DIPARTIMENTO
DI INFORMATICA
SAPIENZA
UNIVERSITÀ DI ROMA

Echo: Hand-Tracking Driven Audio Effects Manipulation

Claudia Di Carlo 2107996
Zofia Elzbieta Dobkowska 2106489

***Abstract*—Music production tools are often complex and inaccessible to beginners, limiting experimentation and creativity. This project presents a beginner-friendly system that integrates real-time audio manipulation with visual feedback enabled by computer vision. Using Mediapipe and OpenCV for hand tracking, the program maps hand gestures and rotations to control audio effects such as gain, distortion, compression, and chorus in real time. Audio is processed with Spotify's Pedalboard library, ensuring low-latency performance through continuous streaming, while Tkinter provides an intuitive GUI displaying effect parameters and track information. Right-hand finger combinations select and modulate effects, while left-hand gestures control track selection. This approach demonstrates how gesture-based interaction can simplify music production, making it more accessible and engaging for novices while opening avenues for creative performance.**

Index Terms Hand tracking, Mediapipe, Pedalboard, Real-time Audio Processing

I. Introduction

Music production software can be inaccessible and complicated for beginners, making it difficult to try different things and let creativity flow. Not only is it difficult to understand the purpose of each effect, but also to learn how to use each different program.

The aim of the project is to create a beginner-friendly, accessible and engaging way to approach music production, by integrating real-time audio manipulation with visual feedback enabled by Computer Vision. The program tracks the user's hand movements and uses their rotation to change the audio effects parameters,letting the user both hear and visualize the changes made.

Current research on computer vision-based audio processing is mainly focused on audio classification, such as recognizing the characteristics of a soundtrack[1] or labeling environmental sounds[2]. However, there are some projects that focus on modifying audio effects with hand tracking but are made using the TouchDesigner software or are written in programming languages different from Python[3].

Our project is divided into two main sections, an audio one and a visual one, included in a single *main* file. A second *gui* file contains the class defining the Graphical User Interface. Hand tracking, choice and modification of parameters are achieved mainly through the use of the Mediapipe and OpenCV libraries, while audio processing is done by Spotify's Pedalboard library. For the GUI we have used the Tkinter library.

II. Methods

A. Audio

Audio is handled by a daemon thread which runs continuously in the background. Inside the output stream's audio callback, sound is processed in chunks: it is necessary to conceptualize audio as a stream, since working on a complete file at once would be inefficient and slow. A chunk is taken so that, if its size is larger than the length of the remaining audio file, it loops back to the beginning. The size of the chunk (*blocksize*) determines the latency of the sound, which must be almost imperceptible in real-time systems such as ours. We use a fixed block size of 1024, which, for example, results in a latency of 23 ms at a 44.1 kHz sample rate.

The effects are then applied to the current track through a board object. The board is recreated at each iteration, adding effects as they are triggered by the user. In this way, the audio file starts out clean and, later on, effects already active can be preserved even when the hands are no longer visible. For applying and modifying audio effects in real time, we use the Pedalboard[4] library, developed by Spotify. On the next page, in Table I, all the effects used and the ranges achievable with our hand-tracking system are presented. The effects chosen

Effect	Definition	Range
Gain	Amount of audio signal increased by an amplifier, refers to the signal amplification applied to the output	[-20, +20] dB
BitCrush	Reduces the resolution or bandwidth of digital audio data, giving ”lo-fi” effect	[4, 24] bits
Distorsion	Applies tanh() function, deforming the signal’s original waveform	[0, 40] dB
Chorus	Duplicates the signal and slightly detunes the copy, obtaining ”ensemble-like” sound	[0, 5] Hz
Compression	Reduces the difference between the loudest and quietest parts by lowering the gain of sounds that exceed a set threshold.	[0, -40] dB

TABLE I: Effects and ranges

depend on the combination of open fingers of the right hand and are modulated by the rotation of the hand.

We provide five possible base tracks to choose from, each in the form of wav files and read through Soundfile. It is possible to change tracks according to the selected track index, corresponding to the number of extended fingers of the left hand. Furthermore, to stabilize the base track and prevent it from changing due to minor movements, the last five tracked indexes are stored in a list, and a value is selected as the next current index only if at least four of the five elements in the list are identical. When all fingers are closed and the hand forms a fist, the music is paused. This is implemented by playing silence, so that as soon as the hand shows a different number, the sound resumes.

B. Visuals

The visual aspects of the project are handled mainly by the OpenCV and Mediapipe libraries. Mediapipe’s hand landmark model detects the keypoint localization of 21 hand-knuckle coordinates (Fig.1), which are drawn on screen to better visualize it.

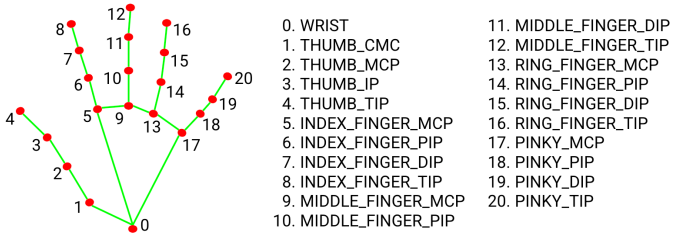

Fig. 1: The hand landmarks

These landmarks are the foundation of the project, since their relative position controls the audio plugins. The left and right hands labels are also drawn on screen along with their coordinates.

First, we compute the rotation of the right hand as the rotation of the vector extending from the wrist (Landmark 0), which serves as an anchor point, to the tip of the index finger (Landmark 8). The vector is obtained by calculating the difference in the horizontal and vertical coordinates of the two landmarks:

$$\vec{v} = (x_8 - x_0, y_8 - y_0)$$

Next, the angle between the new vector and the previous one is computed using the Numpy $arctan2(x, y)$ function (Fig.2), which takes the vector components as input and returns the angle in radians of the vector relative to the x-axis. We determine the angles relative to the horizontal axis for both vectors and then compute their difference. At each frame t, the angle in between vectors is computed as:

$$\Delta\theta = \theta_{\mathrm{t}} - \theta_{\mathrm{t\text{-}1}}$$

where:

$$\theta_{\mathrm{t}} = arctan2(x_{\mathrm{t}}, y_{\mathrm{t}})$$

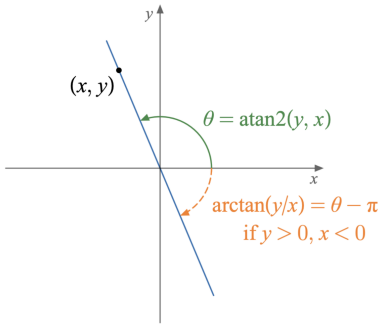

Fig. 2: The arctan() function in a 2D space

Fig. 3: Different hand rotation examples

The resulting angle, later converted to degrees, represents the hand rotation between two consecutive frames. This value is subsequently added to the overall rotation, defined as the cumulative sum of hand rotation angles across all frames. The full rotation angle is later clamped to restrict it in range [-60°,60°], for ease of movement, and then normalized to be in between -1 and 1 in order to modify the effects according to the ranges in Table I above.

Each effect is selected by using a combination of fingers (Fig. 4,5,6,7 and 8):

- **Gain:** Thumb;
- **BitCrush:** Thumb + Index;
- **Distorsion:** Thumb + Index + Middle finger;
- **Chorus:** Thumb + Index + Middle finger + Ring finger;
- **Compression:** All five fingers up;

The aperture of a finger is estimated by computing the angle between its base and its tip, which is also drawn on screen. Given a list of three landmarks for each joint (base, middle point and tip), we get their coordinates. We are going to call these points respectively *a, b* and *c*. We compute the angle between vector $\vec{BC}$ and the x axis and the angle between vector $\vec{BA}$ and the x axis. The difference between them is the final aperture angle.

$$\theta_{\text{BC}} = arctan2(c_{\text{x}} - b_{\text{x}}, c_{\text{y}} - b_{\text{y}})$$

$$\theta_{\text{BA}} = arctan2(a_{\text{x}} - b_{\text{x}}, a_{\text{y}} - b_{\text{y}})$$

$$\Delta\theta_{\text{final}} = \theta_{\text{BC}} - \theta_{\text{BA}}$$

A finger is considered to be open if its aperture is greater than 160°. For the right hand, if a finger is considered open it updates the corresponding entry in the list of joints, so that we know exactly which fingers are up. Meanwhile, for the left hand we are less strict and we just increment a counter: to select a track we don't need an exact combinations of fingers as we do for effects.

Fig. 4: Gain

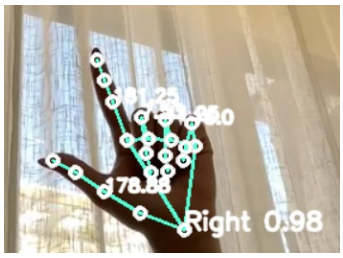

Fig. 5: BitCrush

Fig. 6: Distorsion

Fig. 7: Chorus

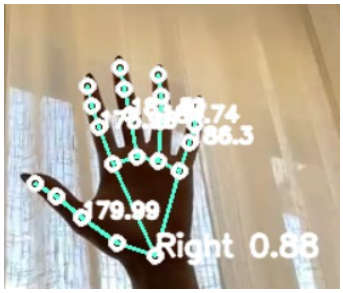

Fig. 8: Compression

C. GUI

The Graphical User Interface is created using the Tkinter library. It is implemented as a class in the *gui* file and then imported in the *main* one, where the root is created. It consists of a simple window showing the track's index and title, the rotation angle and dynamic bars which show how the effects change in real-time within their minimum and maximum values.

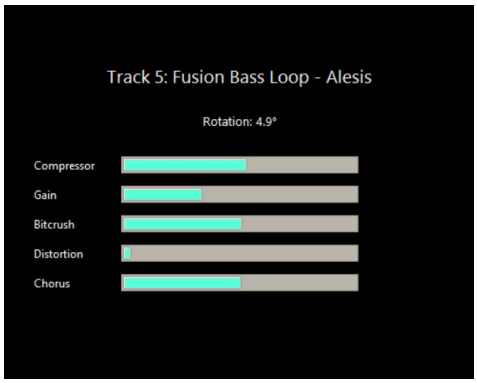

Fig. 9: GUI

III. Results

A. Problems Encountered and Future developments

1) Implementation of CPU-intensive effects

A problem that we aim to solve in the future is to implement some effects that we had initially taken into account such as Reverb or PitchShift. These plugins apply more complicated and CPU-intensive modifications to the track and needed a larger block size to avoid causing an output underflow in the audio stream. It was not feasible in a real-time system such as ours since the latency would have been too noticeable, but we could try using more light-weight versions of the effects from different libraries.

2) Light-weight gesture recognition using CNNs

We had tried to replace the current system for changing tracks with a Convolutional Neural Network that would recognize and classify the gestures of the left hand. We had found a promising dataset on Kaggle[5] and created, trained and tested our model. The model had 20 epochs,0.0001 learning rate and a batch size of 128 for training and 64 for validation. The model structure can be seen in Fig.10. The input were 64x64x1 black and white images and the network was made of four Convolutional layers, which used ReLU activation function. Every two Convolutional layers we used Max Pooling and Dropout. Then we used Flatten to produce 2048 features, followed by three Linear layers, each using ReLU. The output of the model were the logits, used to compute the loss with CrossEntropy. Adam optimizer was employed.

The model performed well. It reached a minimum loss of 0.024206 and a maximum accuracy of 0.892214835 during training, while a minimum loss of 0.001465 and a maximum accuracy of 0.959466696 during validation without overfitting (Fig. 11 and 12).

The problem came when implementing the model in our project. It was needed to create a black and white mask of the hand to match the dataset's images, but it was not possible to fully isolate the hand from the body and the background given the movements and the space required for the project, so the model performed poorly. It could be possible to fix the problem in the future by either changing the dataset used or by restructuring the hand recognition system in order to better isolate the left hand while still ensuring freedom of movement.

If possible we could extend the classification to the right hand too, but it could be more difficult since the effects are changed in real time and using the model could raise performance concerns.

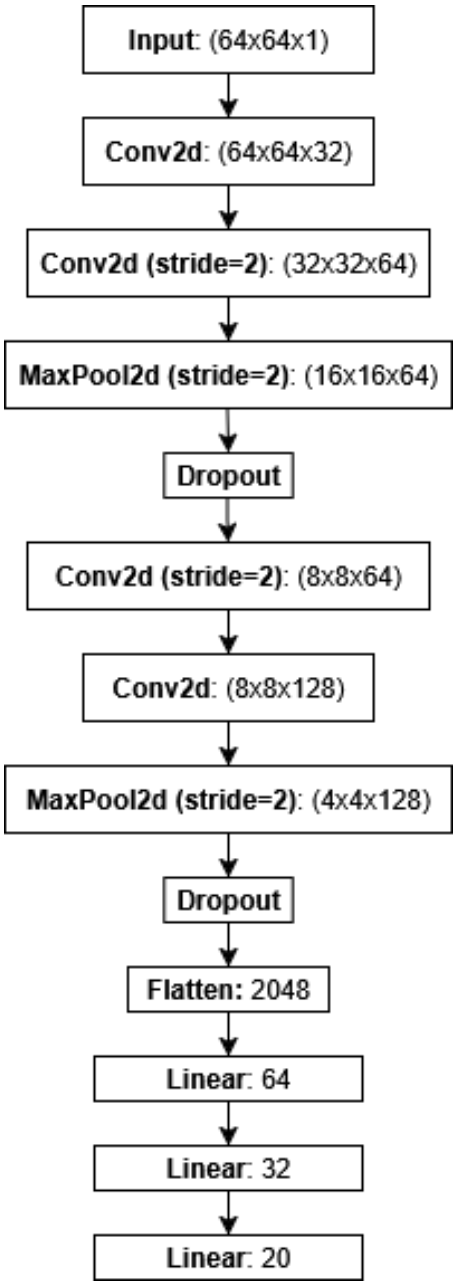

Fig. 10: Our model layers and output size (excluded batch size)

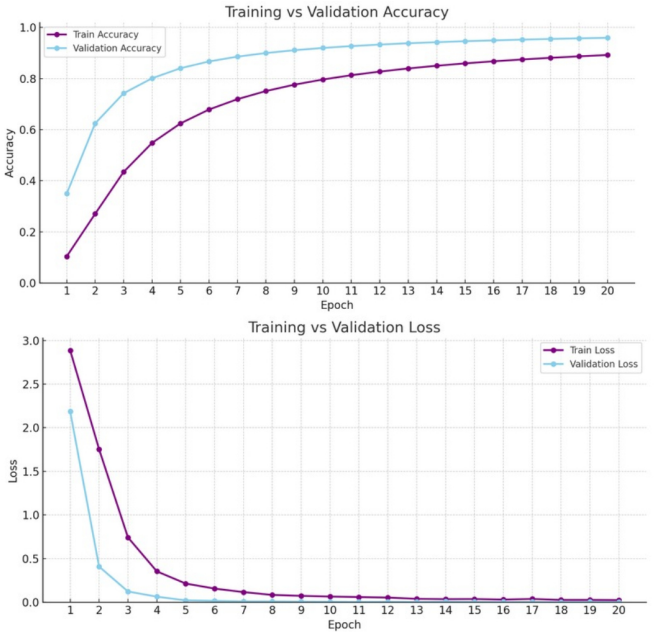

Fig. 11: Accuracy and Loss graphs

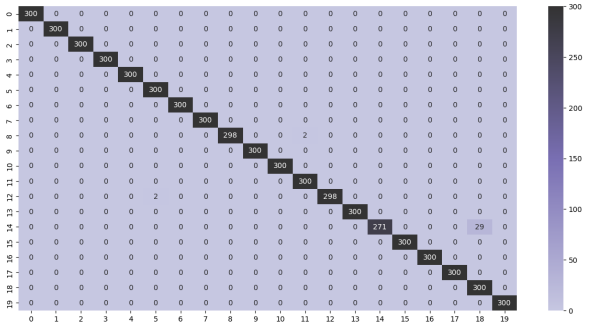

Fig. 12: Confusion Matrix after testing

B. Conclusions

The project successfully combines computer vision and audio processing to create a beginner-friendly tool for music production. By using hand gestures to control audio effects in real time and providing visual feedback through a simple GUI, the system lowers the entry barrier for experimenting with sound. The framework is functional, extensible, and demonstrates a practical approach to making music software more accessible and engaging.

References

[1] S. Hershey et al. *CNN architectures for large-scale audio classification*. (2017). URL: https://arxiv.org/pdf/1609.09430. (arXiv:1609.09430).

[2] L. Nanni et al. *An Ensemble of Convolutional Neural Networks for Audio Classification*. (2021). URL: https://doi.org/10.3390/app11135796.

[3] M. Lim, N. Kotsani, and M. Ceyzar. URL: https://github.com/monlim/Handmate-Effects.

[4] Spotify's Audio Intelligence Lab. URL: https://github.com/spotify/pedalboard.

[5] Rishabh Arya. URL: https://www.kaggle.com/datasets/aryarishabh/hand-gesture-recognition-dataset/data.