NYU Creative Coding - Final Project - Sept 2020

Philip Hughson philipfhughson@gmail.com

# IMITATION LAB

See the project at https://www.openprocessing.org/sketch/969279

# Creative concept

This project as based around the idea of imitation as a means of cultural transmission between humans. It presents a number of short video clips as sources, and allows its user to record themselves reproducing the sources (or otherwise!), and allows the results to be compared and combined.

As well as the fun to be had playing with video in this way, the idea has interest in areas such as music and dance education and language learning. The underlying idea is that to learn through imitation, we need to perceive the difference between our own behaviour and that of the person we're imitating. This can often be hard to do ("know thyself!"), and though teachers may help us, sometimes recording ourselves is the best way to face the truth....

# Inspirations

This ideas behind this grew out of a project I did at the Glasgow School of Art on design for imitative leraning for rock climbing (see http://phhu.org/imitationgames), and also a project I did on visualisation of dance (https://phhu.org/patagraphy). I experimented with projecting videos on to climbing walls, but noticed that I delay was required to review the difference between one's own performance and somebody elses.

The ideas behind this come from reading Richard Dawkins and later Daniel Dennett on memes and cultural evolution (see "From Bach to Bacteria and Back" especially).

More recently, when teaching dance (tango), and in particular when taking online classes via Zoom during Covid-19 restrictions, I started thinking more about video as a didactic and communicative medium. Could, for example, students study and compare themselves with examples of dance on YouTube?

Pipilotti Rist's installation "Ever is Over All" (see https://www.youtube.com/watch?v=a56RPZ_cbdc&ab_channel=vobk) also provided the notion of blending two videos together to counterpose two ideas. P5.js can blend images, so I tried this in my visualisations.

Jason Sigal's consideration of visualisation of sound (http://slides.com/jasonsigal/deck#/1 and https://therewasaguy.github.io/p5-music-viz) give me some ideas on visualisation of sound. Much visualisation in p5.js can be somewhat arbitrary: if it doesn't help you understand the form better than your ears can, it's just distraction. But Sigal's Fast Fourier Transform spectrograph, as well as being colourful, seems useful, helping rhythms and harmonic to be seen.

The brief for this project requires an "interactive collage". The interactivey here is hopefully obvious: it allows the user to combine his own "behavioural output" with that of others, thereby faciliating mimetic evolution. The web cam (and browser) are the key tools for this! (senses other than hearing and vision will have to wait...).

The notion of collage is about combining elements, literally pasting things together. Here the notion of collage implemented is more abstract: by combining the capacities (memes) of others with our own, we create a behavioural collage, perhaps an altered or extended identity. This is collage in terms of memetic evolution.... More literally, the blended video visualisation, and "picture line" visualisation, present a pasting together of elements from different sources.

A major aim in this work is to explore how to make the act of imitation as simple as possible to do and perceive, which still having the act of comparison reside in the brain of the user, which I believe it needs to be if learning is to happen. The "seq-rec" and "seq-play" buttons, allowing sequential and iterative playing and recording of memes without any intervening clicks, are a first attempt in this direction.

# Code overview

The `Meme` class is the core of this sketch. It is essentially a wrapper around the existing p5.js mediaElement class, providing standardisation of setup, methods for updating the source, various convenience properties, and a P5.js Fast Fourier Transform object for producing the required. The main method in `Meme` is addVideo, which sets up the video display. Later `updateVideos` is used to allow changing of the source for the video display (e.g. when the source selector is changed).

The `Meme` class is extended as a `CaptureMeme`, which swaps the createVideo method for a createCapture one. The `creator` method is overriden to do this. (There is some lazy coding in the `Meme` class, where the isCapture property is used to determine some behaviours: this might better be

incorporated into `CaptureMeme`). This extension largely allows the Memes to be treated equivalently in the code.

The three video displays on the top half of the sketch correspond to the three Memes instantiated in the main tab's `setup()` method. It can be seen that two of them, referenced as `a` and `b`, are passed to the visualisation chosen by the user in the `draw` method. These `memes` are used as data sources for the visualisation. Note here that there's flexibility: it's arbitrary which one is which (meaning that, for example, a meme can be combined with itself).

There are three tabs concerned with user interface: "buttons" covers the DOM buttons, mostly transport controls somewhat like on a tape recorder; and the UserInterface tab and class re-uses a class I created in a previous sketch ("Particle Interface", https://www.openprocessing.org/sketch/948413). The uiDef tab contains the instantiation of this class, in other words the settings for the controls.

Both the buttons and the UserInterface tabs have similar functions, and they could perhaps be combined. That they are separate is perhpas an artifact of development: I made the buttons first, and then realised I wanted the same declarative logic I'd used before, especially to get mouseover explanations of controls, so I imported the UserInterface for this purpose. In both cases, you'll note that I prefer a declarative array-of-objects structure for the interface definitions, and then use either functions (`buttonSpecs.forEach` ...) or a class (`UserInterface`) to process them, allowing for standardisation.

The buttons tab contains much of the logic which handles transport controls. Some of this is a simple mapping to methods of mediaElement, such as `play` and `stop`, but some is more complex. As mentioned above, I wanted to be able to automate the process of listening and recording. A slightly complex chain of event listeners (`onended`) and promises (from the record method) is co-ordinated to do this using a state variable `seqRunning`. The works, but the logic should be regulalarised to allow for greater flexibility of sequences.

Also on the buttons bad is logic to allow offsetting of play start times, and speed changes: both as controlled by the UI.

The "record" tab contains functions adapted from MDN (https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API/Recording_a_media_element) for recording from the capture meme to the recording meme. These divide into the purely functional `record`, which takes a stream and returns a data URL, and `doRecording`, which co-ordinates updating the recording meme. Recording like this provided surprisingly easy to set up!

Lastly there are four tabs for visualisations. Excpeting `info`, which handles all three, each of these provides a function which takes two memes as arguments, called a and b, and produces a visual using

familar p5.js drawing techniques (otherwise somewhat absent in this sketch). Since data flow is unidirectional here, functions are simpler to use than classes. `spectrograph` uses the `FFT` from each meme to make a spectrogrpah from coloured rectangles, and `pictureLine` and `blended` both blend images using a choice of filters. In all cases, the visualisations are drawn in real time as the videos play, which is much easier and faster to execute than trying to save the timeseries data from each video. To add a new visualisation, it would only be necessary to add a new tab with a similar defintion.

The sources for this sketch are all videos taken from YouTube, trimmed to a short length, and with altered aspect ratios in some cases. These are given as samples, but clearly more flexiblity in choice (e.g. arbitrary video URLs) would be desirable.

# Further work

This project is a good proof of concept, but it is limited in many ways and could be developed further.

- As mentioned above, arbirary URLs could be used as sources, presumably with a time segment specified (e.g. #t=5,10).
    - CORS restrictions may be an issue here, as might copyright
- It would nice to be able to upload recording memes (the video player provices a download button, which is a start).
- User testing would help to clarify workflow, perhaps working in multiple domains.
- A more elegent and flexible layout would help.
- more flexibility in sequencing (e.g. arbitrary specifications of actions like play-source, rec, play-recording, repeat....)
- Visualisations consisting of multiple iterations of a meme could illustartion / facilitate evolution and comparison.