value that should be mapped to this parameter, the minimum and maximum values the sensor value can be. The scaling exponent for the behaviour of the sensor mapping. 1 being linear and dB specifying decibel scale.

Specifying Static Behaviour

Output

SIMpL outputs messages down #SYNTHESIZER in the format

```
       <synth module (unique Symbol)> <synth parameter (symbol)> <value (float)>
```

Synthesis Module

Module Public Send and Receive Documentation

Social/Gestural Analysis Module

all messages are sent on the #ANALYSIS `send channel'.

EXACT VARIABLE NAME	RANGE/UNIT
centroidXPosition	0-1f
centroidYPosition	0-1f
distanceBetween1and2	0-1f
distanceBetween1andCentroid	0-1f
distanceBetween2and3	0-1f
distanceBetween2andCentroid	0-1f
distanceBetween3and1	0-1f
distanceBetween3andCentroid	0-1f
everybodyIsAlive	BOOL
averageDistanceToCentroid	0-1f
deltaIPod1	>0
deltaIPod2	>0
deltaIPod3	>0
crossCorrelationDIFFERENCE	0-1f
crossCorrelationSUM	0-1f
iPod1Roll	
iPod2Roll	
iPod3Roll	
iPod1Pitch	

iPod2Pitch	
iPod3Pitch	
iPod1Magnitude	
iPod2Magnitude	
iPod3Magnitude	

Social/Gestural Analysis Module Details

- **centroidXPosition:** the current X coordinate of the centroid. Updated only if all three people are `alive'
- **centroidYPosition:** the current Y coordinate of the centroid. Updated only if all three people are `alive'
- **distanceBetween1and2:** the distance between persons 1 and 2.
- **distanceBetween1andCentroid:** the current distance between person 1 and the last calculated centroid position
- **distanceBetween2and3:** the distance between persons 2 and 3.
- **distanceBetween2andCentroid:** the current distance between person 1 and the last calculated centroid position
- **distanceBetween3and1:** the distance between persons 3 and 1.
- **distanceBetween3andCentroid:** the current distance between person 1 and the last calculated centroid position
- **everybodyIsAlive:** a boolean which is high if all three people are `alive', low otherwise.
- **averageDistanceToCentroid:** the average distance of all three people to the current centroid. Updated only if all three people are `alive'
- **deltaIPod1:** the current rate of change of iPod1's total acceleration magnitude.
- **deltaIPod2:** the current rate of change of iPod2's total acceleration magnitude.
- **deltaIPod3:** the current rate of change of iPod3's total acceleration magnitude.
- **crossCorrelationDIFFERENCE:** the current cross-correlation between all three people. this value is calculated by looking at the difference of the cross-correlation values between each of the player combinations
- **crossCorrelationSUM:** the current cross-correlation between all three people. this value is calculated by looking at the sum of the cross-correlation values between each of the player combinations
- **iPod1Roll:** iPod1's roll
- **iPod2Roll:** iPod2's roll
- **iPod3Roll:** iPod3's roll
- **iPod1Pitch:** iPod1's pitch [angle, not 'frequency']

- **iPod1Pitch:** iPod2’s pitch [angle, not ‘frequency’]
- **iPod1Pitch:** iPod3’s pitch [angle, not ‘frequency’]
- **iPod1Magnitude:** iPod1’s total magnitude of acceleration on its 3 axes
- **iPod2Magnitude:** iPod2’s total magnitude of acceleration on its 3 axes
- **iPod3Magnitude:** iPod3’s total magnitude of acceleration on its 3 axes

Synthesizer Module

all messages are sent on the #SYNTHESIZER `send channel'.

EXACT VARIABLE NAME	RANGE/UNIT	DEPENDENCIES
slowMetroRate	>20 [ms]	-
fastMetroRate	>10 [ms]	-
masterFrequency	[Hz]	-
minMasterFrequency	[Hz]	SIMpL
maxMasterFrequency	[Hz]	SIMpL
masterFrequencyScale	[Hz]	SIMpL
stepSize	n/a	-
vibratoRate	[Hz]	-
minVibratoRate	[Hz]	SIMpL
maxVibratoRate	[Hz]	SIMpL
vibratoRateScale	[Hz]	SIMpL
vibratoSize	0-1 [float]	-
randomFrequencyMagnitude	0-1[float]	-
randomFrequencySpeedHighOrLow	bool	fastMetroRate, slowMetroRate
masterAmplitude	0-1[float]	amplitudeResponseCurve
amplitudeResponseCurve	0-1[float]	-
curvedOrLinearADSR	bool	amplitudeResponseCurve
noteOnOff	0-1[float]	-
oscillatorPreset	0-14[int]	-
noiseLevel	0-1[float]	masterAmplitude [adsr]
reverbMix	0-1[float]	-

reverbPreOrPostMasterAmplitude	bool	-
ADSRattack	>0[ms]	-
ADSRdecay	>0[ms]	-
ADSRrelease	>0[ms]	-
grainread	0-1[float]	
_grainspeed	0-12[int]	
grainspeedspread	0-15[int]	
grainmultiply	0-12[int]	

Synthesizer Module Module details

- **slowMetroRate :** sets the speed of the slow control metro in milliseconds, this is used in the timing of random events such as the randomFrequencyMagnitude parameter
- **fastMetroRate :** sets the speed of the fast control metro in milliseconds, this is used in the timing of random events such as the randomFrequencyMagnitude parameter
- **masterFrequency:** sets the current frequency of the fundamental in hertz.
- **minMasterFrequency:** sets the maxi frequency for when masterFrequency is mapped to a continuously changing sensor value. - default value = 20Hz
- **maxMasterFrequency:** sets the maximum frequency for when masterFrequency is mapped to a continuously changing sensor value. - default value = 20 000 Hz
- **masterFrequencyScale:** determines whether a logarithmic or linear frequency scale is used when masterFrequency is mapped to a continuously changing sensor - default setting is Logarithmic. Bool 0 = linear Bool 1 = Log.
- **stepSize :** `quantizes' the masterFrequency parameter into discrete steps, smaller values result in smaller steps. This parameter is especially sensitive to small changes in value <1.
- **vibratoRate :** sets the rate of vibrato in Hz. Does not have upper limit, making AM-like modulations possible.
- **minVibratoRate:** sets the minimum frequency for when vibratorRate is mapped to a continuously changing sensor value. - default value = 0Hz
- **maxVibratoRate:** sets the maximum frequency for when vibratoRate is mapped to a continuously changing sensor value. - default value = 1 000 Hz
- **vibratoRateScale:** determines whether a logarithmic or linear frequency scale is used when vibratoRate is mapped to a continuously changing sensor - default setting is Linear. BOOL 0 = linear BOOL 1 = Log.
- **vibratoSize :** sets the vibrato size. limited to a float between 0 and 1.
- **randomFrequencyMagnitude :** sets the random frequency fluctuation size.Can be used to create more interesting/realistic vibrato with small values. Limited to a float between 0 and 1.
- **randomFrequencySpeedHighOrLow :** A boolean parameter which sets the randomFrequencyMagnitude's refresh rate to either that of the slowMetroRate or fastMetroRate's.

- **masterAmplitude :** sets the master amplitude at input *before* the amplitudeResponseCurve. Limited to 0-1f. *This parameter also adjusts the ADSR envelope during the sustain phase.*
- **amplitudeResponseCurve:** sets the response mapping curve, limited to 0-1f. a value of 0.5 results in a linear mapping with gain of one.
- **curvedOrLinearADSR:** accepts a boolean to switch between a quartic and linear ADSR envelope.
- **ADSRattack:** attack duration for envelope in ms. this sets the time taken to reach peak amplitude of the envelope, whose value is set with the noteOnOff method.
- **ADSRdecay:** decay duration of envelope
- **ADSRrelease:** release duration of envelope
- **noteOnOff :** plays note through the ADSR. Float [>0 - 1f] starts note at specified peak amplitude, sending a zero [0] triggers release.
- **oscillatorPreset:** preset waveform, 0 to 14.
- **noiseLevel:** set noise generator level between 0 and 1f. constrained by the main ADSR currently, though it may be given its own envelope in future updates.
- **reverbMix:** float [0-1f] specifies wet/dry mix of reverb
- **reverbPreOrPostMasterAmplitude:** boolean inserts reverb effect pre or post masterAmplitude.

Kinect Management

Output Message Structure

<UserID (int)> < Xcoord (float 0-1)> < Zcoord (float 0-1)>

< Xcoord (float 0-1)> and < Zcoord (float 0-1)> values will be used to calculate the overall distance

<UserID (int)> and <AliveOrNot (0 or 1)> will be used to limit the amount of Kinect user ID numbers. Kinect Management will take care about the new_user - lost_user- re_enter_user and exit_user features.