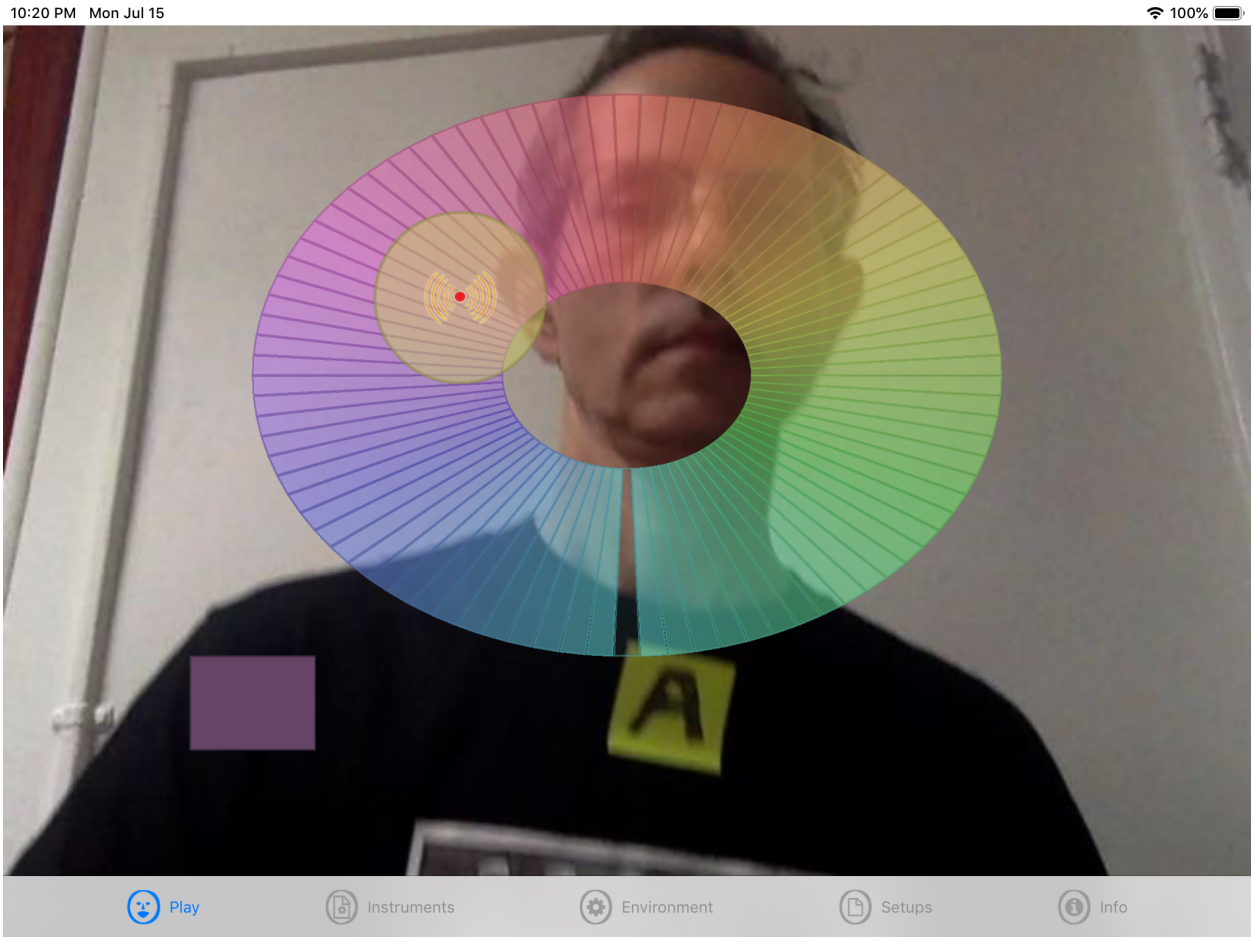
Henry Lowengard
AUMI Sings
December 22, 2019

AUMI Sings:
Technology and Philosophy

Technology for adaptive use improvised vocal music

A voice, a singing voice, a singing voice singing with other voices.



AUMI Sings[1] is a software musical instrument designed to use the video tracking technology used in AUMI to control a synthesized voice for the purpose of using that voice in structured improvised music pieces. The project was proposed in 2017 and funded in 2018 with a grant from the Bruneau Center for Excellence in Choral Music at Memorial University of Newfoundland.

AUMI has demonstrated over the years how including people with little voluntary movement in improvisatory music groups creates a situation for them to listen and be heard. **AUMI Sings** is a response to the question of whether this idea can be extended to vocal music, and more structured pieces.

Why structure?

The problem of turning a low information source of real time signals, such as video tracking, into the highly subtle articulations of human voice already implies that the signals would be directing the synthesis model toward collections of parameters (states), rather than having access to all the individual parameters controlling the vocal model. These states are context sensitive, implying a directed graph of states. Singing needs even more information to synchronize those states in time, intensity, articulation, and pitch spaces. Once in a particular state, a selection of those synthesis parameters should be able to be changed in real time, and also there should be a way to choose a different state.

It's a simple extension of state based processing needed to produce a voice to go past the level of sound to the level of composition. This is fortunate, because this allows more structured improvisations to be created and expressed. The state graph required to articulate and control vocal articulation can also move from note to note, measure to measure, section to section in a

[1] AUMI Sings, an as yet unreleased app by Henry Lowengard, in part based on AUMI, the Adaptive Use Musical Instrument app http://aumiapp.com

structured score.

Design Philosophy:

When this project was started, I pointed out that the problem of creating singing synthesis is really large. I suggested that there be a piece commissioned that would focus the project on particular capabilities to implement first, for research purposes. It turned out that I was chosen to be the composer for this, so I put together a simple piece named "Call and Response". Call and Response is designed for AUMI Singers to interact with a vocally abled choir. The piece combines long samples that are static text with more synthetic voices in a different section to test out more melodic real time playing. The idea is for the AUMI Singers to reach out to the choir, saying / singing "I am here" , and the choir hears and responds, and eventually the two interact musically.

Flexibility:

AUMI Sings was designed to be flexible with regard to the tracking technologies and audio generating models. The trackers currently supported are a motion tracker, a face tracker and a touch-only tracker, while other trackers based on recognizing the movement of blocks of color, fiducials, and rectangles are also being developed. The tracker technology is generalized to allow them to be changed and configured for a particular user or situation.

At the most general level, an **AUMI Sings** Instrument is structured as a series of states in a graph, with state transitions depending on the signals from its trackers' positions among sound and control "boxes" onscreen.. Each state has a user interface of a number of these boxes which interact with tracker driven cursors to generate events. The sound boxes are associated with a rank of sounds and a range of frequencies and intensities. Moving the cursor within a sound box will generate an event with information about what frequency and intensity that position

represents. These events are mapped to synthesizer commands, and queued for interpretation by the state control and synthesis software. The synthesis software applies these new commands to provide articulation parameters for the voice it generates. The queuing allows temporal uncertainty to be smoothed out and scheduled for specific beats. This indirection allows for more flexibility in determining how movements control the synthesis and the state transitions.

Trackers:

The *Motion tracker* does not try to recognize objects in a video image, it only looks for high contrast edges, spaced at least a specified distance apart, and only tracks movement more than a few pixels, to help filter out video noise. The average of the significant motion in the video is used to move the cursor relative to where it currently is.

The *Face tracker* uses Apple's face detection algorithms and locates the eyes and mouth. With this information, the cursor can be positioned on the nose, forehead, a a halo or necklace. The halo and necklace allow small angular movement to translate into larger screen movement. The face detector, though, fails on side views and obscured views of a face.

The *Touch tracker* doesn't use video, it merely tracks touches on the screen. It's good for testing the instruments, but can also be used as a legitimate interface. The Touch tracker is also available when the other trackers are in use.

Other trackers in development would allow for multiple cursors, and more information per cursor, such as angle of a detected shape, and closeness to the camera. This cursor information, and derived information such as its identity, cursor velocity and whether it has started or stopped moving, can then be combined with the characteristics of the sound and control boxes onscreen to make the event stream.

Event Stream:

Rather than directly controlling the synthesis and state transition engines from events, there is an object called the *Mapper* which takes the event stream and translates it into synthesis commands and control commands. This opens up the possibility for the event stream and the command stream to be made available to external programs that just want to use AUMI Sings' interface as a controller. Events are generated as cursors enter and leave boxes on the screen, and they are also generated when cursors aren't in any boxes.

A single event may be mapped to several synthesizer commands, creating such effects as echos and arpeggiations. The synthesizer commands are assigned times and put in a queue, which is the true source of commands for the synthesis engine. Control boxes currently only change states and instruments, but they will also be able to change timing parameters and other real-time aspects of the synthesis engine and queue.

The queue can schedule commands equally on beats, or play them immediately, or it can hold off on reading events until the current event is done playing. That last mode is good for stringing phonemes together.

Synthesizer:

AUMI Sings can have a number of models for its synthesis engine. Right now, the one that is implements uses audio samples that are interpreted in two ways: sustaining loop and granular synthesis. This allows for different levels of compositional use, for instance, whole phrases and loops, or phonemes. There are a number of "synthesis slots" available so there can be polyphony.

Samples:

The samples used in an instrument can be one long sample that excerpts are taken from, or a set of samples that are played entirely, or any combination of the two. The portion of a sample that is actually used is called a snippet, which is assigned a frequency and intensity as well as

attack, sustain, and release sections. The snippets are organized in ranks, which is to say, to play a note, a snippet is chosen from the rank with a matching frequency and intensity. Pitches that don't match the frequency find the closest sample and tune it to the intended pitch. This is also true of the intensity.

This concept of "real time pitch and intensity" can be used for more synthetic models, such as spectral or LPC or wave guide based synthesis. That would result in some very smooth transitions. Intensity does not have to mean intensity, it may just mean a different sample, making a kind of round robin sampling possible. For instance different vowel sounds can be related to intensity, so a single instrument could speak several vowels.

Sustaining means that when a "Note On" synthesis command occurs, a snippet is chosen from a rank and adapted for the intended pitch, and a slot for it is allocated. Then the Attack and Sustain portions play through. If there is no "Note Off" synthesis command, the sustain portion will loop. When the "Note Off" comes, the sustain will finish playing and the release portion will play out and the slot will be made available again. The lengths of any of these sections can be zero, thus being able to make one-shot sounds like AUMI or sustaining drones.

Granular synthesis mean the sustain portion is broken up into tiny pieces and those pieces are composited together to independently manipulate length and pitch. Instead of the whole sustain portion repeating, it's as if there were hundreds of tiny sustain portions. There are parameters for the granular synthesis the should be able to be manipulated like vibrato once more event information is available.

While the rank is being played, pitch and intensity information can influence its pitch and intensity. A sound that starts using one sample and its section information can be changed while it's playing. Ideally, this would be a 4-way bilinear interpolation, but it just jumps to the closest

snippet as it is now.

AUMI Sings User Interface:

The runtime configuration of AUMI Sings is in two parts: the *Instrument part* and the *Environmental part*. The Instrument part handles the states, the graph, and the assignment of sounds, and default skin information. The Environmental part handles the configuration of the video camera, the trackers, custom skins, and debugging options. The UI lets you load and save Instruments and variants on instruments if you have customized them a little. The same goes for the Environments. A combination of Environment and Instrument, called a Setup, can also be saved and otherwise maintained. The UI also provides an online manual.

Currently, the creation of instruments has to be done in software because its too hard to do directly using the UI. I'm thinking of putting in a simple programming language to help with that. Some of the tools for manipulating the Instruments are available in the UI, though. Sound boxes can be generated and laid out in standard patterns, and then rearranged, and some queue and runtime characteristics can be changed. For instance, a sound box, which may not be shaped like a box, may be set to have pitch manipulation on its 'u' axis and intensity on its 'v' axis or swap them. It can be assigned granular or sustain loop synthesis methods. It can also be moved around manually on screen.

I mentioned Skins: these are the colors or images for user interface elements, mostly for the control boxes, cursors, and background. The custom colors can help distinguish different characteristics of the boxes. It can also be adjusted to accommodate visual impairments.

AUMI Sings is a Universal app, but needs help structuring itself on an iPhone because there is so little space! The UI will need to be customized for iPhone use. Nevertheless the

instruments and transitions run on iPhones.

Current State:

AUMI Sings can run well enough to demonstrate its vocal and state changing possibilities. As mentioned, it cannot configure instruments yet because that will need a lot of design work and possibly a language to help build the instruments. It will need interfaces to import sounds and visual media (for use by Skins), and it will need lots of testing and composition use cases. It will need its documentation to be kept up to date, and accompanying documents like this one should be places on its website.

More importantly, I'm thinking of changing how the onscreen boxes work. Instead of having sound and control boxes, the two concepts would combine so that boxes could both trigger sounds and control commands, and furthermore, some logic could be coded for each box in a sccript to provide more than one command and interpret signals with more subtlety. The script language could be also be associated with the instrument itself, and used to lay out and color the boxes and set their scripts.

When it has those things, it will be ready to release to the general public.

There is a video playlist with examples and more explanations at:

https://www.youtube.com/playlist?list=PLFp-XbtuB0odMUN1N1zYvFrVjLEXxYCJW

or https://bit.ly/32vWFxY