COMPUTER MUSIC - LANGUAGES AND SYSTEMS

HOMEWORK N.3:

DESIGN AND IMPLEMENTATION OF A COMPUTER MUSIC SYSTEM

Color Mixer

Professor: **Fabio Antonacci**

Academic year: 2022-2023

Group 7 - Lemonjuice:
Guglielmo Fratticioli (Matr: 214322)
Chiara Lunghi (Matr: 233195)
Alessandra Moro (Matr: 102283)
Elia Pirrello (Matr: 223050)

Contents

1 Introduction

The objective of the project was to develop an audio application employing JUCE, Processing, and Arduino. The application aimed to enhance the listening experience by blending the multitracks of a song based on the color detected using a sensor.

To capture color information, an RGB color sensor was connected to an Arduino board. Furthermore, an LCD RGB backlight screen was integrated into the system, displaying the application's name, "Color Mixer," illuminated with the scanned color and presenting the corresponding RGB values. The RGB values were then sent to a serial port and processed using the Processing programming language.

1.1 Interaction scheme

The interaction scheme between Arduino, Processing, and Juce is depicted in the following block scheme. It highlights the flow of serial communication, involving the exchange of scanned RGB values between Arduino and Processing.
Additionally, the scheme illustrates the exchange of OSC messages containing the computed volumes of each track, which occurs between Processing and Juce.

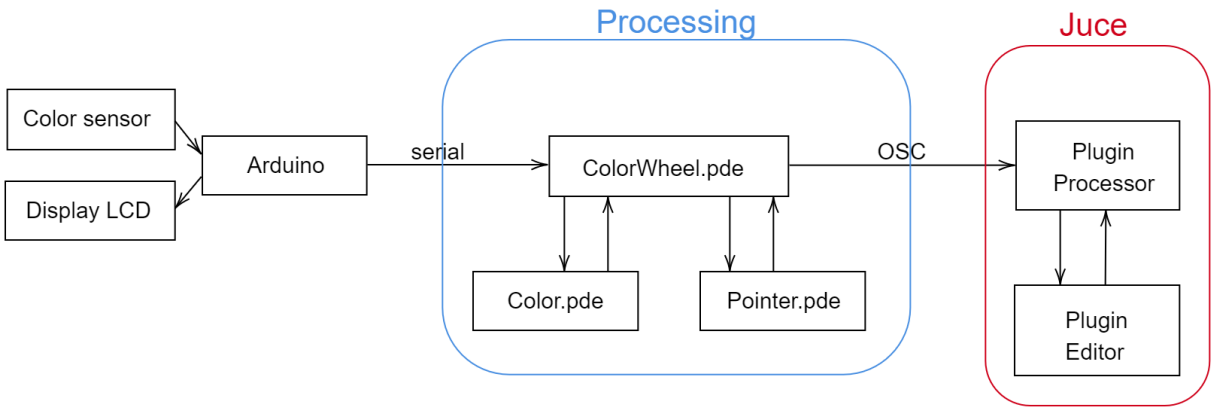

Figure 1: Block scheme

2 Arduino

The purpose of using the Arduino Uno R3 board in this project is to enable communication between the color sensor (DFRobot_TCS34725), the computer, and the LCD display (DFRobot_RGBLCD1602). The sensor and display are connected to the Arduino; which is then connected via USB cable to the computer.

In the `setup()` function, serial communication is started, after that, sensor and lcd screen are initialized.

```
void setup()
{
  Serial.begin(9600);
  tcs.begin();

  //LCD initialization
  lcd.init();
  lcd.clear();
  lcd.setBacklight(true);
  lcd.setCursor(3,0);
  lcd.print("ColorMixer");
  .
  .
  .
}
```

In the loop() function, Arduino, acquires from the sensor three values (RGB) and their sum (denoted as "clear") through the `getRGBC()` method. These have to be normalized on their sum (clear) and multiplied by 255, to obtain standard 0-255 values. The color is set on the LCD display (via the `setRGB()` method) and sent on the serial port.

```
void loop() {
  uint16_t clear, red, green, blue;
  tcs.getRGBC(&red, &green, &blue, &clear);
  // turn off LED
  tcs.lock();
  .
  .
  //COLOR LCD SCREEN
  lcd.setRGB(r, g, b);
  .
  .
}
```

3 Calculation of distances and volumes in Processing

The appropriate volume levels for the different tracks in the multitrack audio are computed in Processing. This involved calculating the distances between the actual color scanned and a set of predefined fixed colors associated with each track in three-dimensional space using the HSV (hue, saturation, value) color model:

```
float calculateDistance(float[] actualColor_hsv, int pointIndex) {
  // Convert HSV into XYZ coordinates
  // ...

  // Compute distance
  float distance = sqrt(pow((fixed_x - actual_x), 2) + pow((fixed_y - actual_y), 2) + pow((fixed_z - actual_z), 2));

  return distance;
}
```

The actual volumes for each track are obtained through normalization of the distances. The nearest track has a volume of one, while the farthest track has a volume of zero. In

cases where the distances between the actual color and the fixed colors were all within a certain threshold, indicating similar distances, all the volumes are set to 1.

```
float[] computeVolumes(float[] actualColor_hsv) {
  // Calculate distances
  // ...

  // Compute volumes based on distances

  boolean allZeros = true;

  for (int i = 0; i < distances.length; i++) {
    //min distance = max weight. Then normalize on distance excursion
      if((maxDist-minDist)<10){
        weights[i] = 1;
      }else weights[i] = (maxDist-distances[i])/(maxDist-minDist);
    if(weights[i]!=0) allZeros = false;
    //if(weights[i]<0.01) weights[i] = 0;
    print("d ",i," = ",distances[i]," ");
    print("w ",i," = ",weights[i]," ");
    println();
  }

  if (allZeros){
    float[] ones = new float[5];
    for (int i = 0; i < nrColors; i++) ones[i] = 1;
    weights = ones;
  }

  return weights;
}
```

Therefore, the unique immersive listening experience that ColorMixer provides is the result of dynamic adjustment of the multitrack audio volume in response to the scanned color.

4 Juce implementation

In the Juce application a multitrack mixer is implemented with basic playback controls. Once the user has selected the song to mix, the stems inside the song's folder are loaded in an array named `trackBuffers[]`. A related array named `trackVolumes[]` stores the volumes levels of the different tracks that depends on the detected colors. The `isPlaying` and

`isStopped` flags are declared to control the playback behaviour. For instance, `isPlaying` is checked before iterating through the buffers while the variables `loopStart` and `loopEnd` will control the time bounds of the looping region. The iteration over the time samples

is performed in a *for cycle* and for every new audio block the final output is obtained by summing the samples in the different track buffers weighted by the track volumes.

```
    if(isPlaying) {

        for( int sample = 0; sample<buffSize; sample++ )
        {
            for( int n = 0; n < ntracks; n++ )
            {
                if( playHead >= loopEnd) playHead = loopStart;

                leftChannel[sample] += trackVolumes[n] * (trackBuffers[n]->getSample(0, playHead));
                rightChannel[sample] += trackVolumes[n] * (trackBuffers[n]->getSample(1, playHead));
            }
        playHead ++;

        }
}
```

5 Graphic User Interface

The GUI consists of two adjacent windows: the first one created using Processing, and the second one created using Juce.

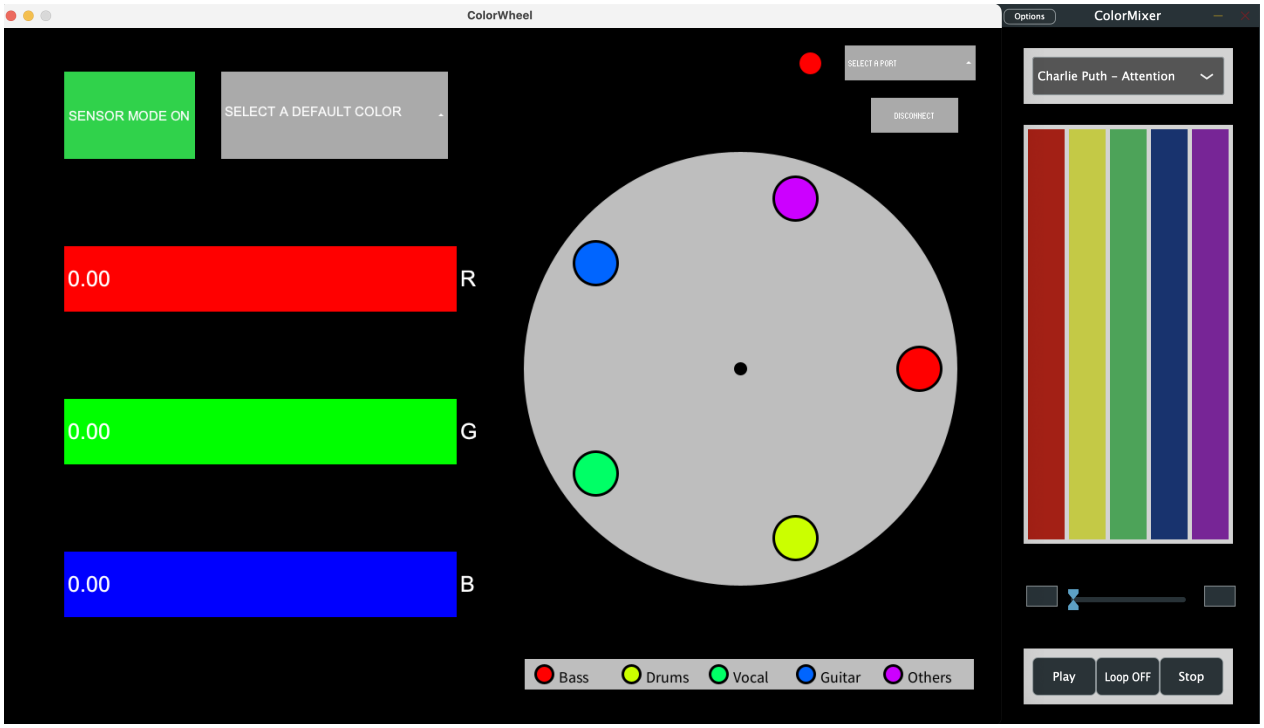

Figure 2: GUI

5.1 Processing

The Processing component, composed of a core file (`‘ColorWheel.pde‘`) and two classes (`‘Pointer.pde‘` and `‘Colors.pde‘`), receives the three RGB values scanned by the sensor through OSC messages. These values are then passed as input to the `rgbToHsb` function, which is declared in the Pointer class. This function converts the RGB values into HSB values and returns them as a vector (`hsv_colors[]`).

The `hsv_colors` vector is then used as an argument for the `updatePointer` function, also declared in the Pointer class. This function updates the position of the pointer in the color wheel using the new HSB values.

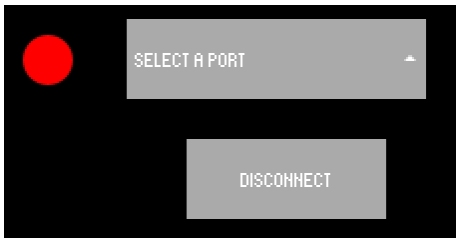

Figure 3: Port Selection

Within this GUI, users have the option to select a port for connecting the Color Sensor. If the selected port is available, the sensor establishes a connection and the red circle located next to the port menu, functioning as a status LED, turns green.
At any point, users can press the `"Disconnect"` button to sever the connection and reset all the values.

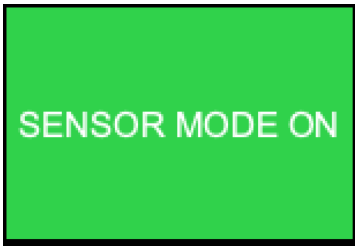

Figure 4: Sensor Mode ON

By default, the ColorMixer operates in `"Sensor Mode ON"`, where the three sliders and the preset Pop-Up-Menu remain locked. Once the sensor starts transmitting RGB values, both the RGB sliders and the HSV color wheel will display them real-time.

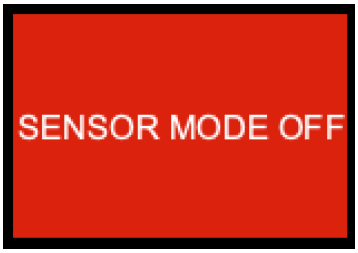

Figure 5: Sensor Mode OFF

When the user selects `"Sensor Mode OFF"`, the color sensor disconnects, and the status LED turns yellow to indicate that the port is still connected, but the sensor is not sending values. Additionally, the sliders and the default colors Pop-Up-Menu become interactive.
Indeed, in this mode, users can manually choose RGB values by adjusting the sliders.

Another feature available is to select a preset color from the popup menu:

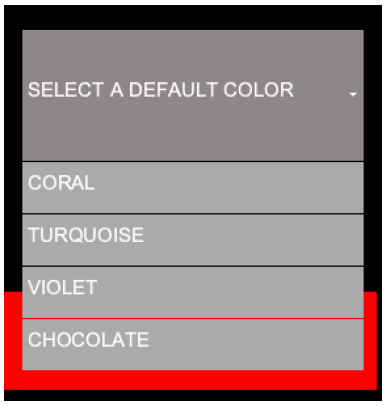

Figure 6: Pre-Set Menu

5.1.1 `public float[] rgbToHsb(int r, int g, int b)`

The conversion function, which transforms the input RGB values to their corresponding HSV values, is implemented in the Pointer class. This function is then called within the draw function of the core file, `ColorWheel.pde`.

```
float h=0, s=0, br;
float max = max(r, max(g, b)); // Find the maximum value among r, g, and b
float min = min(r, min(g, b)); // Find the minimum value among r, g, and b

br = max * 100 / 255.0; // Calculate the brightness value as a percentage

float delta = max - min;

if (max != 0) {
  s = delta * 100 / max; // Calculate the saturation value as a percentage

  // Calculate the hue value based on the position of the maximum value
  if (r == max) {
       h = (g - b) / delta;
  } else if (g == max) {
    h = 2 + (b - r) / delta;
  } else {
    h = 4 + (r - g) / delta;
  }

  h *= 60; // Convert hue to degrees

  if (h < 0){
    h += 360; // Wrap hue value if it is negative
  }
  if (delta == 0){
    h = 0; // Set hue to 0 if the difference is 0, i.e., all RGB values are the same
    }
 }

hsv_colors[0] = h;
hsv_colors[1] = s;
hsv_colors[2] = br;

return hsv_colors;
```

5.1.2 `public void updatePointer(float hue, float sat, float bri)`

This function, implemented within the Pointer class, is responsible for creating the pointer and updating its position based on the converted HSV values. It is called within the draw function of the core file, `ColorWheel.pde`.

```
hueRadians = radians(hue); // Convert the hue value from degrees to radians

// Map the saturation value from the range 0-100 to the range 0-205 (distance of the tracks circles from the center)
saturation = map(sat,0,100,0,205);

// Calculate the x-coordinate and y-coordinate of the ellipse based on the hue and saturation values
x = centerX + cos(hueRadians) * saturation;
y = centerY + sin(hueRadians) * saturation;

// Map the brightness value from the range 0-100 to the range 10-50 (min and max radius of the pointer)
volume = map(bri, 0,100,10,50);

colorMode(HSB, 360,100,100); // Set the color mode to HSB
c = color(hue,sat,bri); // Create a color using the hue, saturation, and brightness values
fill(c); // Set the fill color for the ellipse
strokeWeight(5);
stroke(0);
ellipse(x,y,volume,volume); // Draw an ellipse at the calculated position with a size based on the volume value
```

5.2 Juce

The Juce GUI is primarily focused on audio settings.
At the top of the GUI, there is a Pop-Up Menu that allows the user to select a specific song to work on from a predefined set of songs.

In the center of the GUI, there are five level meters. Each level meter corresponds to a specific track in the audio processing system. The color of each level meter is associated with the `Processing` color chosen for that particular track. These level meters provide real-time visualization of the volume variation for each track based on the selected `Processing` color.

The GUI provides the functionality to select a specific portion of the song. This is achieved using a two-thumb value slider. The user can adjust the position of the two thumbs to define the desired portion of the song.

Figure 7: Slider

The Play button allows the user to start playing the song. If the Play button is pressed again while the song is playing, it pauses the song.
When the user turns on the Loop button, the loop function is activated, and the selected

portion of the song will be played in a loop.
The Stop button stops the playback of the song. If the loop function is active, it will start again from the beginning of the loop, otherwise it will start from the beginning of the song.

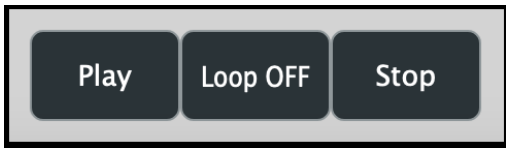

Figure 8: Buttons

6 Conclusions

In conclusion, our project combined JUCE, Processing, and Arduino to create an audio application that dynamically mixes multitracks based on scanned colors. The integration of an RGB color sensor and LCD backlight screen enhanced the user experience. This project demonstrates the exciting potential of merging music, color, and technology for immersive audio experiences.

JUCE has proven to be a comprehensive and stable framework for developing audio applications. Being based on C++, it ensured higher performance compared to languages like JavaScript. The main drawback encountered was the lack of built-in serial communication functions, which were crucial for our project. As a solution, we intercepted the serial communication using Processing and sent the information to JUCE through the OSC (Open Sound Control) protocol.