

# Mixxx-based cdj

**A raspberry-pi Cdj project realized with the usage of mixxx for the audio software and Arduino Leonardo for input reading.**




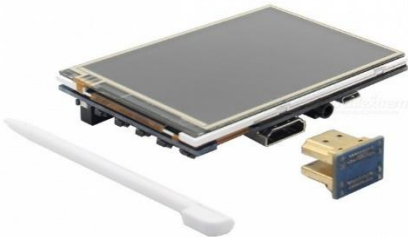
**The MIDI protocol is used for the communication between Arduino and the Raspberry.**



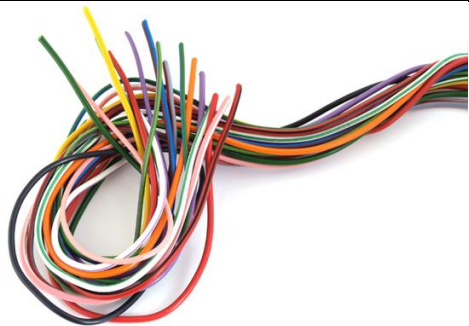




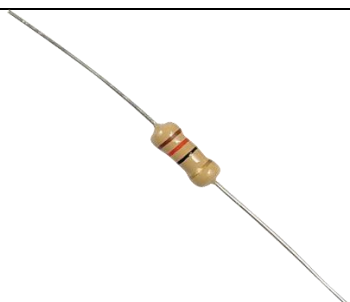
**Lorenzo Balducci**

**github: <https://github.com/LorenzoBalducci96/>**

## Used material (for 1 cdj):

cdj	<p>In this project we will refer to the pioneer cdj 200.</p> <p>You can use a broken one (all the buttons you want to use, the pitch and the jog must be in working conditions).</p> <p>You can even build your own using buttons, switch, rotary encoders...</p>	
raspberry pi 3 b+	<p>Not tested on other raspberry pi.</p> <p>We will use for running mixxx with our personalized skin.</p>	
Micro Sd card	<p>You need a micro sd card because the raspberry pi doesn't have an internal memory and uses an external sd card for the mass storage.</p>	
Raspberry pi screen	<p>I used a 3.5" screen with a 480x320 resolution with touch screen support.</p> <p>You can use a different one but consider that the mixxx skin is all pixel-binded, so if you choose a screen with a different resolution, you will have to scale or adapt my skin. Moreover the waveform will be very fluid with a solid 60fps in a 320p resolution, but will be laggy on higher res.</p>	

Arduino leonardo	<p>You have to use the Leonardo version of Arduino because we will use the serial communication over usb.</p> <p>We will use Arduino for continuously reading input data from the cdj and sending over MIDI protocol to the raspberry pi.</p>	
Usb sound card	<p>We will use an external sound card for a better sound quality and for reduce latency on mixxxx.</p> <p>(You can find very good and cheap usb sound cards but also the 3.5mm jack audio on the raspberry isn't all that shame).</p>	
Electric wires	Used for Making connections between cdj and Arduino.	
Raspberry power supply	Classic 5v usb plug (Raccomanded minimum output of 2.5A).	

Arduino power supply	You should provide an external power source for Arduino unless you will have a lot of noise on analog reading. Raccmended a 9v.	
4.7K Resistors	Used for driving leds. If you don't want a "bright" cdj you don't need resistors.	

Of course screwdrivers, welder, and heat-resistant tape.

## Fase 1: preparing the cdj

The first step is to prepare the cdj for the connection to the Arduino board.

If you decided to build your own cdj feel free to jump to the next step because here, we are going to analyze the disassembly process of pioneer cdj 200.

As stated before, we want to throw away the cdj player and the logical unit from the cdj because we will use just the “buttons”, and we will inject another control unit.

A good video guide that you can follow is this

<https://www.youtube.com/watch?v=enjz8VNCzuE> but please note that you don't have to disassembly the boards on the upper side of the cdj (the silver one) but you just have to “clean” the bottom side (the black one).

In the end you should have this:

The silver upper part of the cdj:



Disconnected from the bottom black part (leave alone the little flat on the upper part)



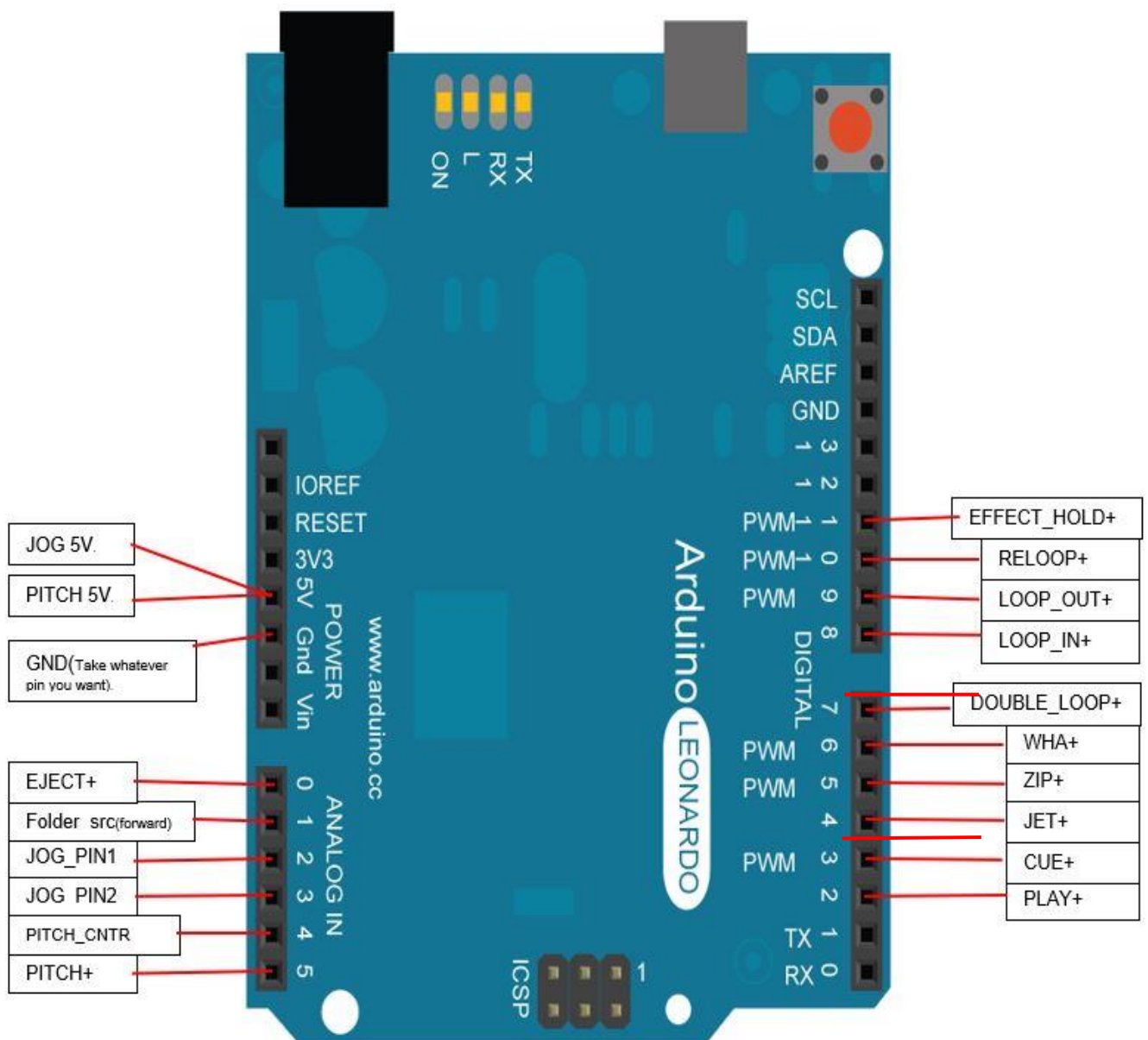
## **Fase 2: connecting all the buttons, the jog and the pitch to Arduino.**

Notice: all the buttons on the cdj board shares the same ground, so you have to connect to the Arduino gnd pin just one ground. You can check with a multimeter that all the ground are nearly short-circuited.

Notice2: each end of the wire that will go to the cdj board must be welded.  
Each end of the cable going to Arduino can just be plugged in.

In the Arduino code we will read all the buttons with the pull\_up method (there are some exception that will be explained).

The decided standard for connections is this:



Some photos and explanations about the wiring on the cdj...

The CUE/PLAY BOARD.





The folder search (forward) button: note that there is a voltage divider between the navigations buttons.

So we connect just one button and in the Arduino code we will analyze the voltage for guessing the button pressed.

The Button's group interested:



From now we will refer to this group of buttons as seek buttons.

The black wire is connected to the front led with a 4.7Kohm resistor (more detailed instructions at the end of the project).

The remaining 2 red wires are connected the the + pins of the cue and play buttons.

The gray wire is connected to GND of play button and goes the the gnd pin of Arduino. (all the gnd pins are nearly short circuited so you can take another pin if you want).



There isn't a clear view on the photo but consider that the loop in, loop out, and exit/reloop buttons have the positive pin on the left, the beat loop button has the positive pin on the right.

Once again: if you are not interested to have led working in this cdj you can skip this step.



9

Now let's take a look in the effects section.

Here you have to connect the 3 effects button and the hold button (make shure to take all the + pins because as you can see in the picture, the jet and the zip buttons have the positive pin on the left, but the wah and the hold buttons have the positive pins on the right).

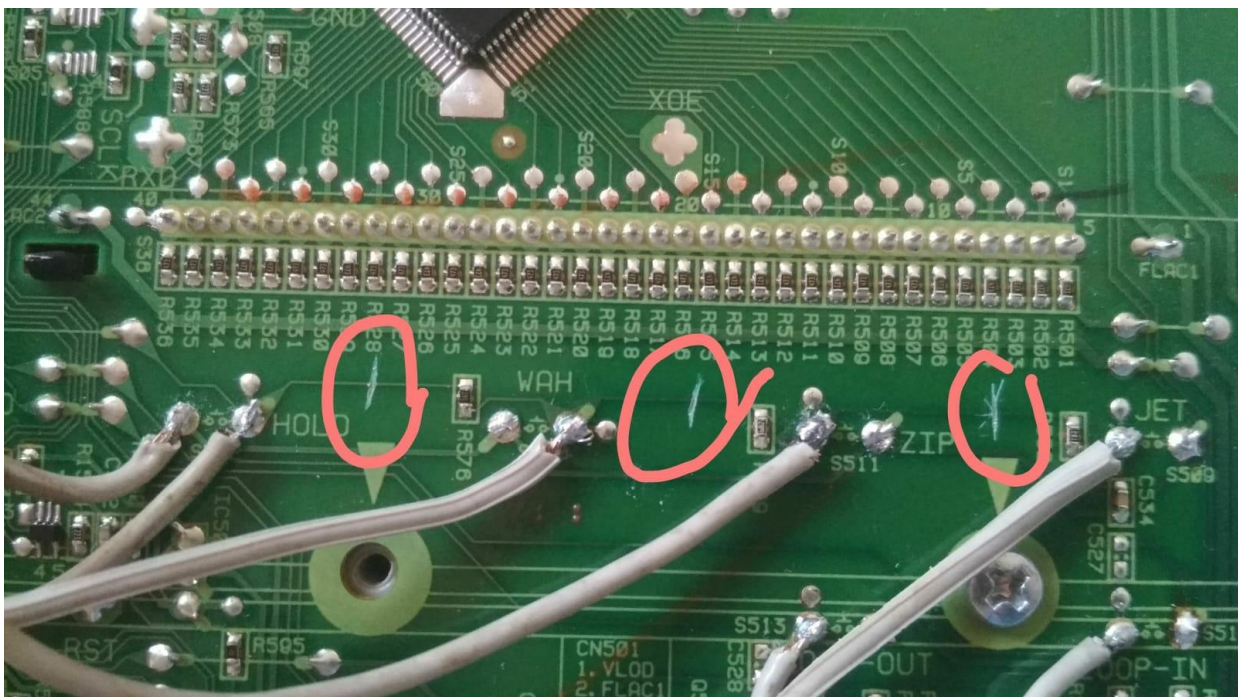
On the left pin of the hold button I took another gnd and I connected that too on the gnd Arduino pin.

As the search section, the effect section of the cdj 200 is realized with a tension divider, so we could have connected just one button and make in the Arduino code the same analog reading of the search section for guessing the pressed buttons.

Unfortunately the readings on a tension divider have some problems: the Arduino code and the analog readings takes precious time so we will have a less responsive cdj but the most important fact is that we can press just one button at the time.

From the moment that I decided to use the effects buttons for controlling hotcues on mixxx (the three effects are 3 hotcues and the hold button is the shift button for deleting them like a cdj 2000) we will usually press more than on button at the time.

So you have to break the connections between the buttons as shown in this photo:



The last thing to connect on the main board is the eject button.

The eject button is inserted in a tension divider as for the search buttons and as for the effects buttons.

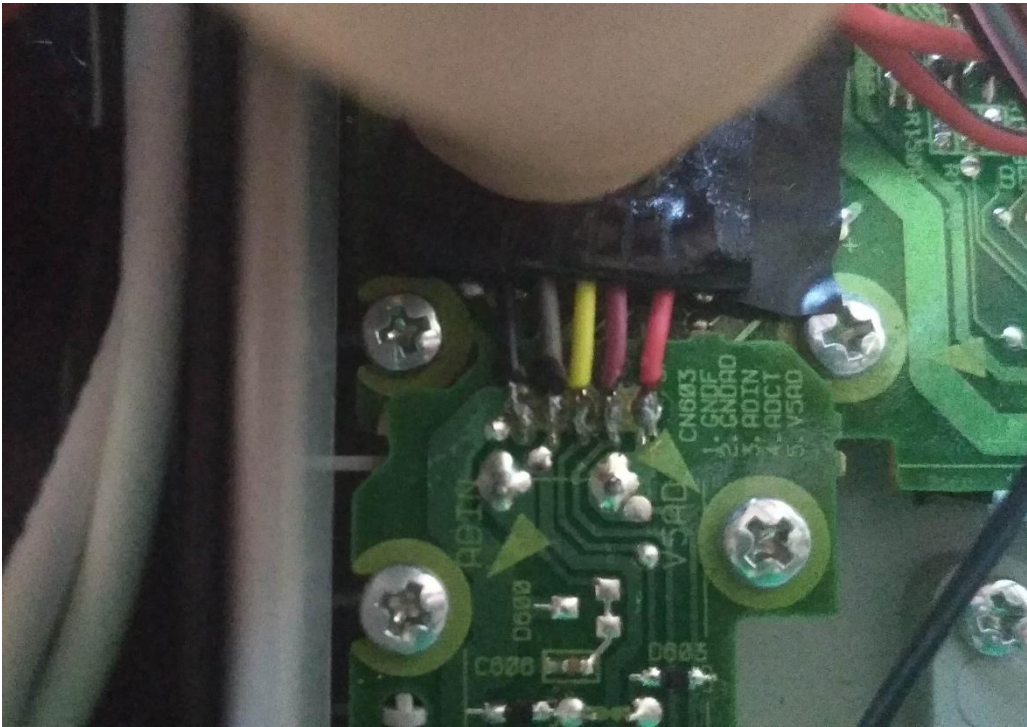
The buttons in questions are those in the picture.

From now we will refer to this group of buttons as function buttons.





The PITCH BOARD.



Here you have to connect these wires to the Arduino as shown in the initial schema:

From the moment that the logical names used in the initial schema are different from the ones used from pioneer in this board I'm going to make a table with the correspondence.

NOTE: THE GNDP AND GNDAD CONNECTIONS MAKE THE THINGS WORKING WELL BUT I DON'T KNOW WHY. I DON'T KNOW WHY THIS BOARD HAS 2 GND PINS. IF YOU SHORT THESE PINS NOTHING SEEMS TO HAPPEN AND THE THINGS WORK WELL.

IF YOU DON'T CONNECT THE SECOND GND PIN (GNDAD) YOU WILL HAVE MORE NOISE ON THE ANALOG READING) IF YOU DON'T CONNECT THE FIRST GND PIN (GNDP) NOTHING WILL WORK.

GNDP (black cable): is connected to all others gnd pins (you should have one more free gnd pin on Arduino).

GNDAD (gray cable): no I do not know exactly why you have to connect this pin but you have to.

For a good cable management I connected this pins to the gnd of the jogwheel.

ADIN (yellow cable): referred ad PITCH+ in the Arduino connection schema so connect it to the A5 pin.

ADCT (purple / red cable): referred ad PITCH\_CNTR in the Arduino connection schema so connect it to the A4 pin.

V5AD: connect to 5v pin on Arduino Leonardo.

Quick explanation:

In the ADIN pin you will read a voltage between 0 and 5v (if connected to 5v power source) depending on the pitch position.

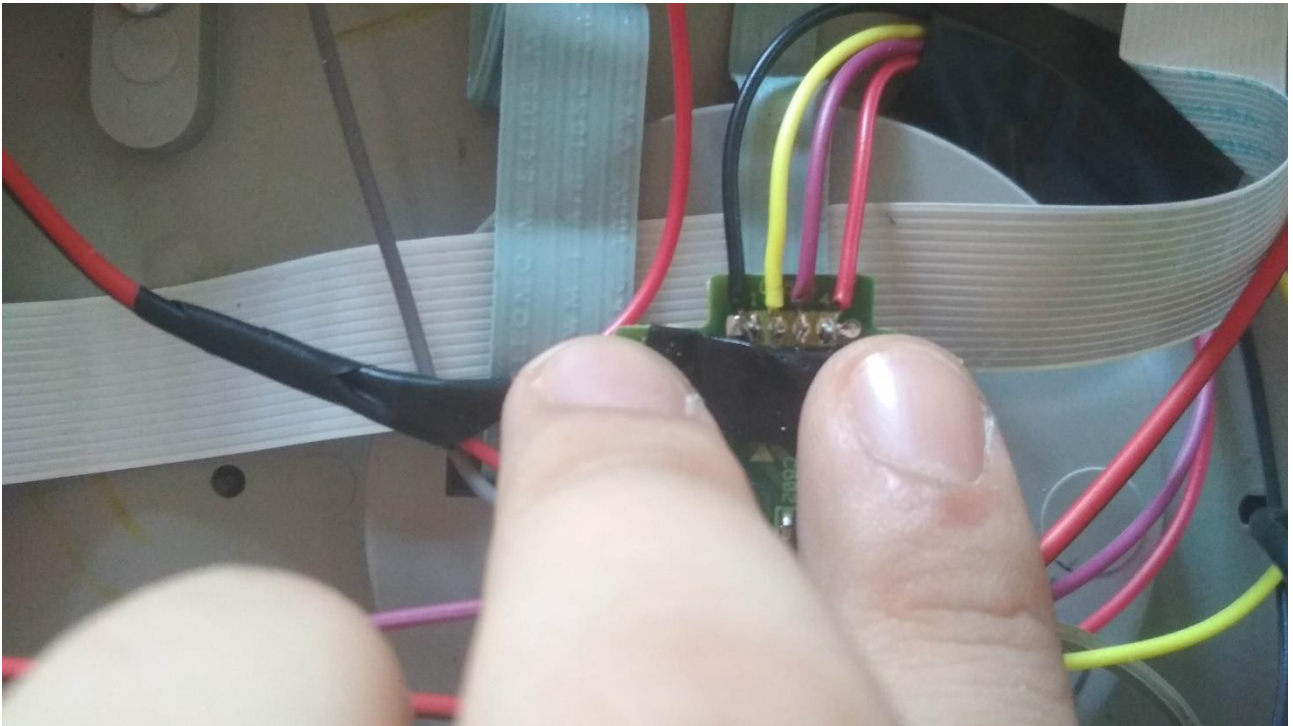
You could think that is very simple and if you read 5v the pitch is in the upper position(it's right!), if you read 0v the pitch is in the bottom position (it's right!) and if you read 2.5v this means that the pitch is exactly in the middle position (IT'S NOT RIGHT!).

When the pitch fader is exactly in the middle position in the ADIN pin you will read the same voltage shown in the ADCT.

In other words: in the ADCT pin you will read a static voltage indicating the voltage that you should read in the ADIN pin if the pitch is in the middle position.

So in the Arduino code we have to make shiftings of readed ADIN value in function of the ADCT value.

The JOG.



The jog drives a standard Rotary encoder: an encoder that use two digital square signal for coding left movement or right movement (in the Arduino code we will use a library for read those values easily).

Wires connections here are:

Black wire: connected the the GNDAD pin of the pitch as said before.

Yellow wire: connected to JOG\_PIN1 so pin A2 (see Arduino connections schema).

Purple wire: connected to JOG\_PIN2 so pin A3 (see Arduino connections schema).

Red wire: connected to 5v pin for power source.



### **Fase 3: preparing Arduino.**

First of all you have to put Arduino in the black bottom shield of the cdj.  
For a well done job you can use double sided adhesive tape.  
I used black heat resistant tape.



The twisted wire of the power plug goes to the 9v Arduino power supply.  
The other end of the USB cable seen in the picture is a normal USB plug that you will have to connect to the Raspberry but for now just connect to the PC for loading the Arduino sketch.

In the pc, once you have installed Arduino ide (<https://www.arduino.cc/en/Main/Software>) and you downloaded the Arduino sketch from my repository (<https://github.com/LorenzoBalducci96/custom-cdj/blob/master/midi-controlller-for-cdj.ino>) you have to modify (if you want) the sketch and then load into your Arduino. The used library are: "MIDIUSB" (easily findable on the official library repository) and "RotaryEncoder" (in particular I used the one in this repository: <https://github.com/sivanandareddy/arduino-rotary-encoder-with-velocity>).

Let's make some consideration on the Arduino code:

After the warm up, in the loop It basically:

<pre>void loop() {   readButtons();   readControl();   readIntensity();   readRotary();   playNotes();   delay(1); }</pre>	<ol style="list-style-type: none"><li>1. read all inputs (readButtons)</li><li>2. compare with the previous reading.</li><li>3. if there is a change there is a message sent and an update of the previous reading.</li><li>4. In the end there is 1 ms delay.(if you want a super responsive cdj you can decrease the delay using delayMicroseconds).</li></ol>
--	--

1) Read buttons: in this function we read all the normal buttons and save into a temporary array.

```
for (int i = 0; i < NUM_BUTTONS; i++)  
{  
  if (digitalRead(buttons[i]) == LOW)  
  {  
    premuti[i] = 1; //registra nell'array dei bottoni che e stato premuto (scrive 1) il tasto i  
    //Serial.println("letto:");  
    //Serial.println(i);  
  }  
  else  
    premuti[i] = 0; //registra nell'array dei bottoni che non e stato premuto (scrive 0) il tasto i  
}
```

2) We also read all the tension divider wrapped buttons (seek buttons and functions buttons).

```
int valoreTastierino = analogRead(pinTastiera);  
//Serial.println(valoreTastierino);  
if (valoreTastierino < 50)  
  tastoPremuto = 1;  
else if (valoreTastierino < 230 && valoreTastierino > 185)  
  tastoPremuto = 2;  
else if (valoreTastierino < 395 && valoreTastierino > 345)  
  tastoPremuto = 3;  
else if (valoreTastierino < 545 && valoreTastierino > 495)  
  tastoPremuto = 4;  
else if (valoreTastierino < 690 && valoreTastierino > 640)  
  tastoPremuto = 5;  
else if (valoreTastierino < 845 && valoreTastierino > 795)  
  tastoPremuto = 6;  
else  
  tastoPremuto = 0;
```

This is for the seek buttons, same for the functions buttons...

After the buttons reading we have the readControl function in which we read the control tension of the pitch (as said before there is a tension indicating the center position of the pitch).

```
void readControl(){//analizziamo il valore del ct del pitch
    valoreControl = analogRead(controlPin);
}
```

In the readIntensity function we normally read the value of the pitch (we will have to map this readed value in function of the control value).

```
void readIntensity(){//analizziamo il pitch
{
    valorePitch = analogRead(intensityPot);
}
```

In the readRotary function we read the jog wheel rotation using the rotaryEncoder library mentioned before.

```
void readRotary(){
    rotaryValue = encoder.readEncoder();
}
```

The encoder is defined in the warm up section. (refer to the library documentation for explanation).

In the playNotes function we will compare all the read values with the previousReadings, if there is a change we will sent a midi packed with the MIDIUSB library.

```
for (int i = 0; i < NUM_BUTTONS; i++)
{
    if (premuti[i] != premutiInPrecedenza[i])//se c'e stato un aggiornamento dei tasti c
    {
        if (premuti[i])//se e stato premuto...
        {
            premutiInPrecedenza[i] = true;//registra su i tasti premuti precedentemente che
            //Serial.println("hai premuto");
            //Serial.println(i);
            noteOn(0, notePitches[i], 0);
            MidiUSB.flush();
        }
        else
        {
            premutiInPrecedenza[i] = false;
            //Serial.println("hai lasciato");
            //Serial.println(i);
            noteOff(0, notePitches[i], 0);
            MidiUSB.flush();
        }
    }
}
```

For the jogWheel we send the rotaryValue if there is a change (note that the library give us a value with 0 in the middle, using negative values fo anticlockwise turning and positive numbers for clockwise turning).

From the moment that the dj controlles midi standard expect a positive value centered in 64 we have to translate the value).

```
if(rotaryValue != 0){
  //Serial.println("mando rotary");
  //64 is 0x40 that is the center jog value
  uint8_t valueToSend = 64 - rotaryValue;
  controlChange(0, jogControlValue, valueToSend);
  MidiUSB.flush();
}
```

In the reading of the pitch I applied some filtering techniques because even if you use an external power supply you will still have some noise on analog reading.

Basically there is a window in witch if you readings changes a little, will not provoke new MIDI packets.

If the readings leave the window, you will have a MIDI packet sent and the sliding of the window. This is reached with the salendoScendendo var that indicate if we are sliding up the fader or sliding down. The rest of the code is auto-explained.

In this way you wil have very stable readings if you don't touch the pitch, and you will have an impressive precision (much more than you had in the original cdj).

After these process the readed value is mapped using the controlValue because of the reason explained before.(the value is also mapped from 0-1023 to 0-16383 for respecting the midi standard).

```
if(abs(valorePitch - valorePitchPrecedente) > 1 || ((valorePitch - valorePitchPrecedente) > 0 &&
|salendoScendendo > 0) || ((valorePitch - valorePitchPrecedente) < 0 && salendoScendendo < 0)){
  //Serial.println("mando pitch");
  //Serial.println(valorePitch);
  salendoScendendo = valorePitch - valorePitchPrecedente;
  valorePitchPrecedente = valorePitch;
  if(valorePitch < valoreControl)
    valorePitchMappato = (unsigned int) map(valorePitch, 0, valoreControl, 0, 8191);
  else//valorePitch >= valoreControl
    valorePitchMappato = (unsigned int) map(valorePitch, valoreControl, 1023, 8192, 16383);
  pitchChange(valorePitchMappato);
  MidiUSB.flush();
}
```

In the voltage divider sets of buttons the readings are filtered with the `primaLetturaButtata` for the seek sets of buttons and `primaLetturaTempoButtata` for the second voltage divider set of buttons.

This was made because when you press a button, the voltage doesn't immediately go down (due to the physics of the voltage divider) and when you release the voltage doesn't immediately go up.

With these two vars we are going to check if in the last cycle we read the same value (in case this means that the voltage is stable and we can consider the button pressed).

```
if(tastoPremuto != tastoPrecedentementePremuto){
  if(!primaLetturaButtata){
    primaLetturaButtata = true;
  }
  else{
    primaLetturaButtata = false;
    if(tastoPremuto == 0){
      noteOff(0, (0x20 + tastoPrecedentementePremuto), 0);
      MidiUSB.flush();
      //Serial.println("lascito");
      //Serial.println((tastoPrecedentementePremuto));
    }
    else{
      noteOn(0, 0x20 + tastoPremuto, 0);
      MidiUSB.flush();
      //Serial.println("premuto");
      //Serial.println(tastoPremuto);
    }
    tastoPrecedentementePremuto = tastoPremuto;
  }
}
```

This is for the seek group buttons, same for the function buttons.



## **Fase 4.1: preparing the raspberry pi.**

In this moment you basically have a “midi usb single deck cdj controller” that with the right mapping you can make it working with a plenty of dj mixing software.

We will provide a mapping for mixxx but first let's prepare the raspberry.

You have to make your sd card suitable for the raspberry pi and you have to put there the operating system (Raspbian).

Please refer to the official site for download Noobs and for the installing procedure.

<https://www.raspberrypi.org/downloads/raspbian/>

## **Fase 4.2: installing touch screen drivers and making the screen work.**

There isn't a way for providing standard step for the touch screen drivers installing because this depends on which touch screen you bought.

Normally raspberry touch screen takes power, video signal and touch screen data from the gpio.

The one used by me takes video signal from hdmi so it's maybe a little different from the one that you probably have.

In any case you should have a cd inside the screen box with all the information for driver installing and swintching the output (if needed) from hdmi to gpio.

The touch basically provide a mouse data: when you press on a position of the screen you are moving the mouse in that position an pressing with the left mouse button.

This is perfect for making on skin mixxx buttons working.

## **Fase 4.3: installing mixxx.**

You can install the mixxx version of the Raspbian repo by typing into the terminal:

```
sudo apt-get install mixxx
```

This will install mixxx 2.0.0 not the latest version 2.1 (today is 04/08/2018).

If you want the latest version you have to compile by yourself (not covered in this guide).



## **Fase 4.4: making the cdj working with the mapping file.**

In this moment if you start mixxx (you can find into the graphical ui under audio → mixxx) and you go into preference menu, under the controller section if you see Arduino-Leonardo you are in the good way (of course arduino must be connected over usb cable to the raspberry before starting mixxx).

Now download from my repository the mapping file:

[https://github.com/LorenzoBalducci96/custom-cdj/blob/master/Arduino\\_Leonardo\\_MIDI\\_1.midi.xml](https://github.com/LorenzoBalducci96/custom-cdj/blob/master/Arduino_Leonardo_MIDI_1.midi.xml)

and the script file

<https://github.com/LorenzoBalducci96/custom-cdj/blob/master/arduino-scripts.js>

And put these files into your raspberry pi under:

/usr/share/mixxx/controllers/

For doing this you maybe need root permissions, so you can reach that folder in terminal and copy those files with

sudo cp "source\_path" "destination\_path"

in the other hand you can change the privilege to the mixxx/controllers folder with  
sudo chmod 0777 controllers

After this, restart mixxx and under preferences→controllers→Arduino Leonardo load the preset Arduino\_Leonardo\_MIDI\_1.

Now you will have your controller working with mixxx.

Quick explanation about this mapping:

If you don't understand the sense of some function read the user-guide manual for understanding what kind of mapping is decided and all will make sense.

In the xml file you have all the midi code mapped to a mixxx function.

```
<control>
  <group>[Channel1]</group>
  <key>cue_default</key>
  <description>MIDI Learned from 1 messages.</description>
  <status>0x80</status>
  <midino>0x02</midino>
  <options>
    <normal/>
  </options>
</control>
```

In this example we can see as the 02 midi code with the 80 status code (noteOn so buttonPressed) are mapped with the mixxx cue\_default function.

We don't need anything else for making the cue working.

All the simple buttons like this can be mapped with the learning wizard in mixxx.

More complex function like: shifting of the hotcues, jogwheel and pitch bend rate changing (between +- 10% and +-90%) are script binded.

These function are defined in the js file.

```
<control>
  <group>[Channel1]</group>
  <key>Arduino.wheelTurn</key>
  <status>0xB0</status>
  <midino>0x27</midino>
  <options>
    <script-binding/>
  </options>
</control>
```

For example the turning of the jogwheel (midi control code 27) is mapped with the Arduino.wheelTurn function (described in the js file) infact in the options section there is the script-binding parameter.

In the js file infact we have the Arduino.wheelTurn function.

```
Arduino.wheelTurn = function (channel, control, value, status, group) {  
  
    // B: For a control that centers on 0x40 (64):  
    var newValue = value - 64;  
  
    // --- End choice  
  
    // In either case, register the movement  
    if (engine.isScratching(1)) {  
        engine.scratchTick(1, newValue); // Scratch!  
    } else {  
        engine.setValue(['Channel'+ 1 +'], 'jog', newValue); // Pitch bend  
    }  
}
```

In which we will set the value of the jog rotation or of the scratch position in function of the isScratching value.

The isScratching value is true if the engine recognize that we are scratching (pressing the jogwheel but from the moment that the cdj 200 doesn't have a touchable jogwheel we used the beal loop button for simulate the jog pressure as described before).

The touch of the jogwheel is also a script.

The midiControl message 0x27 with code NoteOn(0x90) or NoteOff(0x80)is mapped to wheelTouch that enable/disable scratching on the engine.

```
Arduino.wheelTouch = function (channel, control, value, status, group) {  
    if ((status & 0xF0) === 0x90) { // If button down  
        //if (value === 0x7F) { // Some wheels send 0x90 on press and release,  
            var alpha = 1.0/8;  
            var beta = alpha/32;  
            engine.scratchEnable(1, 128, 33+1/3, alpha, beta);  
        } else { // If button up  
            engine.scratchDisable(1);  
        }  
    }  
}
```

Another thing script-bound is the shift button that enable us to delete cue points.

```
Arduino.hotCueShift = false;

Arduino.hotCueShiftOn = function (channel, control, value, status, group) {
  Arduino.hotCueShift = true;
}

Arduino.hotCueShiftOff = function (channel, control, value, status, group) {
  Arduino.hotCueShift = false;
}

Arduino.hotCue1Pressed = function (channel, control, value, status, group) {
  if(Arduino.hotCueShift == false)
    engine.setValue(group, 'hotcue_1_activate', 1);
  else
    engine.setValue(group, 'hotcue_1_clear', 1);
}
```

As you can see there is a var hotCueShift that when you press the shift button is set to true and when you release the shift button is set to false.

When you press an hotCue button, this will be activated or deleted in function of the hotCueShift var.

The last thing script binded is the rate range of the pitch.

I decided to enable just 2 rate: +-10% and +-90%.

Unfortunately mixxx doesn't allow al +-100% pitch rate range.

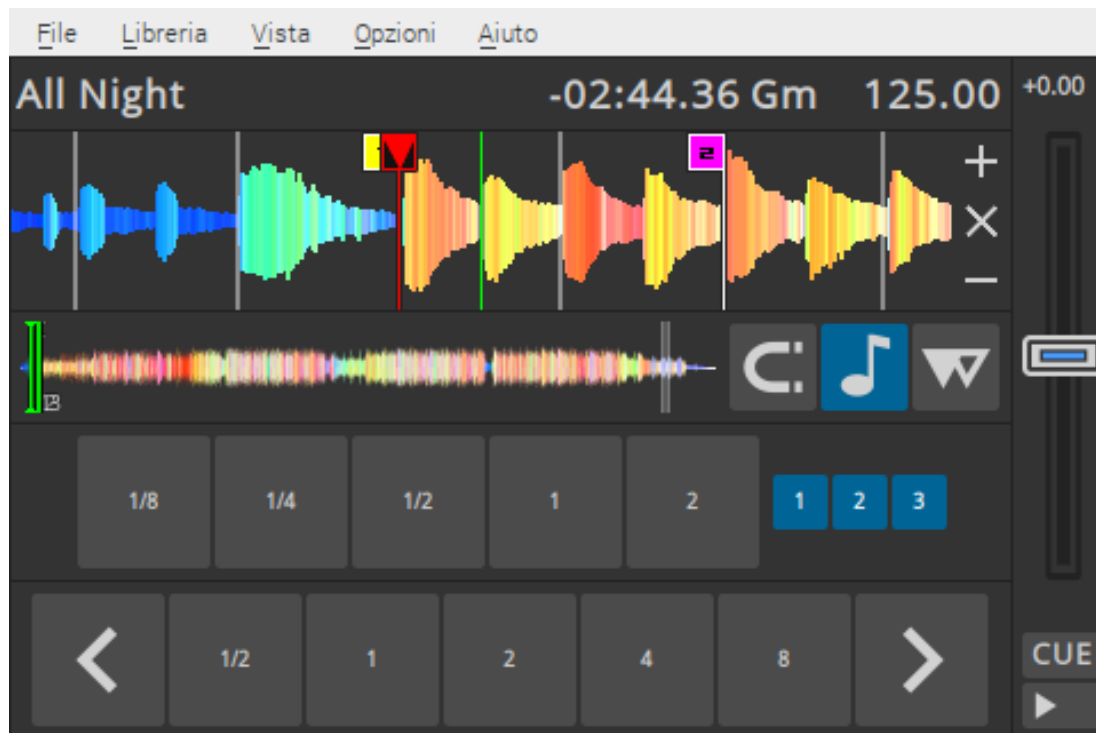
```
Arduino.pitchRange = function (channel, control, value, status, group) {
  var currentRange = Math.round(engine.getValue(group, "rateRange")*100)/100;
  var currentPitch = engine.getValue(group, "rate") * currentRange;

  if(currentRange != 0.1){
    engine.setValue(group, "rateRange", 0.1);
    engine.setValue(group, "rate", currentPitch/0.1);
  }
  else if(currentRange == 0.1){
    engine.setValue(group, "rateRange", 0.9);
    engine.setValue(group, "rate", currentPitch/0.9);
  }
}
```

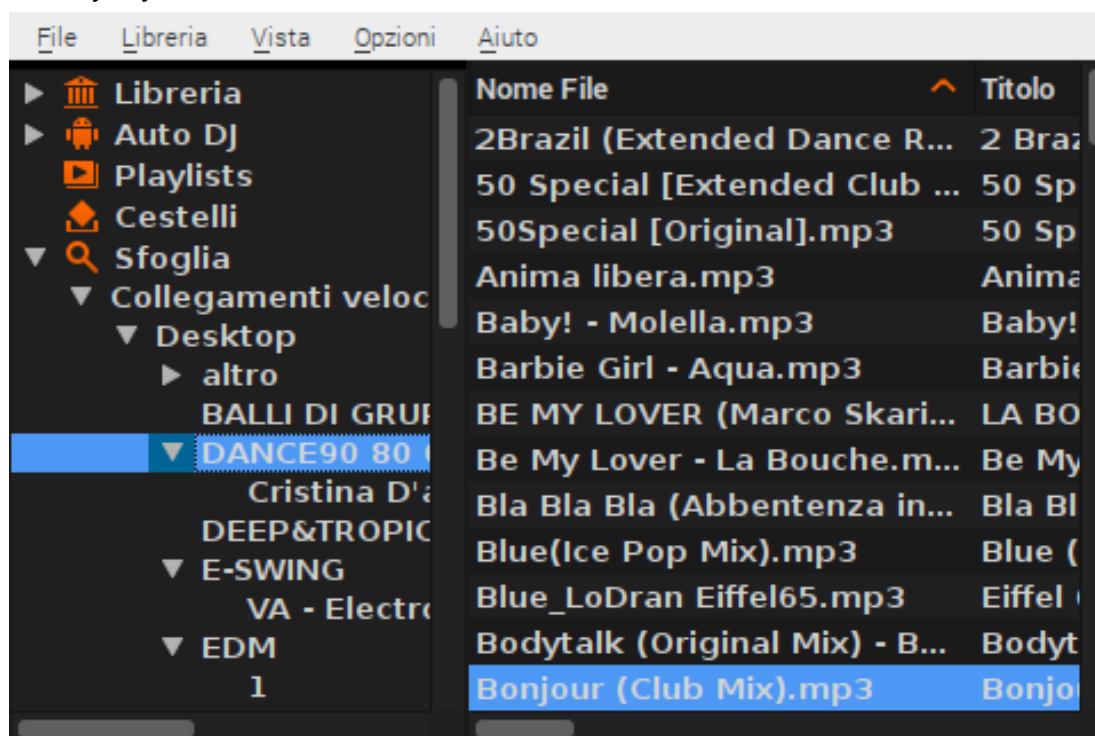
## Fase 4.5: applying single deck skin.

From the moment that we need to use mixxx a cdj player software instead of a mixing tool I created a custom skin:

Performance layout



Library layout



This skin is available inside a zip file on my GitHub

<https://github.com/LorenzoBalducci96/custom-cdj/blob/master/Skin.zip>

It is based on the Deere default skin of mixxx.

Quick explanations of the files:

Skin.xml: most important layout definition (resolution, size of the buttons and piece of skins to display).

In the rest of the xml files describing the interface you will see reference for example to `SquareButtonMinimumSize` `SmallSquareButtonMinimumSize`...

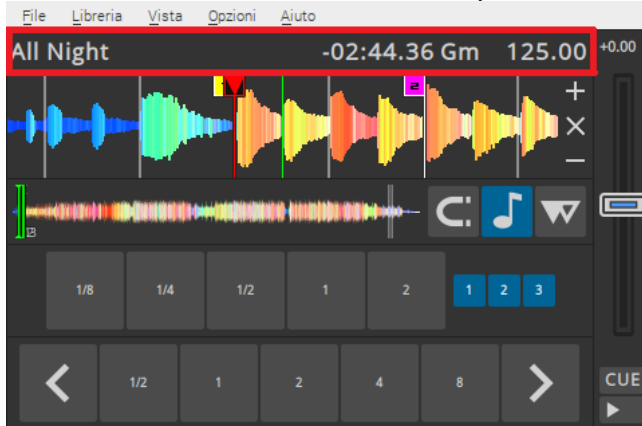
All this size are described in this main file.

Note that the horizontal resolution isn't exactly 480 but 484 because the touch part of my touch screen is a little bigger on the right that the screen.

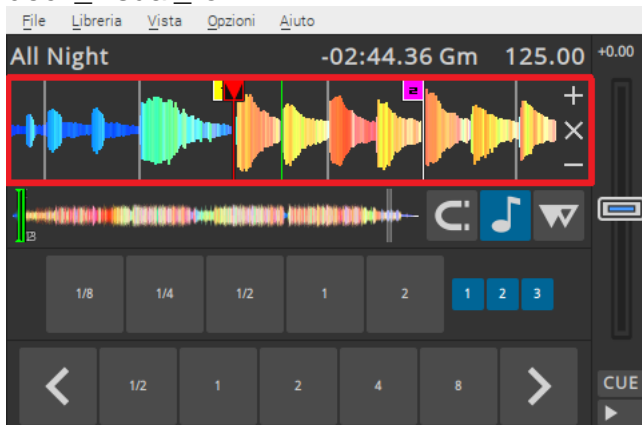
With this trick I have something like a scroll bar on the right of the display because the scroll bar on the screen is across the edge of the screen, so touching outside of the screen will cause a touch exactly in the scroll bar.

MainDecks: this file describe the main layout of the decks (in our project is just one: `deckLeft.xml`) section. More detailed description is in others file:

`deck_text_row.xml`: describe this part of the layout

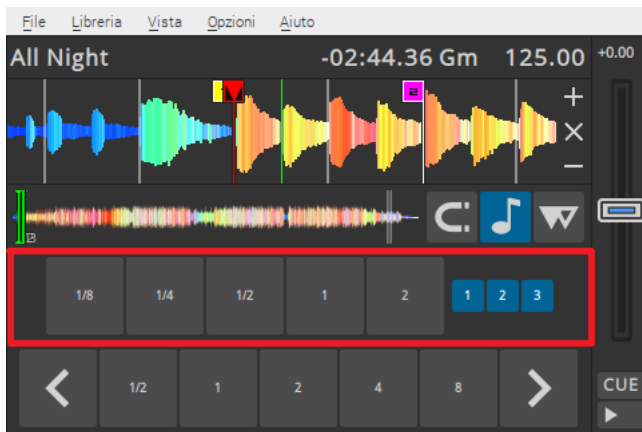


`deck_visual_row.xml`

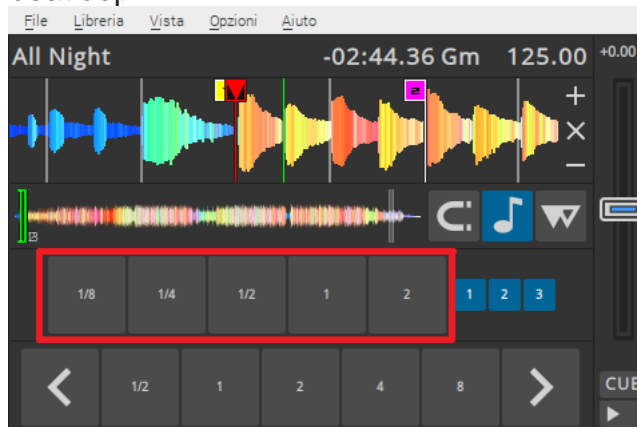


`Deck_control_row.xml`



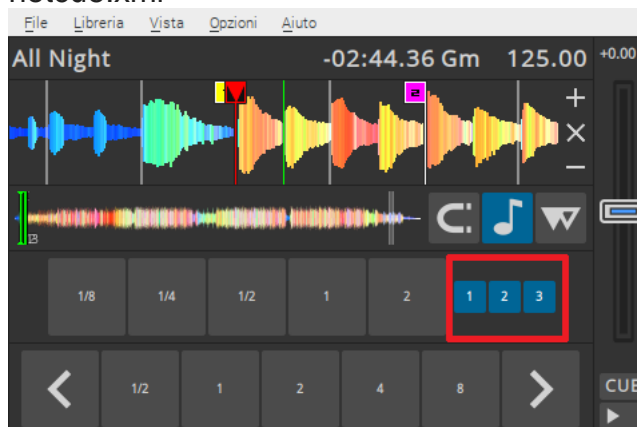


Inside deck\_control\_row.xml are called beatloop.xml



This beatloop.xml is different from the original because make the beatloop with the leftKey instead of the right. In this way when you will touch this loops in the touchscreen you will make a left mouse click wich will (translate "scaturisce") a loop roll.

hotcue.xml

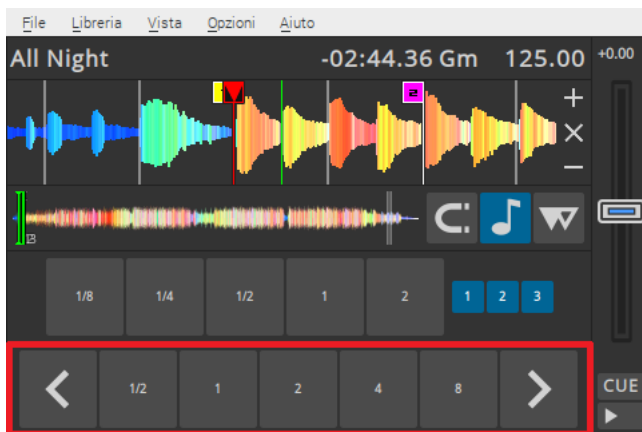


Is different from the original one because also here left and right function of the mouse are swapped.

In this way clicking on the touch screen on a hot cue will delete it instead of activating.

These are very little buttons because are just an indication if there is hotcues in memory (blue) or not (gray). You can call them or delete them with physical buttons and you don't have to use the touchscreen.

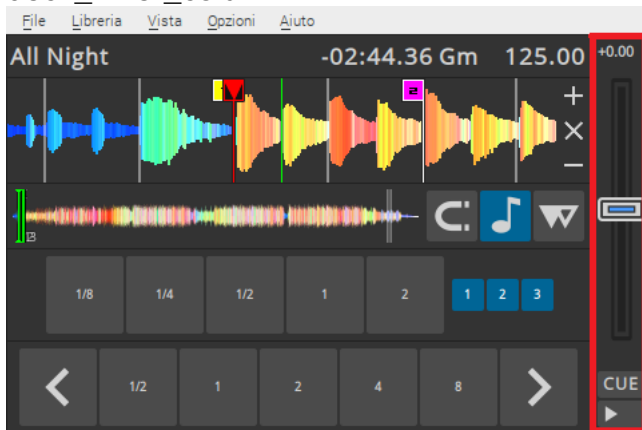
### deck\_control\_row2.xml



Inside deck\_control\_row2.xml is called beatloop2.xml that is the default beatloop.xml of the deere skin.

By left click you will activate a loop and adjusting length with left and right buttons.

### deck\_inner\_column.xml



About the library layout there is not to much to say but how the switching between the performance layout and the library one is handled?

The switching between the performance layout and library layout is reached by putting the library down the 320 pixels of the skin and when you press the button mapped for the full screen library you will have an overlapped library on the performance layout.