Universidad Zaragoza

Trabajo Fin de Grado

An early approach to hardwareless DJing

Autor

Pablo Martínez Blasco

Directores

Georg Umlauf

Dennis Grißer

Ponente

Javier Fabra Caro

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2019

Hochschule Konstanz
Fakultät Informatik
Institut für Optische Systeme

Eingereicht von
Pablo Martínez Blasco
Matrikelnummer 297853

pablomzblasco@gmail.com

Bachelor Thesis

An early approach to Hardwareless DJing

Konstanz, 31st July 2019

Bachelor Thesis

An early approach to Hardwareless DJing

for obtaining the academic degree

Bachelor of Science (B. Sc.)

at the

Hochschule Konstanz

Technik, Wirtschaft und Gestaltung

Fakultät Informatik

Studiengang Angewandte Informatik

Bachelor candidate: Pablo Martínez Blasco
Hindenburgstraße 5 WG 09
78467 Konstanz

1st Supervisor: Prof. Dr. Georg Umlauf
2nd Supervisor: Dennis Grißer, M.Sc.

Proposal date: 01.05.2019
Submission date: 31.07.2019

Abstract

Title:	An early approach to Hardwareless DJing
Author:	Pablo Martínez Blasco
Supervisor:	Hochschule für Technik, Wirtschaft und Gestaltung HTWG Konstanz, Institute for Optical Systems Prof. Dr. Georg Umlauf Dennis Grißer, M.Sc.
Submission date:	31.07.2019
Keywords:	Computer Vision, Arduino, MIDI

This bachelor thesis provides a first step towards Hardware-less DJing, simulating a controller by mapping it to image coordinates obtained from a camera feed. Coloured gloves are used to perform image analysis, as it is far easier to accomplish a good result with bright colours as opposed to skin, by using a HSV filter and morphological operations. Subsequently, contours and regions are later recognized for the fingers. As the real-time component of the system is important, gesture recognition is performed electrically by the use of an Arduino UNO board, therefore skipping the need to train a visual model to recognize certain gestures. A MIDI loopback and MIDI messages are used to connect the application to the DJing software. The main goal is to provide the same interactions a DJ is already used to, as to not require changing the muscle memory they have acquired by using normal controllers.

Ehrenwörtliche Erklärung

Hiermit erkläre ich *Pablo Martínez Blasco, geboren am 25.07.1997 in Zaragoza, Spain*, dass ich

(1) meine Bachelorarbeit mit dem Titel

An early approach to Hardwareless DJing

bei der Hochschule für Technik, Wirtschaft und Gestaltung HTWG Konstanz, Institute for Optical Systems unter Anleitung von Prof. Dr. Georg Umlauf selbständig und ohne fremde Hilfe angefertigt und keine anderen als die angeführten Hilfen benutzt habe;

(2) die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

(3) dass die eingereichten Abgabe-Exemplare in Papierform und im PDF-Format vollständig übereinstimmen.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 31.07.2019

(Unterschrift)

Statutory Declaration

I, *Pablo Martínez Blasco, born on the 25.07.1997 in Zaragoza, Spain*, herewith declare that

(1) I have composed the present thesis, with the title

An early approach to Hardwareless DJing

myself and without use of any other than the cited sources and aids. This document was written at the Hochschule für Technik, Wirtschaft und Gestaltung HTWG Konstanz, Institute for Optical Systems under the supervision of Prof. Dr. Georg Umlaufand Dennis Grißer, M.Sc.

(2) sentences or parts of sentences quoted literally are marked as such; other references, tables, images and programs with regard to the statement and scope are indicated by full details of the publications concerned.

(3) I declare that the submitted written (bound) copies of the present thesis and the version submitted as pdf are consistent with each other in contents.

I acknowledge that this declaration is true and correct, and I make it with the understanding and belief that a person who makes a false declaration is liable to the penalties of perjury.

Konstanz, 31.07.2019

(Unterschrift)

Contents

1

Introduction

1.1. Motivation

Electronic music has been uprising since 2010, with more and more music festivals appearing and becoming famous, a lot of people have picked an interest in making and mixing such genres. Let us draw a line differentiating both, music producing uses a DAW (Digital Audio Workstation), to synthesize sounds, and organize them in such a way to create a musical piece. On the contrary, mixing, or DJing, as it will be addressed in this document, focuses on the conjunction of different already created musical pieces; this is achieved through the use of a DJ software, which is historically connected to a hardware controller that maps the available action on the software to the buttons/sliders of the controller through MIDI.

This project is born from this controller necessity, as they are not always available for the people that want to start learning as a hobby. Therefore, it aims to provide a computer vision approach to DJing by using coloured gloves, that is cheap and does not rely in obtaining a hardware controller, thus creating a mapping of the software to the camera view and consequently adding individual customization of element position.

The use of gloves as MIDI controllers has already have some iterations, such as the MiMu Gloves [5], which focus on music production and are designed for a DAW mapping; or the Tornado Gloves A1 [15], which focus on mapping certain actions from the DJ software to gestures, which are then obviously limited.

On the other hand, this project attempts to provide a new approach, the idea of a 1:1 map of real space to the DJ software, thus allowing experienced controller users to use the same motions/actions they were previously doing in one but without the need of such hardware device, hence simulating the controller in real space and accessing it virtually by the use of the camera (see future work).

1.2. Objective

This project aims to provide a program that can map most features of a DJ software to the real space seen by the camera. Whenever a hand performs a “gesture” inside of the visual field of the camera, said gesture is to be identified and its position coordinates obtained. There are two gestures defined, one performed by touching the thumb and the index finger and the other performed by the index finger touching with the middle finger, each gesture enables certain aspects of the mapping to be enabled and the correct action is determined by the position of the fingers that perform the gesture. The position of the hands is obtained from the used of colored gloves

Since the DJing will be performed real-time, there is a high emphasis placed on the efficiency and computational complexity of the program. The program must process the camera feed without delay and identify a gesture as soon as it occurs. To achieve this, the program assumes gestures as a boolean state and performs the corresponding action through the whole gesture, thus giving the impression of continuous analysis when it is just processed at a human real-time speed (100ms). This allows for a significant reduction in the number of frames to be computed since millimetric precision is not needed.

2

Basics

This chapter provides the necessary fundamentals for the presented approach to hardware-less DJing. Among other things, DJ software/hardware, MIDI messages and Arduino essentials will be discussed.

2.1. DJ Tools

While there are multiple DJing programs (Virtual DJ, Serato, etc.), this project will be using Traktor Pro 2. Each software connects to the DJ controller via MIDI channel so the program could be virtually used with any software as long as the setup is done correctly.

For a DJ controller example we are going to use the Traktor Kontrol 2 as an example, since it has a similar layout to the Traktor Pro software and all types of inputs are clearly visible (Figure 2.1). In the depicted image we can observe four types of inputs: buttons, sliders, knobs and rotary encoders. The controller is divided in two equal sides, each one corresponding to a deck (A and B from left to right). The section in the middle is mirrored and shares access to both decks, it is called the mixer and controls the volumes of each deck as well as the global one.

Figure 2.1: Traktor Kontrol S2. Source: [16]

Now to compare with the utilized software (Figure 2.2). The software is divided in two areas vertically, the top one corresponding to the controller and the bottom to the library from where the music files are loaded. In the top section, we can observe (as in the controller) both decks, one on each side, with the mixer in the middle. The mixer usually maps 1:1 to the one in the controller, same with the deck jogwheel (rotary encoder), while the buttons map to different actions (usually samples/loops).

Figure 2.2: Traktor Pro 2

2.2. MIDI

MIDI messages are composed by a sequence of bytes (8 bits). The first byte is a status byte and the ones that follow it are data bytes. The status byte indicates the type of MIDI message sent in its first 4 bits, and determines the number of bytes that follow it depending on its type. The second 4 bits of the status byte indicate the channel in which the information is sent (out of 16 available).
MIDI running status references the ability of sending multiple same type elements one after the other while omitting the status byte, as it is assumed to be equal to the previous. This is useful when the message traffic is really high and not needed in this project. For more information regarding MIDI messages see [8] or [9].

2.2.1. MIDI Notes

This message type subdivides into Note ON and Note OFF messages. A note stays active in the MIDI channel until a Note OFF message is received. The structure for both of this messages is found next.

NOTE ON: **1001 CCCC** | **0PPP PPPP** | **0VVV VVVV**
NOTE OFF: **1000 CCCC** | **0PPP PPPP** | **0VVV VVVV**

- CCCC is the MIDI channel
- PPP PPPP is the pitch value
- VVV VVVV is the velocity value
- | represent the byte separation

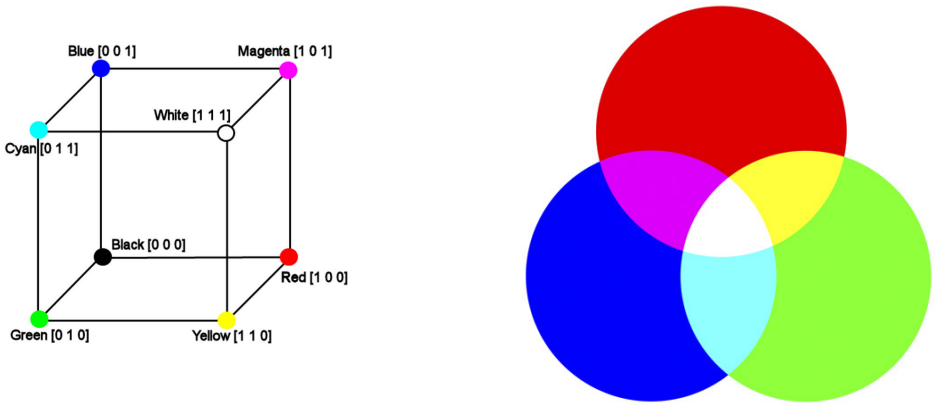
2.2.2. MIDI Controllers

MIDI controllers define a specific value (like a variable would) for a certain parameter. There are 128 MIDI controllers defined and the message is similar to that of notes, but they do not need to be set off as every new message updates the value. The structure for controller messages is: **1011 CCCC** | **0NNN NNNN** | **0VVV VVVV**. C and V represent the same values as previously, and NNN NNNN specifies the corresponding MIDI controller number.

2.3. Color Models

2.3.1. RGB Color Model

The RGB model combines the hue of colors red, green and blue. Each base color can have an intensity between 0 and a maximum. The maximum is the same for all three values, and is often 1.0 (when rendered as a floating-point number) or 255 (when a byte is used for each base color). If all three values are 0, a black color is obtained, when all three values are maximal, white is obtained. The RGB color model is thus additive, since colors are added starting from black until the desired color is reached.



(a) RGB cube

(b) Additive color mixing

Figure 2.3: RGB model as a cube representation and additive color mixing. Source [17] [11]

The color space spanned by the RGB model can be represented as a cube (see 2.3a) with the base colors at the corners. One corner of the dice represents the origin of a three-dimensional coordinate system in which each of the three primary colors sit at one of the corners, defining the axes. This origin of the coordinate system is black as it is an additive system.
Similar to the RGB is the CMYK color space, in which, starting from a white background, the three colors: cyan, magenta and yellow (CMY) are applied to obtain the desired hue. No color applied gives white, and black is obtained when all three colors are applied at their maximum value. This is a subtractive color model since it starts with white and addition gives black, and is mainly used in printers. Since all three colors are required for black, printers often have an extra tank for black ink. This variant is called CMYK. Since this is an opposite of RGB, the base colors for this space can be seen in each of the opposite corners of the bases for the RGB model.

2.3.2. HSV Color Model

HSV stands for Hue, Saturation and Value. It is an alternative to the RGB color model that attempts to more closely represent how humans perceive color, perceiving the hue apart from the brightness. If any of the individual values of a HSV tuple change, in the RGB color model. however, all values need to be adjusted to correctly represent it.

HSV provides an advantage in image editing. If a program has to mark all pixels of the same color to recognize an object, a slight change in illumination affects all three values of an RGB model, while in the HSV model, a simple change to the hue or saturation may suffice. The brightness (value) usually stays constant given the image and can usually be ignored. The HSV model also allows for the use of ranges in any of its three components, as a slight change in hue can be easily detected.

The HSV model can be represented as a cylinder (see 2.4). Value (brightness) takes the vertical axis, being black at the bottom and white in the center of the top. Saturation is increased the further we stray from the center of the cylinder. Hue is radial an starts at 0°, corresponding to the color red (also 360°), every 60°, each of the primary colors can be found.

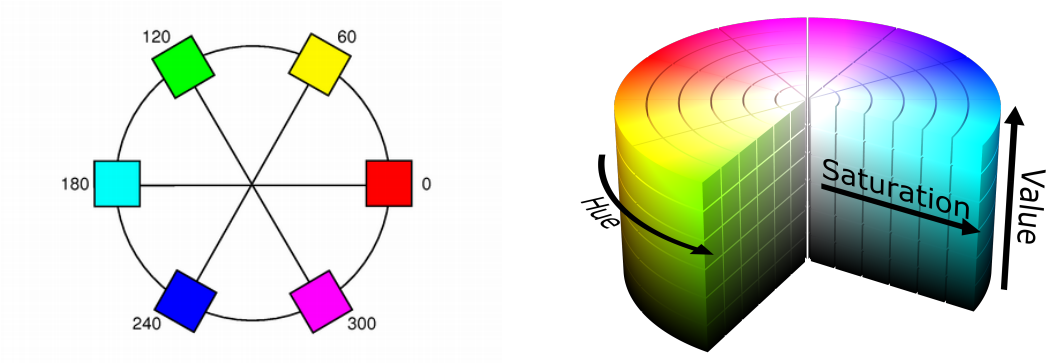


Figure 2.4: HSV model hue wheel and cylinder representation. Source [3] [4]

2.4. Morphological operations

All of the morphological operations are very well explained in [6]. This section is only the basic understanding of them as to show what each of them accomplishes and why it is important.

2.4.1. Erosion

Erosion is one of the basic operators in mathematical morphology. It erodes away the boundaries of regions of foreground pixels, thus shrinking these areas and increasing the size of holes in them. A structuring element (like a sphere, a square) is needed in order to perform erosion, the role of such element is to be superimposed to each pixel in the image in order to detect if all the components in the superimposed structuring element also are labeled as foreground, in which case the pixel does not change its value; if not all values are foreground, the pixel gets labeled as background.

Figure 2.5: Erosion example. Source: OpenCV [10]

2.4.2. Dilation

Dilation is the other basic operator together with erosion. The result of dilating an image is the enlargement of the region boundaries, therefore areas become bigger and holes in them get smaller. Dilation also makes use of a structuring element like erosion, and makes used of it in the inverse way, labeling a pixel as foreground when any of its “neighbors” (defined by the structuring element) is labeled as foreground as well.

Figure 2.6: Dilation example. Source: OpenCV [10]

2.4.3. Opening

Opening, as well as closing, is derived from erosion and dilation. Opening attempts to obtain an effect similar to erosion, removing the noise and some of the region border pixels, but aims to be less destructive. It is composed of an erosion followed by a dilation, both applied with the same structuring element. Like the previous operations, and by being derived from them, opening also depends on a structuring element, and tries to preserve foreground regions with a similar shape to the indicated element, or that fully contain it, while removing the rest. If noise is smaller than the specified element, it gets easily removed.
This effect can be comfortably compared with computer drawing, imagining the structuring element as the brush shape, and trying to recreate a given image with it.

Figure 2.7: Opening example. Source: OpenCV [10]

2.4.4. Closing

Closing is the dual operator of opening, and also consists of erosion and dilation, but applied in inverse order, where the dilation is employed first, followed by an erosion, both using the same structuring element as specified with opening. The effect of this operator is to preserve background regions that contain or have a similar shape to that of the structuring element, so that it can be applied to any background point and made to cover the pixel without taking contact with a foreground region. Consecutive closings with the same structuring element will have no effect on the image. The closing operator fills instantly any holes smaller than the structuring element, and makes those bigger than it smaller.

Figure 2.8: Closing example. Source: OpenCV [10]

3

Implementation

3.1. loopMIDI

loopMIDI [7] is a software developed by Tobias Erichsen. It creates a virtual MIDI loopback which is used to interconnect the application with the DJing software by opening hardware-MIDI ports, using them to send the corresponding MIDI messages.

3.2. RtMidi

RtMidi is a library that provides an API for realtime MIDI input/output ([13]). The version used in this project is 3.0.0. This library is used to send MIDI messages from the application to the virtual MIDI loopback, which in itself connects to the DJing software.

3.3. Employed Gloves

Cotton gloves from Hama and green silk paint from Marabu were used (see 3.1), as they had been already been used in another project. It is recommended that the wrist part of the gloves be turned in and, if possible, fastened, as the gloves cover only the hand and not even parts of the arms. Pink latex gloves were used on top to cover the majority of the green glove but the desired fingers, and therefore were cut to achieve

so.

(a) Gloves

(b) Paint colors.

(c) Painted gloves.

(d) Latex overlay gloves.

(e) Finished glove result.

Figure 3.1: Glove elements until finished version (e).

3.4. OpenCV

All of the image recognition in this project is done by OpenCV, specifically version 4.0.0. OpenCV stands for “Open Source Computer Vision” and is a open source computer vision and machine learning software library. It has C++, Python, Java and MATLAB interfaces, but I will be using the C++ one as this project is developed in C++. The library focuses on a optimized implementation of classic and state-of-the-art computer vision algorithms. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception. Possible applications for OpenCV are, for example, in automated inspection in industry, in medical imaging, in the evaluation of surveillance images, in human computer interfaces, in stereo vision or in robotics [2].

3.5. Binary image using a HSV filter

A binary image is a black and white image where the recognized pixels take the white color, and the background gets converted to black. Binary images can be stored as bitmaps, which take significantly less space in memory than fully colored images. Operations are significantly faster too as they can be interpreted as logical values instead of integers and only have one color dimension versus the three that a color image has.

In order to obtain a binary image, we need to convert the RGB camera feed into a HSV model (see 2.3), this is achieved using the OpenCV function cvtColor() with the corresponding “COLOR_BGR2HSV” parameter. In order to apply the selected HSV range settings (later explained in 3.11), the function inRange() is used on the HSV image matrix in order to threshold it. This function separates each of the HSV channels, and then applies the corresponding threshold depending on the range, utilizing the following function for each of them, where lowerb and upperb are the minimum and maximum values of the corresponding range respectively, src being the image matrix:

$$dst(I) = lowerb(I)_{channel} \leq src(I)_{channel} \leq upperb(I)_{channel}$$

The default range of the HSV values are provided to use lime green gloves, when using the application, the hue will rarely need to be modified as what changes mainly are the environment lighting conditions, which can be adjusting the saturation and value, usually the saturation needs to be lowered to identify darker shades of green in the shadows, and the value can be adjusted to remove some noise.

3.6. Morphological operations in a binary image (post-processing)

The binary image is further processed in order to make the later contour identification more reliable. For this, morphological operations are used (see 2.4) in order to reduce the amount of noise in the image.

This also infers a faster contour detection, as there are less regions to detect and thus to analyze. Therefore, an “opening” is performed (see 2.4.3), which is conformed of a erosion followed by a dilation, preserves foreground regions while removing background ones. After the opening a “closing” (see 2.4.4) is performed, which purpose is to close inner gaps in the finger regions.

For the following images the same HSV ranges and the same environment will be used.

(a) After HSV-filter

(b) Erosion

(c) Dilation

Figure 3.2: Morphological transformations of a binary image.

3.7. Contour detection

Contour detection is performed using OpenCV library. This function retrieves contours from an image utilizing the algorithm in [1]. As the contour detection is performed

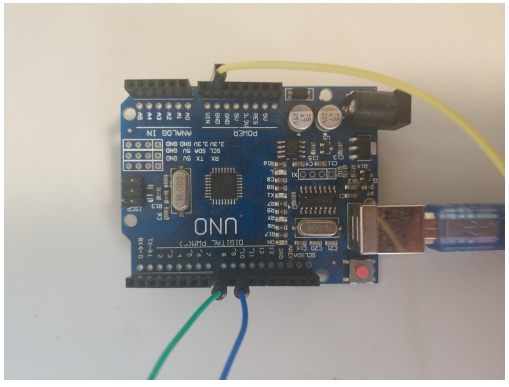
on the binary image, the result is a vector of points of the boundary of every region existing in the binary image. This contours can be later reduced by analyzing their area and thresholding it, providing a way to remove the noise that was still left over from executing both opening and closing operator functions 3.6.

3.8. Arduino

An Arduino Uno board is used to perform gesture recognition via hardware instead of needing to do image analysis for it. The library SerialPort [14] is used to utilize the serial connection to the Arduino board, which allows the reading of the data stream sent by the Arduino.
Since this is a real-time system, it needs to compute the desired action as fast as possible; thus, by acquiring the gesture signal from a electric signal, it only needs to reach the application via the serial connection. Both considered gestures work really well with this idea, since they need physical contact between two from three different fingers (thumb and index, or index and middle fingers).

For the connection between the fingers (gesture detection), a cable is connected to each finger, contacting a aluminum ring which shares the current from one finger to another. Both thumb and middle finger have input pins connected to them, and the index finger has 0V (ground) of current connected so that when it touches one of the other fingers, 0V are registered in the corresponding input thus activating the gesture (see 3.8.1).

(a) Cardboard rings with aluminum



(b) Arduino pin configuration

(c) Connection of the Arduino pins to the rings in the hand.

Figure 3.3: Setup of the Arduino connection for detecting gestures.

3.8.1. Floating pin

Since the index finger is shared between both gestures, we want the other two fingers to carry the input signal (to the Arduino). Because of this, we run into the floating pin problem, where we do not know what an open switch will return as value (LOW or HIGH) as the value depends on the conductivity of the environment and is therefore variable.
To solve this issue, either a pull-up resistor or a pull-down resistor is needed, which have a value of HIGH and LOW respectively when the switch is open. We would preferably use a pull-down resistor, like digital circuits that require a positive one-shot trigger when a switch is momentarily closed to cause a state change. However, Arduino boards only possess a pull-up resistor, so the switch performance is inverted (takes a value of LOW when active). Thus, the current passed to the index finger to give a value of LOW is 0, or ground, so that it activates the value when it enters in contact to one of the fingers that carry an input to the Arduino.
Of the available pins in the Arduino board, it is highly not recommended to use pin 13 as a pull-up pin, as it contains the LED and its own resistor, which make the current have a value of 1.7V instead of 5V since the mentioned elements pull the voltage level down.

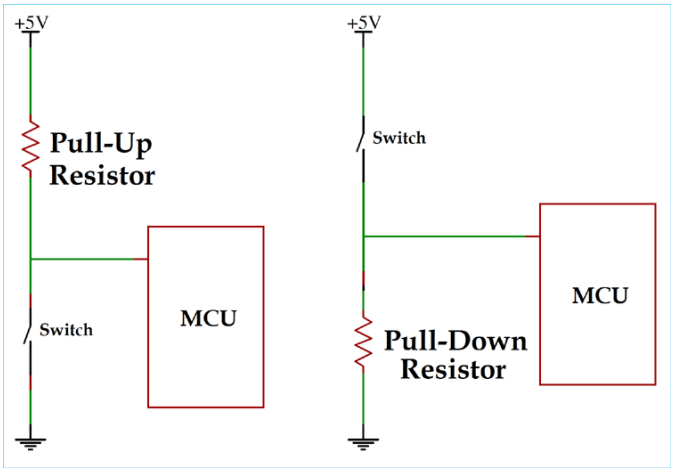


Figure 3.4: Pull-up and pull-down resistors. Source: [12]

3.9. Data Structure

As emphasized previously, the real-time component of the system is of utmost importance. Consequently, we want to obtain the corresponding MIDI function given a gesture location with the least computational cost. For this purpose, a region Quadtree is used to create a mapping of image coordinates to the analogous MIDI function to be called when certain gesture is received from them. Since there are two possible gestures, there needs to be two different configurations of the Quadtree. Two options were then considered, making two different trees, one for each gesture, or combining them into one; the latter was implemented, making the difference of Fader and Button gestures. Each node of the Quadtree contains a pointer to each fader and button function to be executed in that coordinate section, as well as the four children nodes in case it is further subdivided due to a function conflict.

A sketch of what the mapping is considered to look like in a real hardware controller which is used as reference for this project can be observed next (see 3.5). The spheres are dashed since they overlap faders but still exist in the mapping as they use different gestures as previously mentioned. An example of an early Quadtree over this sketch can also be found (3.6), showing the level of subdivision inside the tree until a unique function is reached in the available rectangle.

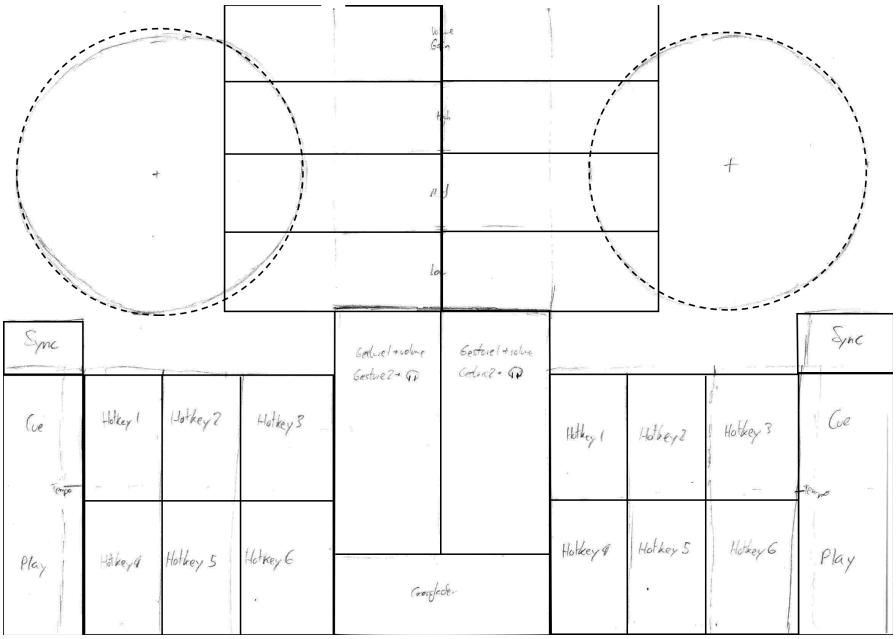


Figure 3.5: Sketch of what the software element distribution looks like.

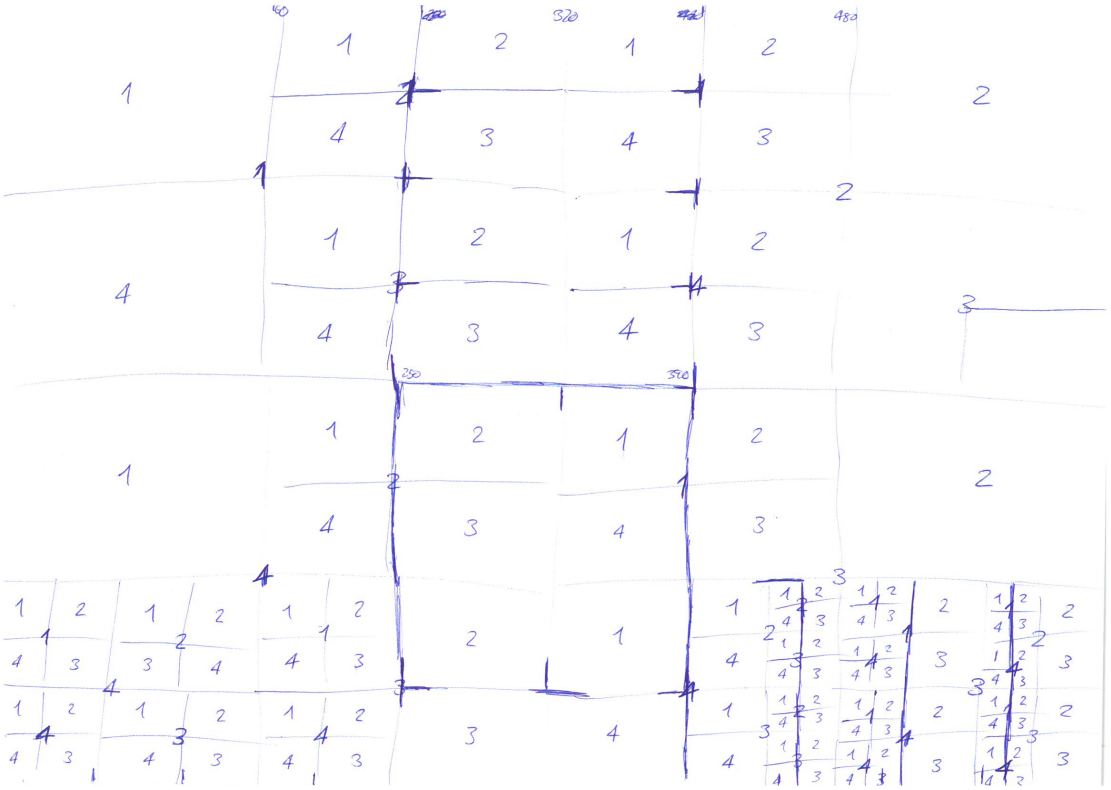


Figure 3.6: Early iteration of the Quadtree subdivision.

3.10. MIDI functions

Tables with the corresponding MIDI channels and controllers for each individual action can be found next, divided by the gesture that enables it. The value of the message goes from 0-127 for the faders indicating position, and 127 or 0 for buttons. For each individual action, there is a C++ function stored in the tree that always takes the deck as parameter, and the value in case it is a fader. Each of this functions can be called directly from the tree node with the correct parameters.

Fader Channels

Function	Deck A		Deck B	
	Channel	Controller	Channel	Controller
Volume	176	0	177	0
Highs	176	1	177	1
Mids	176	2	177	2
Lows	176	3	177	3
Tempo	176	4	177	4
Gain	176	5	177	5

	Global faders	
	Channel	Controller
Crossfader	178	0
Global Volume	178	1

Table 3.1: Fader MIDI channels

Button Channels

Function	Deck A		Deck B	
	Channel	Controller	Channel	Controller
Play/Pause	144	0	145	0
CUE	144	1	145	1
CUE1	144	2	145	2
CUE2	144	3	145	3
CUE3	144	4	145	4
CUE4	144	5	145	5
Headphone bus	144	5	145	5

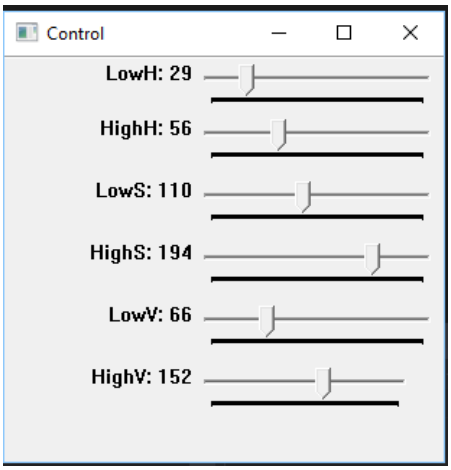
Table 3.2: Button MIDI channels

3.11. Application execution

Before running the application, the MIDI loopback has to be initialized using loopMIDI [7]. When running the application, first two camera feeds appear, one being just the recorded camera feed and the other a binary image. This is shown so the user can configure the HSV values each run, since the environment may be different in each execution and this values may need to be adjusted.

(a) Normal camera feed.

(b) Binary segmentation with current HSV values.



(c) HSV value configuration (minimum and maximum for each).

Figure 3.7: Initial camera configuration with HSV values.

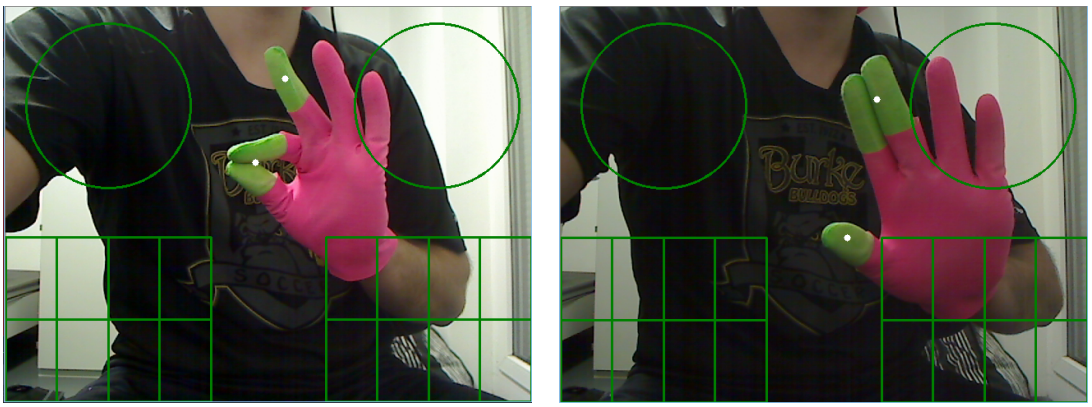
After the setup is done correctly, all the necessary connections are established. MIDI connection is set via the rtMidi package and all of the MIDI note channels are cleared, as they can have a leftover note from previous executions that was not deactivated, this port cleaning is achieved by sending a “NOTE OFF” message to each of the note channels (see 2.2). The Arduino serial port connection is also initialized, both of these connections are established by pointers.

After the connection initialization, three threads are created, one for an image loopback, one for the Arduino inputs, and one gesture loop for the calls to the corresponding functions given a position and gesture.

3.11.1. Image thread

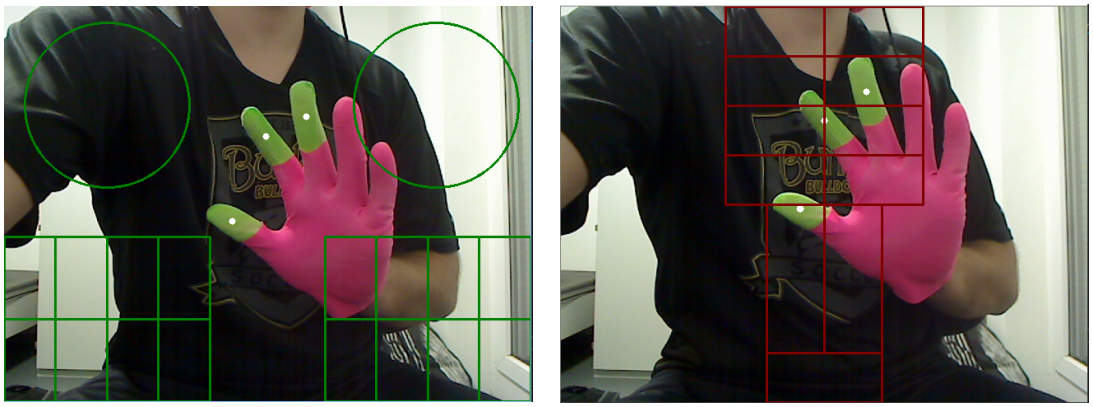
This thread executes an infinite loop that analyzes each image of the camera stream, and converts it to HSV (see 3.5) to perform a opening operation followed by a closing one (see 3.6). Then it extracts the corresponding contours and stores the position of the average coordinates for each region bigger than a certain area threshold (corresponding to the area needed to recognize a thumb, since it is the smallest finger that need to be tracked).
When two fingers touch (gesture is activated, even if captured hardware-wise with the Arduino), the region of both touching fingers combine, and its average position is really close to the point of actual contact (refer to 3.8); that little displacement can be overlooked in this application since millimetric precision is not needed, as the regions are comfortably big as to provide better usability.

This thread also creates an image loopback, in order to allow the user to see the position of his own hands corresponding to the mapping. The stream image is cloned and the position of the corresponding function areas for each gesture are shown, creating two image loopbacks, each superimposed with a mapping of the regions that contain actions in the Quadtree, each loopback displaying only the regions for one of the two gestures as their mappings are different (see 3.9).



(a) Fader gesture average position (b) Button gesture average position

Figure 3.8: Average region points for both available gestures.



(a) Mapping of button gesture.

(b) Mapping of fader gesture.

Figure 3.9: Image loopback with gesture mappings.

3.11.2. Arduino thread

The Arduino thread is a basic infinite loop that analyzes the messages sent by the Arduino board via the Serial Port. It compares the value received, and if it is feasible, updates the value of a global variable (called state) with the received input (see 3.10). An input of 0 means no gestures were activated, an input of 1 means the fader gesture was identified, and a value of 2 indicates that the button gesture was performed. The code for the Arduino board simply sets up two pins as input pins, respectively configured as INPUT_PULLUP because of the floating pin issue (see 3.8.1). When either of these fingers connect with the index finger, an input is received and the corresponding gesture code is sent via the Serial Port (see 3.8 for cable/connection setup).
The global variable “state” never reaches race conditions (it can be accessed by the gesture thread when its value is being changed, being its value not reliable) since the correct value will be read upon next iteration and the delay between iterations is unnoticeable.

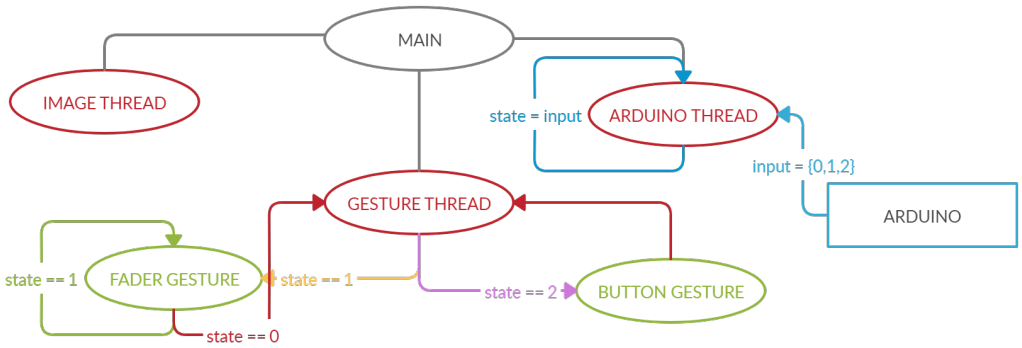


Figure 3.10: Automata of the state variable and its connection between threads.

3.11.3. Gesture thread

The gesture thread constructs the Quadtree before entering a infinite loop where it checks the “state” global variable every 100ms, this delay is dependent on what real-time is considered, in this project, real-time takes a value of 100ms as the user cannot perceive any changes that occur in that interval or a shorter one. This is best observed when calling a fader function, which moves over time until the user “releases” the fader gesture, since it updates in a discrete manner as opposed to a continuous one, but it is imperceptible and thus indicates that the delay is appropriate. This delay is introduced in this loop since it has to perform a search in the tree every time a gesture is recognized, and then execute the corresponding function linked to it, which results in a MIDI message being sent to the software application. Since we want to reduce computation costs, and we do not want to send that MIDI messages since the user cannot observe a difference between the real-time delay and no delay, the delay is applied to reduce the number of tree searches as the quantity of MIDI messages.
When calling a tree function, the coordinates are obtained from a global variable called actual, which is updated by the image thread with the average coordinates of the biggest identified region. The position is always correct when a gesture is active, since the area of two fingers touching is always bigger as that of the third finger.

4

Discussion and Outlook

4.1. Future work

The presented application does not provide a mapping for jogwheels or loading tracks to the decks. The problem with these actions is that in a normal hardware controller, they are implemented with the use of rotary encoders. The jogwheel encoder usually takes a big area of the controller, so it can be implemented easier than the one for loading tracks, as the rotation occurs with the fingers as the center.
For the jogwheel, a logic through time has to be implemented, or do the sequencing at a adequate delay, so that it is impossible to have done a complete turn between two gesture analysis. An alternative of a rotary encoder is to use a simple slider, arising the issue of how to deal with looping and gesture start/stop signals. Also the MIDI messages that need to be send in order to correctly configure a jogwheel have not been looked into in this project.
For the implementation of track loading, either a drag and drop using the fader gesture can be used, but then a new gesture must be added in order to indicate the user is hovering over the tracks and selecting one; or a up/down/select button system can be implemented, since there is plenty of space left in both gestures mappings.

Also an interesting step would be to implement a UI to make mapping editing simpler and more intuitive, since the Quadtree is generated at runtime, the user could configure his desired distribution of elements for each of the gestures, allowing customization and granting the user a more intuitive mapping as not all controllers have the same distribution.

4.1.1. Data gloves

This thesis aims to provide a starting point on the possibilities of DJing without the need a specific DJ controller. The next step would be the removal of the camera and image analysis section, as the places where DJs perform are usually dark and have very inconsistent lighting due to artificial lights that are used in clubs/festivals.
Data gloves would completely remove the need for a camera, since all the gestures and hand position could be obtained from the data gloves themselves, as well as allowing the possibility to make use of the z-axis, as the mapping in this thesis is only two-dimensional. Along this z-axis different mappings could be stored in distinct levels, for example having the lower level with the deck mappings as in this project, and in an upper level have effect configuration. Correspondingly, another type of controller can be mapped at a different height, allowing for the use of multiple controllers, each covering different needs, on the same platform (i.e. Traktor S2 as lower level (analog to this project), and a Native Instruments MASCHINE in an upper level for live performances with samples).

References

[1] Satoshi Suzuki et al. “Topological structural analysis of digitized binary images by border following”. In: *Computer Vision, Graphics, and Image Processing 30(1):32–46* (1985).

[2] Gary Bradski und Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O’Reilly, 2008.

[3] Wikimedia Commons. *HSV Cylinder*. https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png.

[4] Wikimedia Commons. *HSV wheel*. https://doc.qt.io/qt-5/images/qcolor-hue.png.

[5] Imogen Heap. *MiMu Gloves*. https://mimugloves.com/.

[6] *HIPR2*. https://homepages.inf.ed.ac.uk/rbf/HIPR2/.

[7] *loopMIDI*. http://www.tobias-erichsen.de/software/loopmidi.html.

[8] *MIDI Tutorial*. http://www.music-software-development.com/midi-tutorial.html.

[9] *MIDI Wikipedia*. https://en.wikipedia.org/wiki/MIDI.

[10] *OpenCV*. https://docs.opencv.org/.

[11] Pantone. *RGB Wheel*. https://www.lcipaper.com/kb/what-are-the-differences-between-pantone-cmyk-rgb.html.

[12] *Pull-up and pull-down resistors*. https://circuitdigest.com/tutorial/pull-up-and-pull-down-resistor.

[13] *RtMidi*. https://github.com/thestk/rtmidi.

[14] *SerialPort*. https://blog.manash.me/serial-communication-with-an-arduino-using-c-on-windows-d08710186498.

[15] *Tornado Gloves A1*. http://en.global-dj.com/.

[16] *Traktor Kontol S2*. https://www.musicmatter.co.uk/native-instruments-traktor-kontrol-s2-mk3.

[17] Stephen Westland. *RGB Cube*. https://www.researchgate.net/figure/The-RGB-colour-cube_fig1_304240592.