Politecnico di Milano - Computer Music: Languages and Systems (2023), 1–8

HOMEWORK #3 REPORT

teleKINesis: Real-Time Voice Processing

Nicolò Chillè, Enrico Dalla Mora, Rocco Scarano, and Federico Caroppo

Group Name: ***La Lobby*** **– Group ID: 3**

Abstract

The purpose of this document is to describe the process of implementing a *computer music system* based on interaction design principles. We will focus on how we established the connection between the different system units, which in the examined case are the following: **Microsoft Kinect V2** (*interaction system unit*), **SuperCollider** (*computer music unit*) and **Processing** (*graphical feedback unit*).

Keywords: computer, music, interaction, Kinect, Processing, SuperCollider

The designed system is meant to be a tool allowing the performer to **process his voice in real time**, controlling the parameters of the applied effects with his limbs (specifically, his hands and arms). To do so, we interfaced a Microsoft Kinect sensor with a SuperCollider effect bank, providing graphical feedback via a Processing-based GUI. The devices were networked via the Open Sound Control (OSC) communication protocol. The structure of such network is shown in the following diagram:

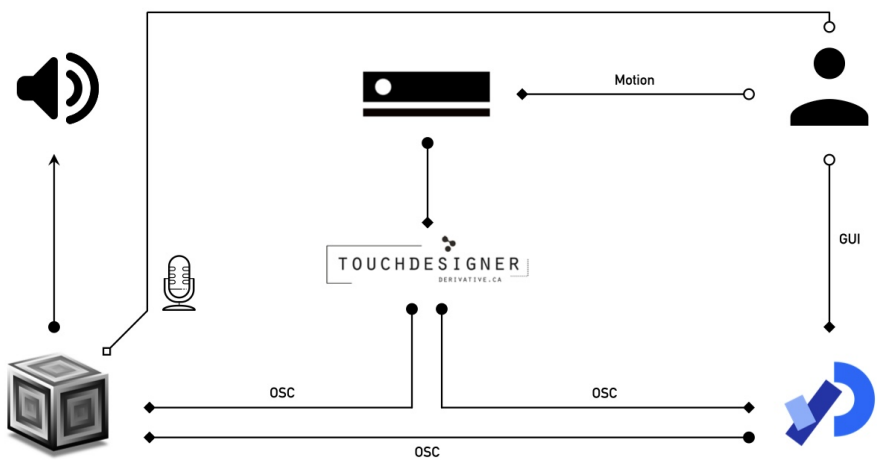
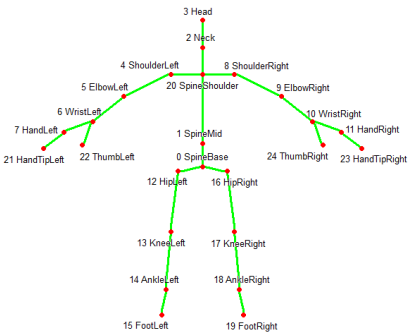


Figure 1. Network architecture.

1. Interaction System Unit

1.1 Microsoft Kinect

The Microsoft Kinect controller is a motion sensitive input device, capable of tracking the skeleton – i.e. the position of limbs and joints – (Figure 1b) of one or multiple subjects. It was chosen because of its accuracy, but its production and support were discontinued, for the Microsoft Windows environment, almost a decade ago: this resulted in a number of problems from the software perspective, due to most libraries designed around the device being deprecated.



(a) The 25 joints of a Microsoft Kinect skeleton.

(b) The Kinect V2 sensor.

Figure 2

1.2 TouchDesigner

The Kinect sensor is not OSC-native, i.e. is not capable of sending OSC messages as output. Thus, it was necessary to use an external software, namely ***TouchDesigner***, to collect the sensor's data and translate them into OSC messages. TouchDesigner is a **node based visual programming language** specifically designed for real-time interactive multimedia content, whose structure slightly resembles the one of a modular synth: by connecting together input, output and processing nodes, it is possible to create a complex multimedia stream. Our configuration consists of the following network:

1. The first node *captures the analytical data* (i.e. the absolute positions of the Kinect skeleton joints) from the Kinect sensor;
2. The second node *selects*, from the whole collected database, *the parameters relative to the hands* of the performer;
3. These values are then normalized with respect to the position of the body and the length of the arm in order to get values ranging from 0 to 1. This is achieved by using various nodes which basically compute the following passages:

$$x = \frac{\text{hand position} - \text{shoulder position}}{\text{humerus length} + \text{forearm length} + \text{hand length}} \tag{1}$$

The procedure is the same for the y coordinate and for both hands.

4. The third and last node *converts the data in OSC messages* and sends them in output to a user-selectable port on the Localhost server.

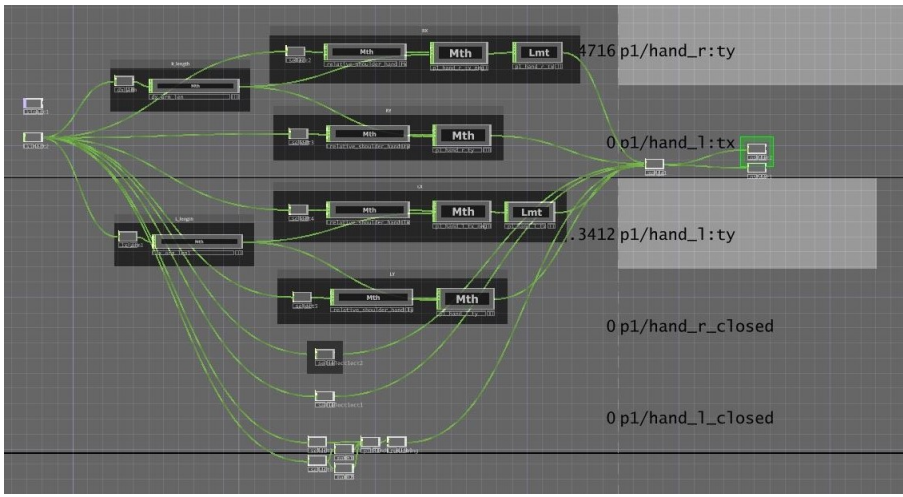


Figure 3. TouchDesigner network.

2. Computer Music Unit

The computer music unit consists of a SuperCollider script structured as follows:

- **effects bank**: a number of `SynthDef`s employed to define the range of effects among which the user will be able to choose.
- **effect selection**: SC listens to the GUI-dedicated server port for a OSC-message signaling that the user has selected/changed one of the two possible effects.
- **signal mix**: the performer can select up to two different effects to process his/her voice, thus a `SynthDef` is designated to mix the two processed signals with the clean one and send them in output. The effects are connected in parallel in order to allow the performer to control them independently;
- **looper**: the user can loopback his voice by closing and opening his fists. This process is handled by the `loop Synthdef`;
- **amplitude and modulation setters**: SC listens to the LocalHost port assigned to the TouchDesigner messages concerning the position of the performer's hands, and when a message is received proceeds to modulate the selected effect(s) parameters (including the signal volume) accordingly;
- **loop control**: dedicated `OSCFunc`s will handle the OSC messages received on the same port. The data sent by TouchDesigner will then be processed and will allow the performer to record, overdub, play and stop the loop.

2.1 Effects Bank

Eight different effects were implemented in SuperCollider in order to process the performer's voice, namely a delay, a reverb, a phaser, a flanger, a distortion, a harmonizer, a tremolo and a chorus. These `SynthDef`s output only the wet signal on dedicated busses, which will later be mixed with the clean one;

2.2 Effect Selection

As mentioned before, SuperCollider listens to two different ports in the LocalHost server (use Figure 1 for reference). One of these is dedicated to the effect selection from the graphical unit: when the user selects an effect from the GUI dropdown menus, Processing sends a message to the specific port containing a reference to the corresponding hand ('`/sx`' or '`/dx`') and a number, from 0 to 8, corresponding to one of the possible effects. When this message is seen by SuperCollider, the following `OSCdef`s (for conciseness, only the left one is reported) are entrusted with the setting of the global variables `~fxIndexSx` and `~fxIndexDx`, which are set to 8 (i.e. no effect) by default:

Code 1. `OSCdef` (left hand).

```
OSCdef.new(\fxIndexSxOSC, {
    arg msg;
    //msg[1].postln;
    ~fxIndexSx = msg[1];
    ~mix.set(\fxIndexSx, msg[1]);
  }, '/sx', nil, 57120);
```

Once the left and right effects indexes are set (note that this is necessary for the modulating `OSCFuncs`), the corresponding parameter of the `mix` instance is also set, allowing for a real-time switch in the performer's voice processing effects.

2.3 Signal Mix

The following `SynthDef` is entrusted with mixing the microphone input with the two processed signals which are completely wet. This is possible because each of the eight effects is instantiated in a global variable. The effected signals are retrieved by the `In.ar SynthDef` which will select, according to the `~fxIndexSx` and `~fxIndexDx` values, the busses corresponding to the selected effects. The volume of the effected signal is not controlled via dry-wet, since it would have prevented the user from controlling only one of the two effects. For this reason, it is controlled by a depth value and the clean signal will always be present.

Code 2. Signal mix `SynthDef`.

```
SynthDef(\mix,
  {
    arg fxIndexDx, fxIndexSx, ampDx=1, ampSx=1;
    var input, ampIn=1, sigDx, sigSx, mix;
    input=SoundIn.ar(0!2);
    sigDx=In.ar((10+(fxIndexDx*2)),2);
    sigSx=In.ar((10+(fxIndexSx*2)),2);
    mix=(input*ampIn)+(sigDx*ampDx)+(sigSx*ampSx);
    Out.ar(0, mix);
  }
).add;

~mix = Synth.new(\mix, [\fxIndexDx, 10, \fxIndexSx, 10]);
```

2.4 Loop

The loop processing block takes as input the output of the Mix `SynthDef`, so it's in series: this allows the performer to loop the voice after it has been effected. The function is based on the use of the `BufWr` and `BufRd` `SynthDefs` which will respectively write and read from a previously allocated buffer.

Code 3. Looper `SynthDef`.

```
~b = Buffer.alloc(s, s.sampleRate * 300);

SynthDef(\looper, {
  arg xfade=0.02, trigg=0, buf=0, run=0, reclev=0, prelev=0,
      recAmp=1, loopAmp=1, micAmp=1, mixAmp=1;
  var mic, trig, max, ptr, loop, rec, mix;
  mic = In.ar(50);
  trig = Trig1.ar(trigg, ControlDur.ir);
  max = Sweep.ar(trig, SampleRate.ir * run);
  ptr = Phasor.ar(trig, 1, 0, max, 0);
  loop = BufRd.ar(1, buf, ptr);
  rec = sum([
    mic * reclev.varlag(xfade,-2),
    loop * prelev.varlag(xfade,-2)
  ]);
  rec = rec * recAmp.varlag(xfade,-2);
  BufWr.ar(rec, buf, ptr);
  mix = sum([
    loop * loopAmp.varlag(xfade,-2),
    mic * micAmp.varlag(xfade,-2)
  ]);
  mix = mix * mixAmp.varlag(xfade,-2);
  Out.ar(0, mix!2);
}).add;
```

2.5 Signal Modulation

Each of the signals amplitude is modulated by the y coordinate of the corresponding hand, while a specific parameter for each different effect is subject to modulation depending on the x coordinate of the performer's corresponding hand. The values are normalized by means of the arms length directly in TouchDesigner, in order to take into account the distance between the subject and the Kinect sensor, and remapped to a suitable range in SC. The following `OSCdef` and `OSCFunc` are entrusted with this process (as before, only the ones relative to the left hand are reported):

Code 4. Modulation `OSCdef` and `OSCFunc`.

```
s = OSCFunc({ arg msg;
  switch (~fxIndexSx,
      0, { ~delay.set(\delayTime, LinLin.ar(msg[1], 0, 1, 0.1,
          1.2));},
      1, { ~reverb.set(\room, LinLin.ar(msg[1], 0, 1, 0.1, 3))
          ;},
      2, { ~phaser.set(\rate, LinLin.ar(msg[1], 0, 1, 0.1, 3))
          ;},
      3, { ~flanger.set(\freq, LinLin.ar(msg[1], 0, 1, 0.1, 3))
          ;},
      4, { ~distortion.set(\dist, LinLin.ar(msg[1], 0, 1, 0.1,
          0.01));},
        5, {
      if (msg[1]>0.2, {~harmonizer1.set(\amp, 1);}, {
          ~harmonizer1.set(\amp, 0);});
      if (msg[1]>0.4, {~harmonizer2.set(\amp, 1);}, {
          ~harmonizer2.set(\amp, 0);});
      if (msg[1]>0.6, {~harmonizer3.set(\amp, 1);}, {
          ~harmonizer3.set(\amp, 0);});
      },
      6, { ~tremolo.set(\freq, LinLin.ar(msg[1], 0, 1, 4, 7))
          ;},
      7, { ~chorus.set(\delay, LinLin.ar(msg[1], 0, 1, 0, 1))
          ;},
      8, {}
    );
  }, '/p1/hand_l:tx', nil, 7002);

 OSCdef.new(\l_h_y,
  {
    arg msg;
    ~mix.set(\ampSx, msg[1].linexp(-0.5,1,0.001,1.5));
  },
  '/p1/hand_l:ty',
  nil,
  7002
);
```

2.6 Loop Control

Since TouchDesigner outputs a stream of values, both for the `hand_r_closed` and the `hand_l_closed`, the global variables `~prev_r_closed` and `~prev_l_closed` are used in order to trigger the actual function only when the value changes. By closing the right hand, the user starts recording a loop and this results in clearing previous buffers (in case another loop was previously recorded) and re-instantiating the `loop` synth (note that the parameter `addAction: 'addToTail'` is needed to keep the order of execution consistent). When the player stops writing the buffer, by opening his right hand, the loop will automatically start playing and, by closing the left hand, he will start overdubbing the loop (keeping its duration unchanged). Finally, closing the hands together will fade out the loop, without clearing the buffer.

Code 5. Loop control `OSCdefs`.

```
s = OSCFunc({ arg msg;
  if(msg[1]==~prev_r_closed,{},{
    if( msg[1]==1,{
      ~b.zero;          //clear buffer
      ~looper.free;     //free synth
      ~looper = Synth(\looper, [\buf, ~b.bufnum], addAction: '
          addToTail');    //re-instantiate synth
      ~looper.set(\trig, 1, \run, 1, \reclev, 0.5, \prelev, 1,
          \xfade, 0.02);    //start rec
    },{
      //stop rec and play loop
            ~looper.set(\run, 0, \reclev, 0, \prelev, 1);
    });
  })
  }, '/p1/hand_r_closed', nil, 7002);

s = OSCFunc({ arg msg;
  if(msg[1]==~prev_l_closed,{},{
    if( msg[1]==1,{
      //start overdub
            ~looper.set(\run, 0, \reclev, 0.5, \prelev, -1.
                dbamp);
    },{
      //stop overdub and play loop
            ~looper.set(\run, 0, \reclev, 0, \prelev, 1);
    });
  })
  }, '/p1/hand_l_closed', nil, 7002);

s = OSCFunc({ arg msg;
  if( msg[1]==1,{
        //Fade out loop
        ~looper.set(\run, 0, \reclev, 0, \prelev, -20.dbamp);
    },{})
  }, '/p1/hands_touching', nil, 7002);
```

3. Graphical Feedback Unit

The Graphical Feedback Unit consists of a Processing script which uses a Kinect-dedicated library (namely `KinectPV2`) to display the data stream captured by the Kinect as a point cloud. This way, the performer can "see what the sensor sees" and perceive how his/her movements affect the GUI and the sound processing[1] (Figure 4). The dropdown menus allow the user to select the chosen effects by means of the default "dropdown" function: this function bears the same name as the dropdown menu instance it refers to, and is dedicated to executing a set of commands each time the user selects an element from the menu. We configured it to send an OSC message (`oscP5` library) containing the corresponding effect ID to the specific port SC is listening to. This message triggers the process described in the previous section. At the same time, the script watches out for incoming messages from TouchDesigner, and the GUI sliders are updated in real time with the normalized values of the hands coordinates sent by TouchDesigner. Finally rec and play icons are also displayed when the corresponding OSC message is received.

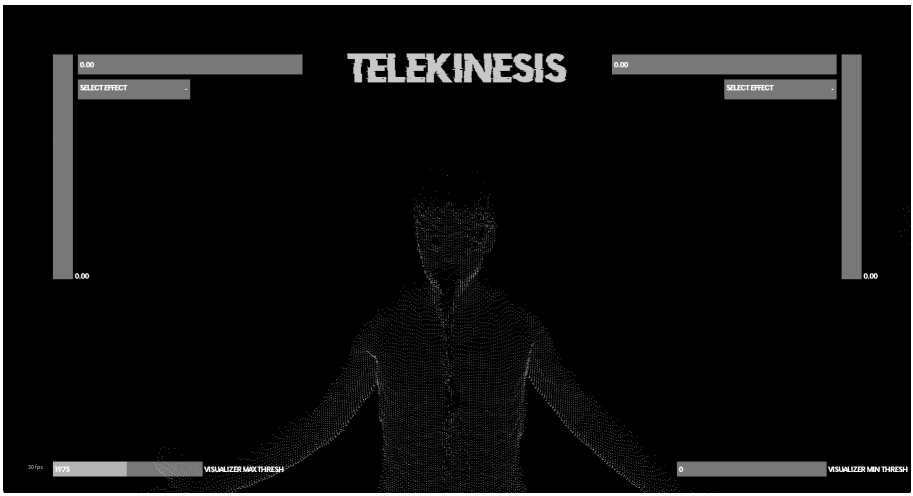


Figure 4. Point cloud GUI.

1. It is possible to manually set the Kinect point cloud threshold, i.e. the depth of view, via the dedicated sliders.