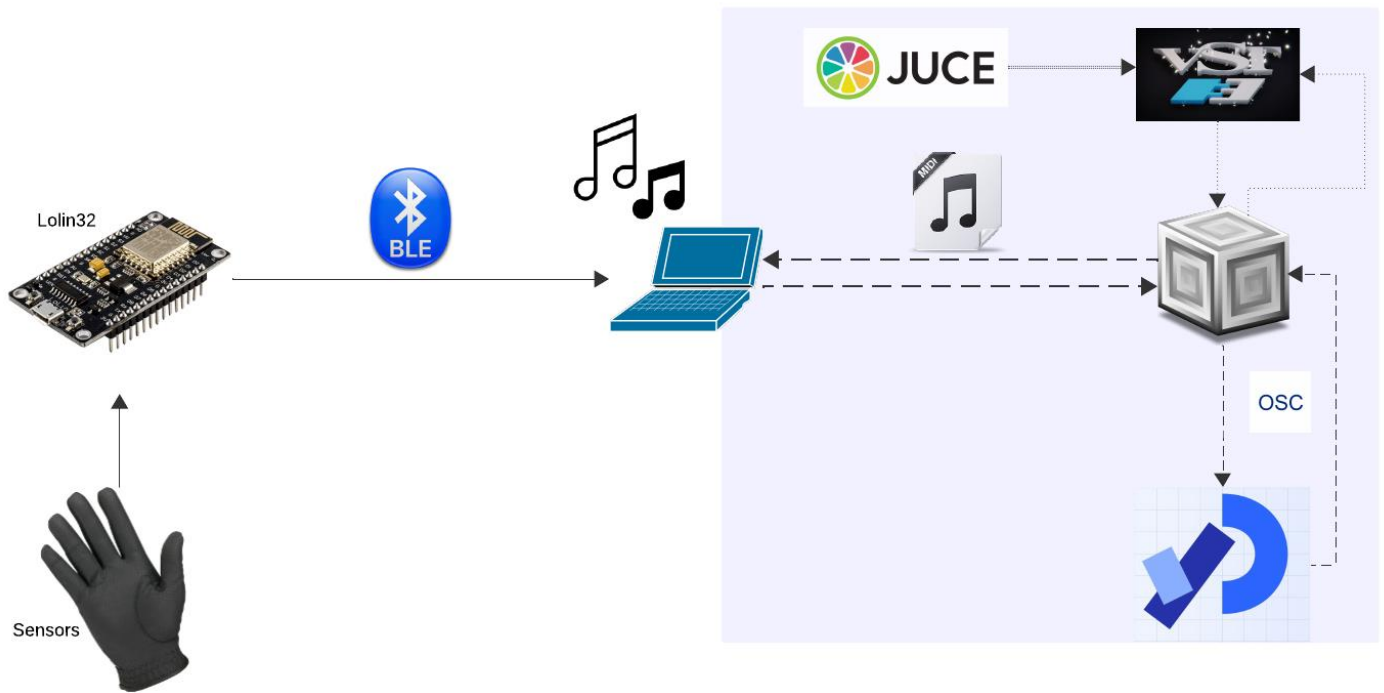


REPORT CMLS

ABSTRACT:

The objective of this project was implementing a computer music system with interaction design principles. The constraints were to use at least these three following blocks:

- Interaction system unit: in our particular case ARDUINO;
- Computer music unit: using SuperCollider and JUCE;
- Graphical feedback unit: using Processing.



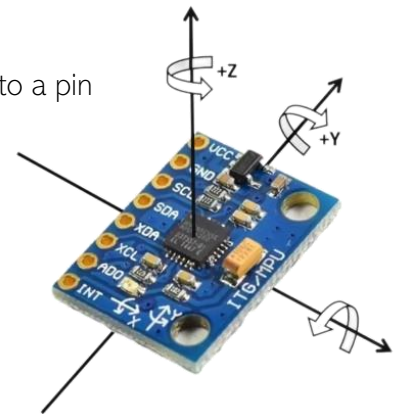
Our project consists of a glove with three sensors which detect the hand's movements and, by using a Bluetooth communication, the data is collected by our interaction system unit. The system unit elaborates the data received, to create a MIDI message suitable for SuperCollider. SuperCollider is used to generate different kind of waves that can be chosen by the GUI. The waves are then elaborated by the VST, created using JUCE, to obtain a sound similar to human voice. The graphic interface, other than choosing the type of wave, gives the possibility to change some parameters of the sound synthesis. This interface communicates through OSC messages with SuperCollider.

ARDUINO:



The Arduino board we used is the "Iolín32", which is connected through the I2C communication to the sensor MPU6050. This sensor consists of an accelerometer and a gyroscope that can estimate the hand's angular rotation and through the Arduino's Bluetooth interconnection it sends the MIDI control message to SuperCollider. Our code utilizes the MPU6050's accelerometer to measure the device's acceleration along three spatial axes and its gyroscope to measure its angular velocity around these same axes. These data are used to accurately calculate the device's orientation in space, in terms of roll and pitch angles. This orientation information is then translated into MIDI messages. Essentially, the device can control music based on its physical movements.

In addition, we have a resistor sensor that estimates the hand's flexion, it is linked to a pin located on the index finger and, by touching it, it defines the MIDI note we want to send. The glove contains also three buttons on the back of the hand that are used for choosing the octave we want the human voice to reproduce. This type of message is still a MIDI control message.



In order to have a Bluetooth communication, the lines of code here represented have to be included in Arduino IDE. This code snippet configures a Bluetooth MIDI server named "MIMU gloves". It then waits for a device to connect to the Bluetooth MIDI server. During the waiting period, it prints "Waiting for connections" on the serial port, and a dot is printed every second until a connection is established. When the connection is established, "Bluetooth connected." is printed on the serial port. The main loop of the program continues to check if the Bluetooth MIDI connection is active, repeating the waiting process if the connection is lost.

Note that in our case, we are using a BLE Bluetooth, this means that, compared to the classic Bluetooth, this device is using less energy.

```
BLEMidiServer.begin("MI.MU ");
Serial.print("Waiting for connections");
while(!BLEMidiServer.isConnected()) {
  Serial.print(".");
  delay(1000);
}
Serial.println("\n Bluetooth connected.");

void loop(){
  if (!BLEMidiServer.isConnected()) {
    Serial.print("Waiting for connections");
    while(!BLEMidiServer.isConnected()) {
      Serial.print(".");
      delay(1000);
    }
  }
}
```

```

void getMidiNote() {
  curr_n = !curr_n;
  int out;
  float_t val = analogRead(FLEX);
  if (val < 1800) val = 1800;
  else if (val > 3400) val = 3400;
  out = int((val - 1800)/(1600/12));
  // Serial.printf("val = %f, out = %i\n", val, out);
  midi_note[curr_n] = out + octave;
}

/* ---- TOUCH FUNCTIONS ---- */

void touchButton(bool *bt, uint8_t pin, uint8_t tresh, void (*when_pressed)(uint8_t),
  void (*still_on)(uint8_t), void (*when_released)(uint8_t)) {
  if(touchRead(pin) < tresh && !*bt) {
    (*when_pressed)(pin);
    *bt = true;
  } else if (touchRead(pin) < tresh && *bt) {
    (*still_on)(pin);
  }
  else if (touchRead(pin) > tresh && *bt) {
    (*when_released)(pin);
    *bt = false;
  }
}

```

In this part instead we are sending MIDI notes, checking if there are new notes and stopping the notes if the touch button is released.

```

getMidiCc();
//Serial.printf("roll: %i --- pitch: %i\n", midi_cc[curr*2 + 0], mid:
touchButton(&strip, T_BUTS, thresh1, evaluate, evaluate, setOctave);
//touchButton(&strip, T_BUTS, thresh1, evaluate, empty, setOctave);
touchButton(&touch, TOUCH, t_tresh, sendNote, checkNote, stopNote);
if(midi_cc[curr*2] != midi_cc[!curr*2])
  BLEMidiServer.controlChange(1, 12, midi_cc[curr*2]);
if(midi_cc[curr*2 + 1] != midi_cc[!curr*2 + 1])
  BLEMidiServer.controlChange(1, 13, midi_cc[curr*2 + 1]);
curr = !curr;
delay(10);
}

```

The first part, the `getMidiNote()` function, handles reading the resistance sensor. This sensor provides an analog value reflecting the flexion of the hand. The code performs some processing on this value to ensure it falls within a certain range. Then, based on this normalized value, a MIDI note representing the position of the flexion is calculated.

The second part of the code concerns the `touchButton()` function. When the touch pin is pressed it get the current note given by the resistor.

```

void sendNote(uint8_t arg) {
  Serial.printf("in sendNote\n");
  getMidiNote();
  BLEMidiServer.noteOn(1, octave + midi_note[curr_n], 127);
}

void checkNote(uint8_t arg) {
  Serial.printf("in CheckNote\n");
  getMidiNote();
  if(midi_note[curr_n] != midi_note[!curr_n]){
    BLEMidiServer.noteOff(1, midi_note[!curr_n], 127);
    BLEMidiServer.noteOn(1, midi_note[curr_n], 127);
  }
}

void stopNote(uint8_t arg) {
  Serial.printf("in StopNote\n");
  BLEMidiServer.noteOff(1, midi_note[curr_n], 127);
}



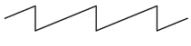

```

In this part, we are sending MIDI cc. These MIDI cc come from the elaboration of the data of the gyroscope and the accelerometer. These MIDI messages are then sent to SuperCollider which tells to the plug-in what is the frequency to cut in the filter.

SUPERCOLLIDER:



Our SuperCollider code defines four different types of synthesizers, each corresponding to a different waveform:

- Sinusoid 
- Square wave 
- Saw tooth 
- Impulse train 

Each synthesizer has parameters such as frequency, pan position, gate, amplitude and detune. Each voice is played with different horizontal panning to create a spatialization effect in the stereo image, this parameter can be modified directly from the GUI, and the same applies for the detune and amplitude. The frequency is given by the MIDI note that comes from the flexion of the hand and at the same time, the gate says to start and stop that note.

In essence, the code creates and manages multiple voices of synthesizers with four different waveforms to produce a polyphonic and spatialized musical composition. It is basically the core of our project as it manages the MIDI messages, the OSC messages coming from the GUI and interacts with the VST created with Juce.

```
//CONNESSIONE ALLE SORGENTI MIDI
MIDIClient.init;
MIDIIn.connectAll;

//RICEZIONE DELLE FREQUENZE DEI FILTRI DA ARDUINO
(MIDIdef.cc(\filter1, {
  arg val;
  ~fx.set(0, val); //0 per la frequenza del primo filtro
}, 12);
MIDIdef.cc(\filter2, {
  arg val;
  ~fx.set(1, val); //1 per la frequenza del primo filtro
}, 13);
//SUONA IL BUS
~myBus.play;
})
```

Through a MIDI communication, SuperCollider is capable of receiving MIDI messages (that include MIDI notes and MIDI cc) from Arduino. The MIDI messages are sent from Arduino to the PC through a Bluetooth communication. Here we have the part of code on SuperCollider that is responsible for the MIDI communication between Arduino and SuperCollider.

```
//RICEZIONE DELLE NOTE MIDI DA ARDUINO
MIDIdef.noteOn(\nota, {
  arg val, note;
  ~noteSet=note;
  ~b1.set(\freq, ~noteSet.midicps);
  ~b2.set(\freq, ~noteSet.midicps);
  ~b3.set(\freq, ~noteSet.midicps);
  ~b4.set(\freq, ~noteSet.midicps);
  ~b5.set(\freq, ~noteSet.midicps);
  ~b1.set(\gate1, 1);
  ~b2.set(\gate1, 1);
  ~b3.set(\gate1, 1);
  ~b4.set(\gate1, 1);
  ~b5.set(\gate1, 1);
  ~n.sendMsg('/MidiNote', ~noteSet);
  postln(note);
});
//ATTIVAZIONE DELLA NOTA CON IL GATE
MIDIdef.noteOff(\noteOn, {
  ~b1.set(\gate1, 0);
  ~b2.set(\gate1, 0);
  ~b3.set(\gate1, 0);
  ~b4.set(\gate1, 0);
  ~b5.set(\gate1, 0);
});
```

As we can see, SuperCollider receives MIDI note messages and in the second part MIDI cc messages in order to control the frequencies of the filter. The last part is related to the activation of the note, indeed if the message is a NoteOn message the function will switch on the gate and send an OSC message to Processing so that it can be visualized as a feedback for the user.

With these sections of code, SuperCollider will search an available VST plugins in the default VST search paths. To do this we use the SuperCollider extension "VSTPlugin" to store the VST in a Ugen. In the second line instead, we create an audio bus to store the source audio signals. The plug-in will take this wave form, add its effect and then replace this final wave on the bus. With VSTPluginController it's possible to manage the parameters of the filters changing their resonance frequencies.

```
// search of available VST plugins
VSTPlugin.search;
~myBus = Bus.audio(s, 2);

(
  SynthDef(\effetto, { arg bus;
    //ReplaceOut.ar(bus, VSTPlugin.ar(In.ar(bus, 2), 2));
    ReplaceOut.ar(bus, VSTPlugin.ar(In.ar(bus, 2), numOut: `[2]));
  }).add;
)

//APRO IL PLUGIN, LO ASSOCIO A UN UGEN E LO CONTROLLO CON VSTCONTROLLER
~fx = VSTPluginController(Synth(\effetto, [\bus, ~myBus])).open("CMLS-Juce.dll");
```

```
//SELEZIONA LA FORMA D'ONDA DA UNA PORTA OSC
~r=NetAddr.new("127.0.0.1", 57120);
OSCdef(\synthSelection, {arg selector;|
  selector[1].postln;
  switch(selector[1],
    1, {
      ~b1.free;
      ~b2.free;
      ~b3.free;
      ~b4.free;
      ~b5.free;
      ~b1 = Synth.new(\Sin, [freq: ~noteSet, bus: ~myBus]);
      ~b2 = Synth.new(\Sin, [freq: ~noteSet, bus: ~myBus]);
      ~b3 = Synth.new(\Sin, [freq: ~noteSet, bus: ~myBus]);
      ~b4 = Synth.new(\Sin, [freq: ~noteSet, bus: ~myBus]);
      ~b5 = Synth.new(\Sin, [freq: ~noteSet, bus: ~myBus]);
    },
```

value is 2, '\Sawtooth' if the value is 3, and '\Square' if the value is 4. All new synths are created with the frequency '~noteSet' and sent to the bus '~myBus'. This allows dynamically changing the type of sound generated by the synths based on OSC inputs. In the picture we plot only the section for choosing the sinusoid.

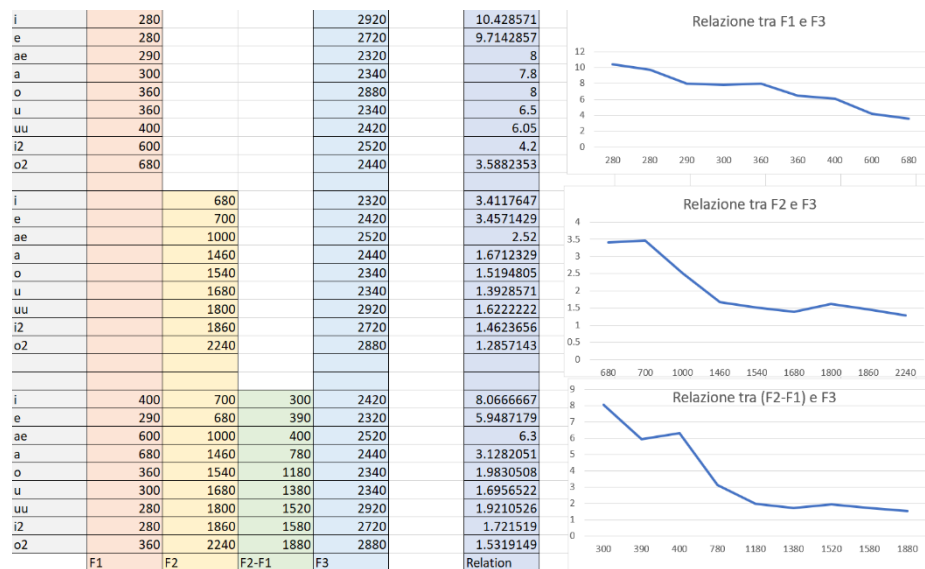
The SuperCollider code establishes an OSC connection to the address '127.0.0.1' on port '57120' and defines a listener that responds to messages sent to the address '/menu'. Upon receiving a message, it prints the value of the second element of the message 'selector[1]'. Depending on this value, it releases five existing synths ('~b1', '~b2', '~b3', '~b4', '~b5') and creates new ones with specific waveforms: '\Sin' if the value is 1, '\Impulse' if the

Detune, Unison and Volume are the parameters that the user can chose from the GUI through the three knobs.

```
OSCdef.all;
OSCdef(\detune, {arg detOsc;
  ~b1.set(\det, detOsc);
  ~b2.set(\det, detOsc);
  ~b3.set(\det, detOsc);
  ~b4.set(\det, detOsc);
  ~b5.set(\det, detOsc);
  postln(detOsc);
}, '/Detune', ~r);
OSCdef(\unison, {arg g1, g2, g3, g4, g5, p1, p2, p3, p4, p5;
  ~b1.set(\gate2, g1, \pan, p1);
  ~b2.set(\gate2, g2, \pan, p2);
  ~b3.set(\gate2, g3, \pan, p3);
  ~b4.set(\gate2, g4, \pan, p4);
  ~b5.set(\gate2, g5, \pan, p5);
}, '/Unison', ~r);
OSCdef(\vol, {arg volOsc;
  ~b1.set(\amp, volOsc);
  ~b2.set(\amp, volOsc);
  ~b3.set(\amp, volOsc);
  ~b4.set(\amp, volOsc);
  ~b5.set(\amp, volOsc);
}, '/Volume', ~r);
```




In order to develop the filters on the VST, using Juce, we consider F1 and F2 (the two cut-off frequencies of the two formants that correspond to the two band-pass filters) and through some mathematical computations we found the third formant (F3) of the third filter. Through these filters the sound wave coming from SuperCollider can be elaborated in order to give as an output similar to a human voice. While we are moving our hand (with the glove), F1, F2 and F3 change to modify the input signal. The sound's change is heard through the differentiation between vowels.



The most important part of our code on Juce is in the process block, for this reason we show here the most important parts of it:

```
void CMLSJuceAudioProcessor::oscMessageReceived(const juce::OSCMessage& message) { // r
    if (message.size() == 1 && message[0].isInt32()) {
        if (message.getAddressPattern() == "/handMovement/x") {
            midi[0] = message[0].getInt32();
            //std::printf("value obtained = %i\n", message[0].getInt32());
        } else if (message.getAddressPattern() == "/handMovement/y") {
            midi[1] = message[0].getInt32();
            //std::cout << "message obtained = " << message[0].getInt32() << std::endl;
        }
    }
}
```

This part is responsible for receiving the OSC messages coming from SuperCollider and storing them in some variables.

With this part of code instead we are calculating the three frequencies of the formants from the OSC input messages.

```
void CMLSJuceAudioProcessor::calcFreqs() { // calculating the f
    freq1 = (midi[0]/127.f)*f1_band + f1_min;
    freq2 = (midi[1]/127.f)*f2_band + f2_min;
    freq3 = (1-((freq2-freq1)/(f2_min+f2_band-f1_min)))*(1-((freq1)/(f1_min+f1_band)))*f3_band + f3_min;
}
```

```
calcFreqs();
```

```
F1.setType(juce::dsp::StateVariableTPTFilterType::bandpass);
F2.setType(juce::dsp::StateVariableTPTFilterType::bandpass);
F3.setType(juce::dsp::StateVariableTPTFilterType::bandpass);
```

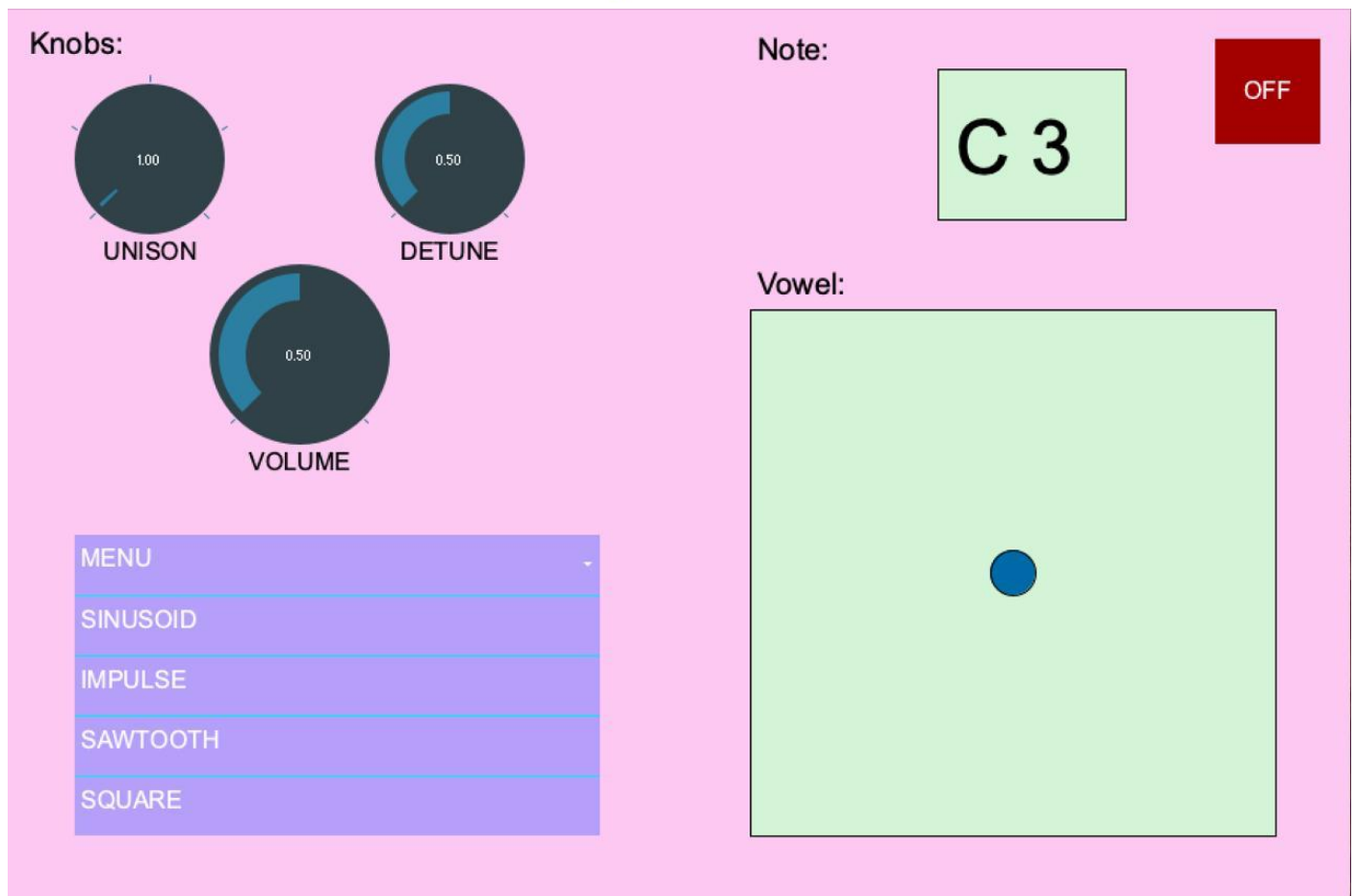
```
F1.setCutoffFrequency(freq1);
F2.setCutoffFrequency(freq2);
F3.setCutoffFrequency(freq3);
```

```
F1.setResonance(10);
F2.setResonance(8);
F3.setResonance(2);
```

```
F1.process(ctx1);
F2.process(ctx2);
F3.process(ctx3);
```

And finally in this part we are processing the three copies of the input signal with three band pass filters with different cut-off frequencies obtained from the arduino-MIDI values.

PROCESSING:



The picture above is our graphical interface created with Processing.

The three knobs are linked with SuperCollider and they control:

- The volume of the output
- The detune of the waves, it is a variable that creates a small oscillation of the frequency
- Unison, with this parameter the user can change the pan parameter on SuperCollider and this effect is reflected in the spatialization of the sound

The drop-down menu instead lets the user select the wave shape.

The button OFF let the user shut down the interface and stops the interpreter of SuperCollider

Inside the higher green rectangular the user can see which note is played with the hand. The number explains the octave.

The green square shows a point that moves following the motion of the hand.

CONCLUSIONS:

As we can hear, since we are controlling only three formants, the output is not exactly similar to a human voice; to reach this goal, are needed more than three formants, but in our case is not possible as we have not enough physical input to control them.

We have also noticed that the resistor sensor is not so good because the note detection is not so precise. To have a better result it is suggested using a more rigid resistor sensor.

The initial goal was to detect the notes through the spatial orientation of the hand, but as we tested it with the MPU6050 sensor we discovered that it was not possible since it is not a high precision sensor. In order to have an identification of the position and the velocity of the hand it is needed a more expensive sensor.