Domenico Panuccio
Damian Dziedzic
Digital Signal Processing Lab
Semester project

The focus of this project is to convert visual input into visual and audio output. This is done by taking in video from a webcam, and outputting piano note audio signals and displaying keys being pressed on a digital representation of a keyboard. This is achieved in python through the use of the openCV and Pygame libraries.

The program exists with 4 major sections which are here explained. Prior to any image processing, python takes a sampled file of middle C played on a piano and processes it. For n semitones, in this case n = 50 ranging from -25 to 25, middle C is transformed into the n domain and all frequencies are shifted in accordance with which semitone is currently be composed. The note is then transformed back into the time domain. This process creates a noticeable slowing, or speeding up depending on semitone, of the note. In order to correct this, the note is up or down sampled over an appropriate range as to preserve the appropriate speed of note. It should be noted that, due to this variate sampling, the further the semitone is from 0, the lower the sound quality is due to sound compression.

Once all notes have been generated, the program begins its preprocessing stage. During this stage, the program displays the current video input and prompts the user to press q when satisfied with the input image. This allows for proper alignment of the piano with the camera. Additionally, this circumvents certain safety features of webcams which may grey out the first few images of video capturing software, which would prevent single first frame image capturing. After this, the program maps the generated keys to a saved set of keyboard keys. These are then used to initiate pygame for key inputs and audio outputs. Thus the output frame in opencv is generated.

The program then enters the main loop, the first half of which is the image processing frame. A frame is taken from video and is preprocessed in the same way as the base frame. Between these two images, a delta threshold image is constructed. This image is then separated into sections corresponding to the input keys. These sections are then individually summed up to see if the total change over the key is greater then a predefined threshold value, and if it is, the key is marked as pressed in a binary matrix, and a circle is placed on the visual output. Whenever a key is marked as pressed, a new key down event is artificially generated in pyagme and pushed to the event queue. Similarly, if a key is marked as pressed, but fails the threshold check for a frame, a key up event is generated and pushed to the queue, and the circle is removed from the visual output.

The final stage processes the pygame events generated that frame. This is done internally through pygame instead of manually compiling and packing each audio signal, as the overhead would be relatively large and induces a slight but noticeable lag in the output audio. When a keydown event is processed, pygame begins outputting the audio signal corresponding to that key. On keyup event processing, the audio file begins a fadeout with a duration 500ms, so as to make the end sound resonant as it would in an actual piano.