CSC_51073_EP: Final project report
Colorhythm - Turning Vision into Music

Benoît SAVARY Thomas POUPONNEAU

Abstract

Passionate about music and computer vision, we wanted to create a pipeline from scratch, using pre-trained models to generate notes and mixing tracks in an Ableton Live music project based on our movements and actions captured by two cameras. Thanks to shape recognition of hands, head and pieces on a physical board, computer vision can help making music. This project aims to generate live music from video input.

1. Project & Motivation

When we were looking for a project, we tried to gather computer vision techniques and music. Since music involves more signal processing rather than computer vision, we found it particularly interesting to combine music and video to create a new way of making art. We decided to develop a ”music conductor” tool and a ”beat-maker” tool: being able to create music and modify it, for instance changing the volumes of different tracks in the same piece of music. Hardware already exists to do so but it can quickly be very expensive. With our solution, only a computer with a webcam and a phone with a camera are mandatory! As soon as the computer has some computation power, *Colorhythm* provides a software solution, which is also really funny to use (conducting with your hands as if you were a classical music conductor!).

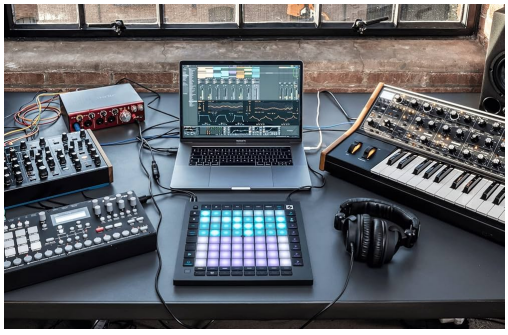

Figure 1. Hardware often used by beat-makers

2. Base works and tools

Our work was inspired by a computer vision and music project found on youtube [1] and consists in generating a music loop from multiple boards. Each board serves a distinct purpose: one specifies the octave, another the beat and instrument choice, and the last one is a slider (to include a sliding sound in the loop). The board actions results in a loop which is updated in real-time.

We wanted to keep this board idea while including new computer vision features to use to control other elements in our example sound level!

For the input videos, we use both a webcam, and a smartphone used to film our board. The processing is done with Python, mainly leveraging two libraries for the two data types we will be working with: OpenCV for images and Mido for music management (used to communicate with a music software -Ableton- through a MIDI port)

3. Pipeline and implementation

3.1. What we decided to focus on

The subject has an infinite potential in terms of creation: we could create as many board as we want, or simulate MIDI signals to generate any music instrument sound! So we needed to limit our study and use computer vision in priority.

We decided to keep it simple: one board to create drum loops (with a kick, a snare, a hi-hat and a bonus sound) and four sliders to control four track volumes of an existing Ableton Live project. So in this Ableton Live project, there will be three pre-recorded tracks, and the fourth track will be live-recorded.

[1] ”The Machine Vision Music Performance”
https://www.youtube.com/watch?v=rMhrd-2--Us

3.2. Pipeline - developper's point of view

1. Initialization: Check dependencies and ensure all required libraries are installed. Verify the availability of the webcam and MIDI port.
2. Video Capture: Initialize video captures for the webcam and board video. Set up the MIDI output for sending signals.
3. Detection and Processing: Detect hand movements and circle positions in the video frames. Process the detected inputs to generate MIDI signals.
4. MIDI Signal Sending: Send MIDI control change messages based on hand movements. Send MIDI notes based on the positions of detected circles.
5. User Interaction: Provide commands to send MIDI sequences or quit the program. Display the processed video frames and MIDI control status.

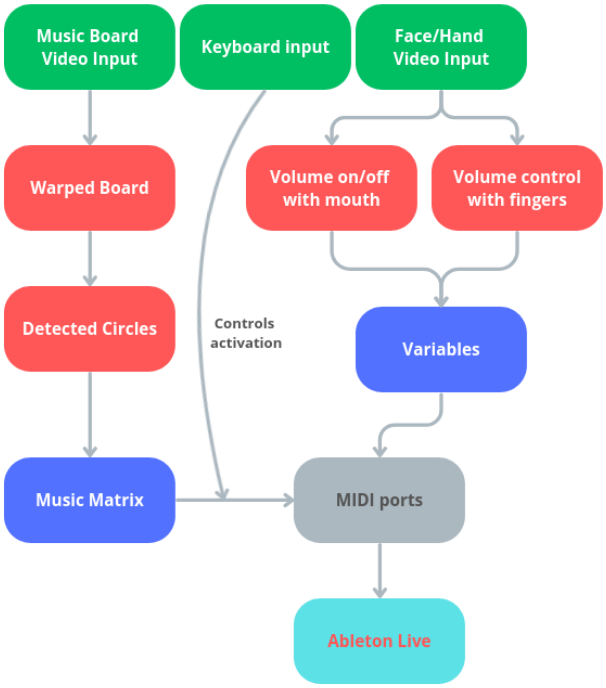

Figure 2. Pipeline of Colorhythm

3.3. Pipeline - user's point of view

The pipeline can be described as follows:

- The user films his face and hands with a webcam
- The user films a *board* with a smartphone
- The data collected by both cameras is processed
- The data is converted into MIDI signals
- The signals are sent to Ableton Live (music production software) to add a new music track with specific notes, and to modify parameters of the current music such as volume or tempo
- On the screen of the user, there are the computer vision calculations and several windows displaying the live modifications.

More specifically about the *board*:

- It is a grid of chosen dimensions
- The horizontal dimension gives the duration of the loop that will be repeated until the user decides to stop it
- The vertical dimension gives the range of different notes available to the user to create its music

This idea of *board* is endless: only your imagination limits the infinite combinations of x and y sizes, choice of instruments and notes, etc. For this project, we decided to limit ourselves to a drum rack of 4 times 8: a loop has a duration of 8 beats and we can choose among 4 different drum sounds.

3.4. About MIDI signals

MIDI (Musical Instrument Digital Interface) is a protocol that enables electronic musical instruments, computers, and other devices to communicate and synchronize. In our project, MIDI signals act as a bridge between the computer vision input and the music software (Ableton Live).

More specifically, one MIDI signal is a 3-feature vector:

- an ID, generally transformed into a pitch by softwares: *60 for C4, 62 for D4, ...*
- a velocity, which is a number between 0 and 100 representing the intensity of the signal: *10 for a very soft note, 100 for a loud note*
- a duration, generally in milliseconds: *1000 for a signal lasting 1 second*

Using the MIDI protocol allows the system to manage real-time communication with Ableton Live, making it possible to generate music and modify it dynamically based on user actions.

3.5. How MIDI is used in Colorhythm

We decided to encode our MIDI signals in 4-feature vectors. The fourth feature is the starting time of the note (with the first note of the track being at t=0). The unite of this feature is the beat (that is, a *click* of metronome).

So a sequence of notes (like chords or arpeggios) is simply a matrix of such 4-feature vectors. Dedicated functions were coded to read these matrices and send the MIDI signals with the 3 features (pitch, velocity, duration) at the right moment (fourth feature).

Here is an example of matrix containing two successive chords (a chord is composed of three notes played at the same time):

```
def create_example_matrix():
    M = [
        [60, 100, 0, 2],   # C4 - root of C major chord
        [64, 100, 0, 2],   # E4 - third of C major chord
        [67, 100, 0, 2],   # G4 - fifth of C major chord
        [65, 100, 2, 2],   # F4 - root of F major chord
        [69, 100, 2, 2],   # A4 - third of F major chord
        [72, 100, 2, 2]    # C5 - fifth of F major chord
    ]
    return M
```

- Control Change Messages: The hand movements detected by the webcam are mapped to specific MIDI Control Change (CC) messages. These CC messages adjust music parameters. Here, the distance between the thumb and the index allows a control of volume.
- Note Messages: Colored circles detected on the board are translated into MIDI Note On/Off messages. These determine the pitch (note) and velocity (intensity) of a sound. Each circle's position corresponds to a specific note and position in the loop.

3.6. Code structure

The system captures hand movements and colored circles from a video feed, processes these inputs, and sends corresponding MIDI signals to control music software like Ableton Live.

The project consists of four Python files:

- main.py
- camera_detection.py
- midi_signals.py
- midi_initializer.py

The main.py file serves as the entry point of the application. It handles the initialization, dependency checks, camera and virtual MIDI ports checks, and orchestrates the main loop of the application. Finally the file calls camera_main from camera_detection.py to start the detection and processing.

The camera_detection.py file contains the core logic for detecting hand movements and colored circles from the video feed. It uses MediaPipe for hand and face detection (webcam) and OpenCV for image processing (board camera). It explicitly detects the main board, identifies the circles on it (along with their dominant color), and applies a perspective transformation to straighten the detected board.

The midi_signals.py file handles the sending of MIDI signals based on the detected hand movements and circle positions in camera_detection.py:

- The hands and face detected by the webcam are translated into MIDI control change messages.
- The circles detected on the board become an encoded matrix that becomes a sequence of MIDI notes with specific pitch, velocity, date and duration.

The midi_initializer.py file provides a utility to initialize and map MIDI controls in Ableton Live. In other words, it is a homemade tutorial to link a button on the user's software Ableton Live and a MIDI identifier (ie a number chosen in the code that identifies the function of the associated signal).

4. Implementation challenges

4.1. Shape & color detection of rectangles and circles

First, in order to create a good scene to detect circles, we had to do a perspective correction of the board, by detecting the rectangle and warping its inside to a new image. It raised instability and inconsistency issues caused my slight movements, lightning and hands passing through. In order to tackle this, we used blurring/dilation on the image during the process to lessen the influence of movement/lightning.

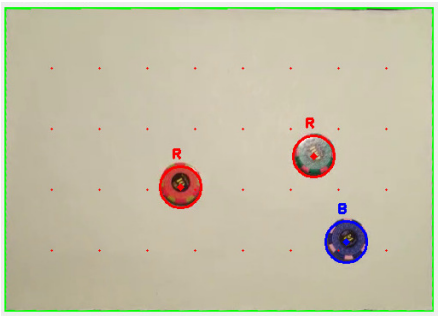

Figure 3. Color detection error due to light

4.2. Time precision of live MIDI signals

When we decide to send a MIDI signal, some Python functions are called, computations are performed, ... these steps take some time, and even if this time is quite short, our ears can hear that the generated notes are not exactly on time. Send it can be easier to generate a MIDI file rather than a succession of MIDI signals, because the MIDI file has exact timings and can be imported into the Ableton Live project. This is not strictly live-recording, but it is the easiest way to win time precision.

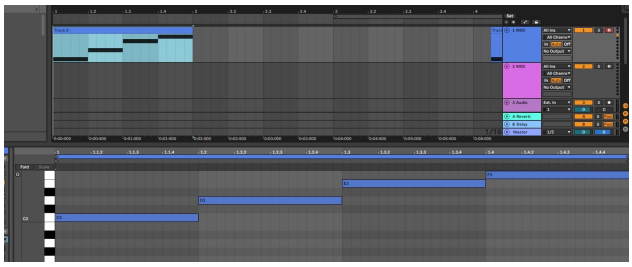

Figure 4. Precise timings with an imported MIDI file

4.3. Computing performance

The computer receives at the same time video inputs from two cameras, creates multiple MIDI signals per second and processes them in Ableton Live. This can quickly lead to latency. Indeed, we faced computation issues at the beginning because we started calculations on every frame of both cameras, which was really not optimal because we wanted to use these calculations only at key moments for music making (at the end of a loop for instance). Then we had to think about how to minimize the calculations, so that the videos could display without latency and hand/board gestures could have an (almost) immediate sound impact via generated MIDI signals.

4.4. Volume sliders

We used functions to scale and smooth the sliders. For instance, without scaling, the length between two fingers could not be strictly equal to zero. With the scaling, if the length is smaller than, let's say, 0.05, then the value of the variable is set to zero and in case of volume control of a track with this variable, a MIDI signal could mute a track completely.

5. Results

After execution of main.py file, if all requirements are valid, a window appears and displays both cameras, the warped image of the board and some sliders indicating some relevant lengths from the webcam.

On the figure below, we show the 4 windows displayed for the user:

- The webcam input with face and hands meshes from Google MediaPipe, with a boolean value "Live Modifier" (toggled by opening the mouth) and specific lengths between fingers : the length between thumbs (red line), between indexes (blue line) and between each thumb and index of both hands (green lines),
- The four volume sliders, containing a scaled value based on the corresponding colored line (first window),
- The raw board video input,
- The warped version of the video with circles and colors detections.

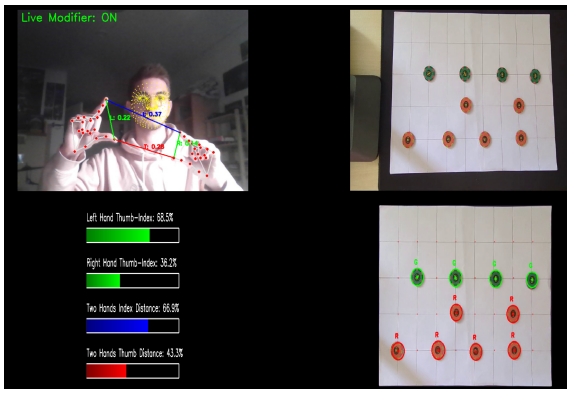

Figure 5. User's windows

6. What could be done next

Ideas for the future of this project are infinite! Here are some of our favorite ideas:

- Leverage machine learning to recognize a broader variety of gestures, such as snapping fingers to trigger loops or waving hands to clear tracks.
- Explore hardware acceleration options using GPUs or specialized devices to enhance computational speed and efficiency.
- Implement generative visuals that synchronize with the music, creating an immersive performance environment.