NYU TANDON SCHOOL OF ENGINEERING

Real-Time Digital Signal Processing Lab
ECE-GY 6183, Fall 2022

Project Report

Virtual Paper Piano

Submitted by: **Jocelyn Zhang (jz5838)** and **Vaishnavi Rajput (vr2229)**

December 22, 2022

TABLE OF CONTENTS

INTRODUCTION

In this project, we developed a virtual paper piano that allows users to play piano notes using hand gestures on paper which has black and white segments that are like keys on a real piano. It is equipped with a graphical user interface as well, which provides the users the functionality to play traditional notes or create their notes by changing the frequency of the notes. It also provides the users with an option of conventional notes or DIY notes, implementing three kinds of effects. The piano is controlled by a computer and uses digital signal processing to generate high-quality sound. Our goal was to create a user-friendly interface that would allow users to easily control the volume and effects of the notes they play.

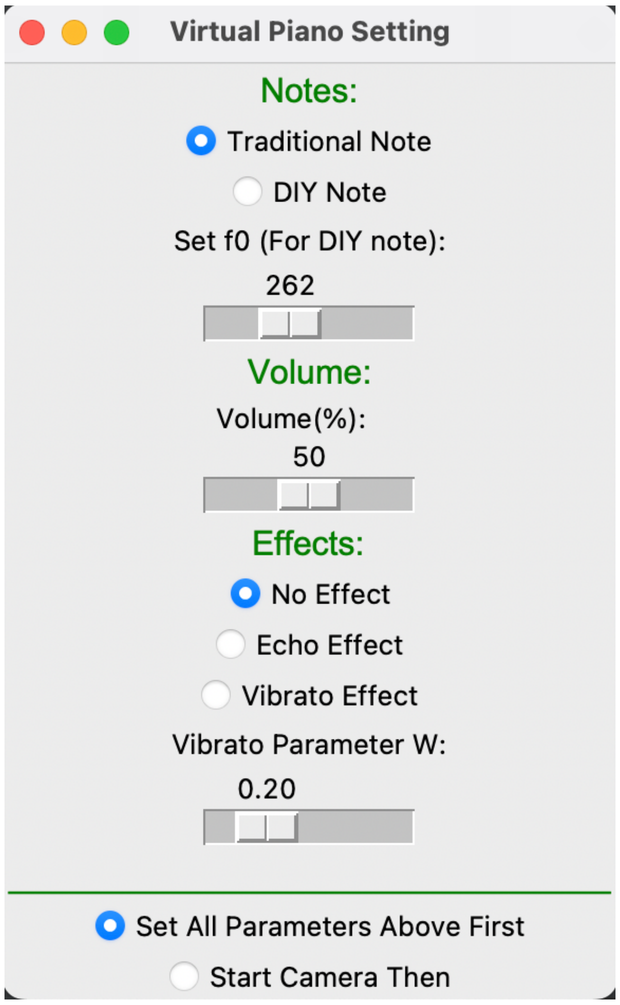
METHODOLOGY

2.1 Graphical User Interface

Designing the graphical user interface (GUI) for the paper piano using Tkinter involved creating the visual elements that users will interact with to control the effects and volume of the notes.
This required creating widgets such as buttons and sliders using Tkinter's built-in classes and functions. We used the ‘Button’ class in Tkinter to create radio buttons and specified the text labels and commands to be executed. We also used the ‘Scale’ class to implement sliders and specified the range and resolution.

We created one set of two Radio Button which provide the users the choice between traditional notes and DIY notes, and two sliders, which helps the users set the frequency of the notes and adjust the volume. The second set of three Radio buttons helps the user to choose which effect they want to apply in the DIY notes. When the users are ready with the effects, they want for the notes to play, they can turn on the camera.

To lay out the GUI elements in a visually appealing and intuitive way, we used Tkinter build-in parameters to optimize different button and scales arrangement. We specified the position and size of the widgets within the GUI window. The first part notes give two options Traditional Note(Note C starts at 262 Hz) or DIY Note, (Note C is chosen by the user). The second part is volume, users could choose from 0% to 100% as volume. The third part is the voice effect, there is no effect, echo effect, or vibrato effect for users to choose from. For the vibrato effect, there is a parameter W to control the vibrato effect(W=0 as no vibrato). The last part is used for the situation where all the parameters are settled, the camera could be opened then. Figure 1 represents the GUI of our work.


Virtual Piano Setting

Notes:

[✓ Traditional Note] DIY Note

Set f0 (For DIY note): 262

Volume:

Volume(%): 50

Effects:

[✓ No Effect] Echo Effect Vibrato Effect

Vibrato Parameter W: 0.20

[✓ Set All Parameters Above First] Start Camera Then

Figure 1: Graphical User Interface

2.2 Open-Source Computer Vision Library

We captured video from a webcam using OpenCV's cv2.VideoCapture function. This function returned a VideoCapture object which is used to read the video frames in real-time. The Webcam.py file has the python code for the implementation of the video-capturing process.

The conventional piano has seven notes, C, D, E, F, G, A, and B, so we printed a paper with seven notes on it. Figure 2 represents the paper that we used to implement our work, captured through the camera. Then the video frames were processed to identify the piano keys. This involved applying image processing techniques such as thresholding, filtering, and edge detection to segment the piano keys from the background and extract their features. We split the image into seven cells.

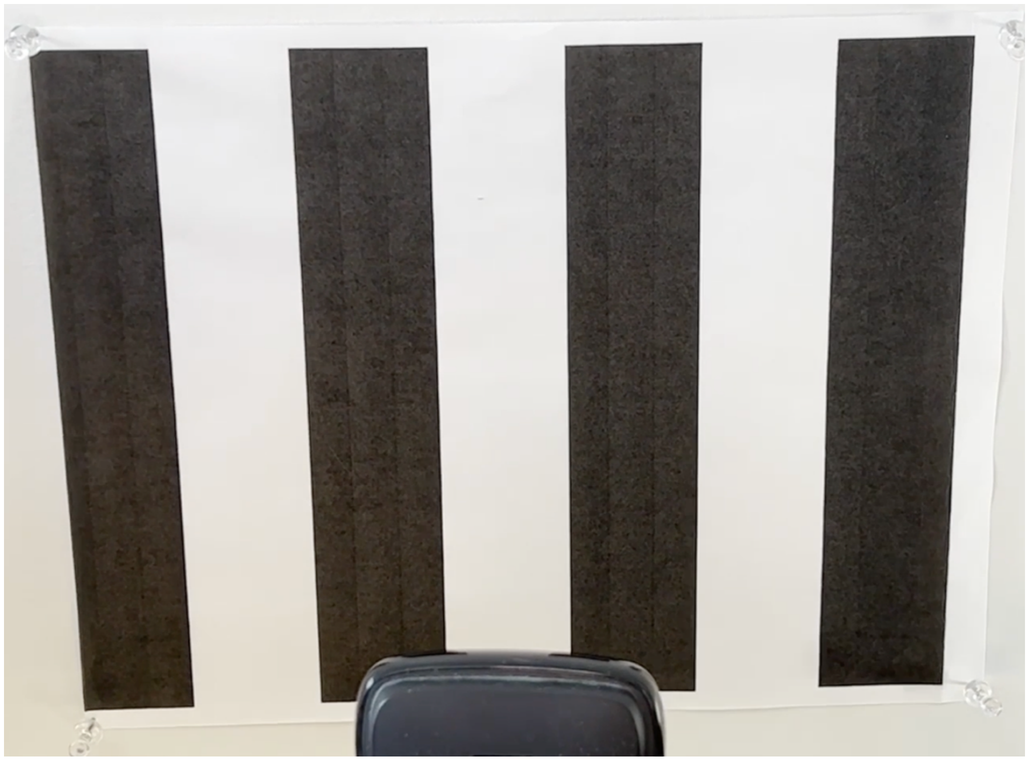
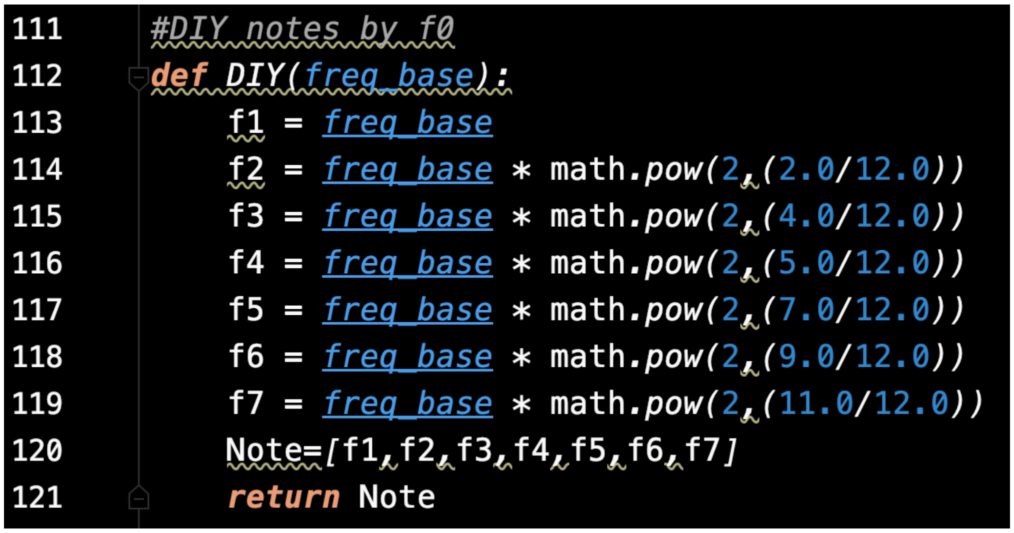


Figure 2: The paper with printed notes

The motion detection technique was used to detect user input. Here, we used Detection.py to get the most active cells. By returning the most active cell, we could define which note the user's finger is on. We implemented it using thresholding, which involves dividing an image into two or more classes based on the intensity values of the pixels. In OpenCV, thresholding can be easily implemented using the cv2.threshold function. This function takes an image as input and applies a threshold to the intensity values of the pixels, resulting in a binary image where pixels with intensity values above the threshold are set to a certain value and pixels with intensity values below the threshold are set to another value.

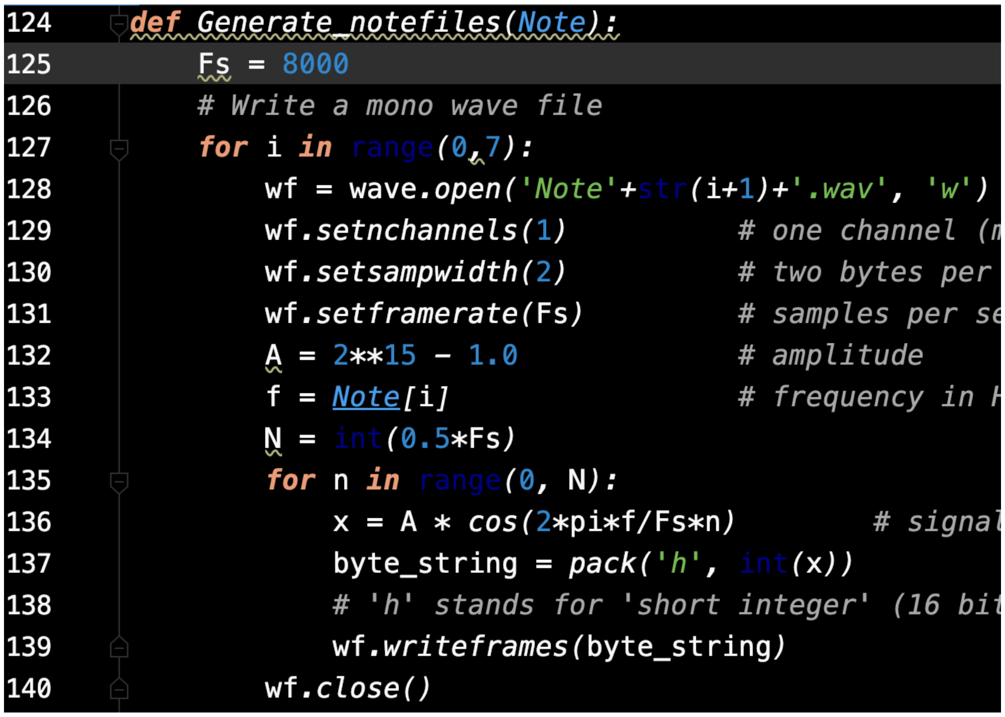
2.3 Note Generation

The 7 notes which are played are customized. When the user selected Traditional, the frequency for C, D, E, F, G, A, and B will be NOTES = [262, 294, 330, 350, 393, 441, 494]. If the user selected DIY note, the frequency will be generated by the DIY function as Figure 3 shows. For piano played with no effect, these 7 notes will be played due to these frequencies. For piano played with echo/vibrato effects, these 7 notes will be generated as wav firstly, just as figure 4 shows, then added different effects within their respective echo/vibrato play class.



```
#DIY notes by f0
def DIY(freq_base):
    f1 = freq_base
    f2 = freq_base * math.pow(2,(2.0/12.0))
    f3 = freq_base * math.pow(2,(4.0/12.0))
    f4 = freq_base * math.pow(2,(5.0/12.0))
    f5 = freq_base * math.pow(2,(7.0/12.0))
    f6 = freq_base * math.pow(2,(9.0/12.0))
    f7 = freq_base * math.pow(2,(11.0/12.0))
    Note=[f1,f2,f3,f4,f5,f6,f7]
    return Note
```

Figure 3: DIY function

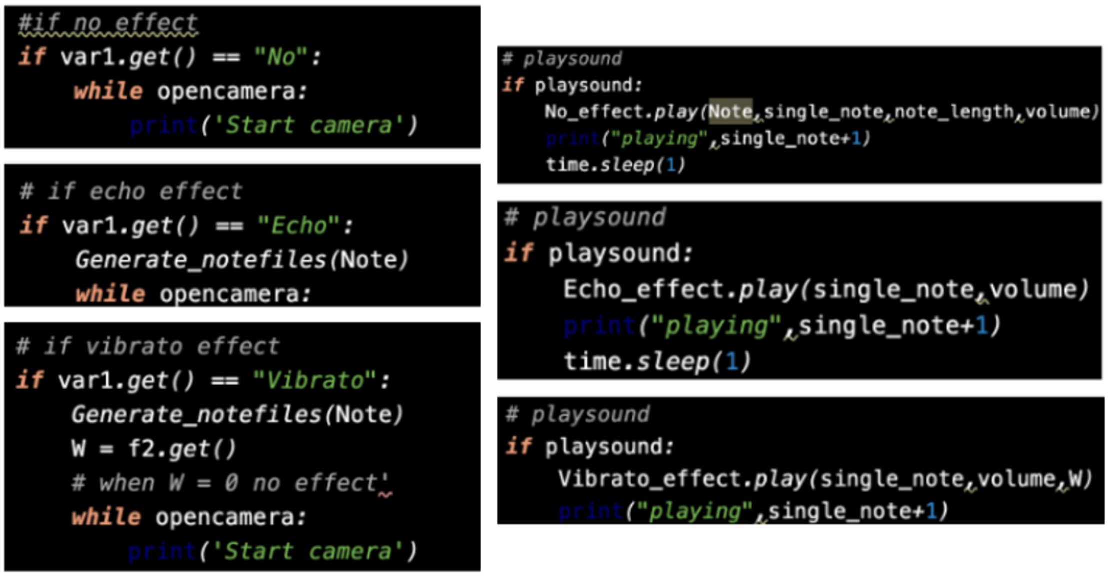


```
def Generate_notefiles(Note):
    Fs = 8000
    # Write a mono wave file
    for i in range(0,7):
        wf = wave.open('Note'+str(i+1)+'.wav', 'w')
        wf.setnchannels(1)          # one channel (m
        wf.setsampwidth(2)          # two bytes per
        wf.setframerate(Fs)         # samples per se
        A = 2**15 - 1.0             # amplitude
        f = Note[i]                 # frequency in H
        N = int(0.5*Fs)
        for n in range(0, N):
            x = A * cos(2*pi*f/Fs*n)        # signal
            byte_string = pack('h', int(x))
            # 'h' stands for 'short integer' (16 bit
            wf.writeframes(byte_string)
        wf.close()
```

Figure 4: Generate_notefiles function

2.4 PyAudio

The sounds of the piano notes were created using the PyAudio library. It has its own set of functions that help in sound synthesis. We used functions like setnchannels, setsampwidth, setframerate, and others to set different parameters. Different frequencies were assigned to each note and the sound was generated according to the motion of the finger. When the camera is open, the main.py will call the play function from three different choices: no effect/ echo effect/ vibrato effect. Figure 5 represents different branches that depend on what effect users choose. For three playing branches, the respective sounds will be generated from three 'play' functions from three python classes.



```
#if no effect
if var1.get() == "No":
    while opencamera:
        print('Start camera')
```

```
# if echo effect
if var1.get() == "Echo":
    Generate_notefiles(Note)
    while opencamera:
```

```
# if vibrato effect
if var1.get() == "Vibrato":
    Generate_notefiles(Note)
    W = f2.get()
    # when W = 0 no effect
    while opencamera:
        print('Start camera')
```

```
# playsound
if playsound:
    No_effect.play(Note,single_note,note_length,volume)
    print("playing",single_note+1)
    time.sleep(1)
```

```
# playsound
if playsound:
    Echo_effect.play(single_note,volume)
    print("playing",single_note+1)
    time.sleep(1)
```

```
# playsound
if playsound:
    Vibrato_effect.play(single_note,volume,W)
    print("playing",single_note+1)
```

Figure 5: Logic and branch for three sound effects

2.5 Effects

We provided users with options between the echo effect and the vibrato effect. The echo effect refers to the simulation of the acoustic phenomenon of echo using algorithms and mathematical techniques and is used to create a wide range of creative and expressive sounds. It is one of the essential parts of the toolkit of any audio engineer or music producer and is widely used in a variety of musical and creative contexts.

For our project, the echo effect was implemented by applying a delay of 50 milliseconds to a sound signal and then mixing the delayed signal back in with the original signal. The Echo_effect.py file has the python implementation of the echo effect.

The vibrato effect is an effect that involves modulating the pitch of an audio signal periodically. It is typically achieved by slightly shifting the pitch up and down at regular intervals, creating a wavering or "vibrating" effect. Vibrato is often used in music to add expression and depth to a performance, and it is commonly used in instruments such as guitar, violin, and vocal music. It can also be used to add vibrato to synthesized sounds or to process a recorded audio signal to add vibrato. The effect can be controlled by adjusting the depth, rate, and waveform of the modulation, as well as other parameters such as the amount of feedback or the shape of the waveform used to modulate the pitch.

In this project, we gave the user the control to modulate the pitch and apply a vibrato effect to the sound produced by the piano notes. The Vibrato_effect.py files have the python implementation of the vibrato effect.

2.6 Reference

We referred to others' work about the camera open and detention function [1], these parts of the codes are not our original work and are stored in the 'other' file. The rest codes which are in the main file are original.

RESULT

In this project, we developed a paper piano using Python and OpenCV. The piano was designed to be played using paper with seven notes imprinted on it. It was capable of synthesizing the sounds of various notes and implementing two different sound effects, namely, the echo effect and the vibrato effect. We carried out a series of experiments to evaluate the performance of the paper piano. These experiments included measures of sound quality and playability.

The paper piano was capable of producing a wide range of sounds with good clarity and tonal balance. The piano was able to accurately reproduce the pitch of the notes being played and was able to simulate the timbre of different effects with reasonable realism. The piano was able to accurately detect key press events and trigger the appropriate audio signals on time.

The following figure 6 represents the setup of a web camera and printed paper and a user trying to play note E.

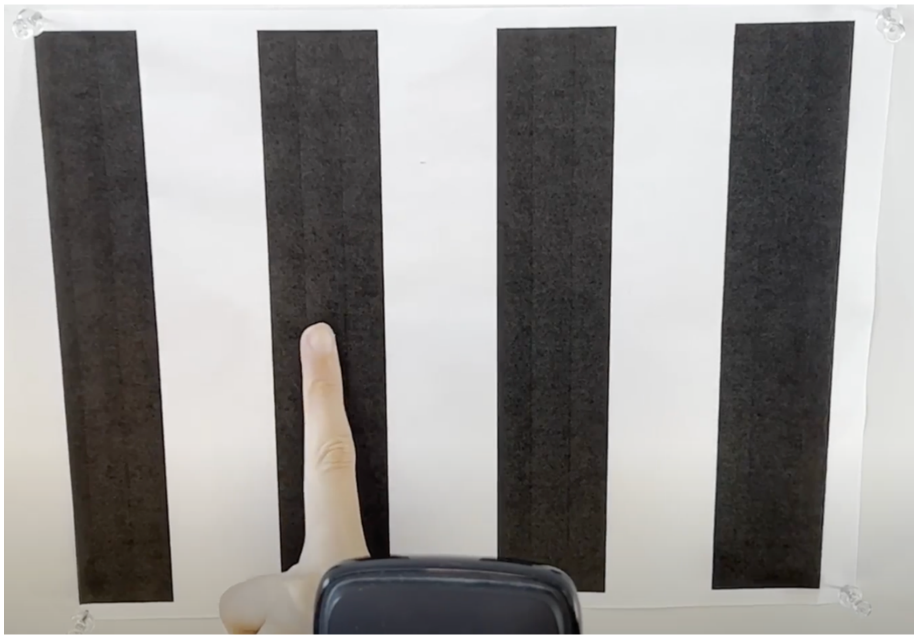


Figure 6: Project setup where a user is trying to play a note

CONCLUSION

The results of our experiments showed that the virtual paper piano was able to produce a wide range of sounds with good clarity and tonal balance and provided a smooth and responsive playing experience. It demonstrates the potential for using Digital Signal Processing techniques and computer vision to create new and innovative musical instruments that can produce a wide range of sounds and providing a fun and engaging musical experience.

There are several areas where the paper piano could be improved in the future, including the addition of new features and the implementation of physical feedback. We believe that this project represents a promising example of the potential for using Python and OpenCV to create innovative and interactive musical instruments, and we look forward to exploring further developments in this field.

REFERENCE

[1] https://rdmilligan.wordpress.com/2015/10/22/paper-piano-using-python-and-opencv/