Image Analysis and Computer Vision - Project
F09. Visual motion analysis of the player's fingers

Martina Licul (10987985)

February 2024

The Assignment

The assignment was to extract the motion of fingers of the keyboard player from a video sequence. In order to do so, I have firstly needed to obtain a suitable video sequence for the task. then extract and preprocess the frames. After that, detect the keys, followed by detecting the hands and the fingertips. And finally, detect the pressed keys and show them.

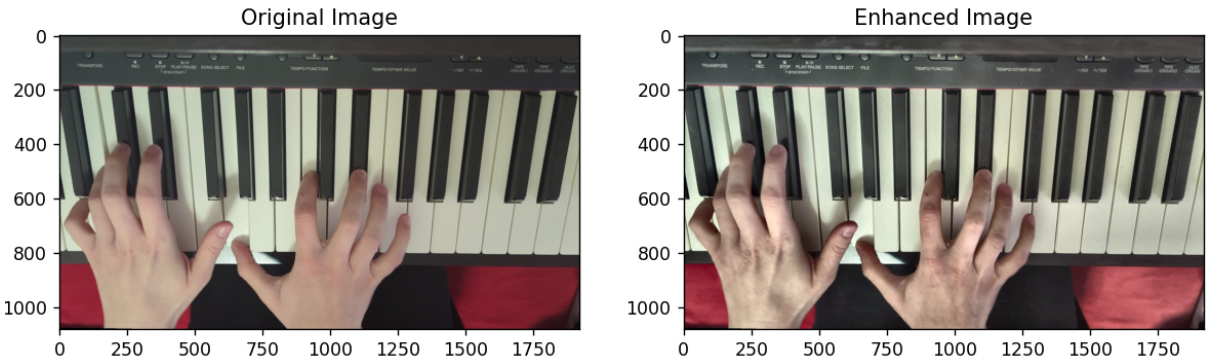
I choose this project topic because in my family there are many keyboard players, and, therefore, I found it very interesting.

1 Camera setup and video filming

For this assignment, I have filmed myself playing on the piano the last five measures of the *Prelude, Op. 28, No. 20 in C minor "Funeral March"* by F. Chopin. I have filmed the video with my phone camera with Full HD resolution and 30 frames per second. The chosen camera setup was to put the camera as where the note stand is, tilted to look at the keys. I have tried using different camera setups, but decided to continue with this one as it was giving the best results.

2 Frames extraction and enhancing

Firstly, I have extracted the frames from the video sequence. Secondly, I have enhanced the color and contrast of the frames using *CLAHE*.



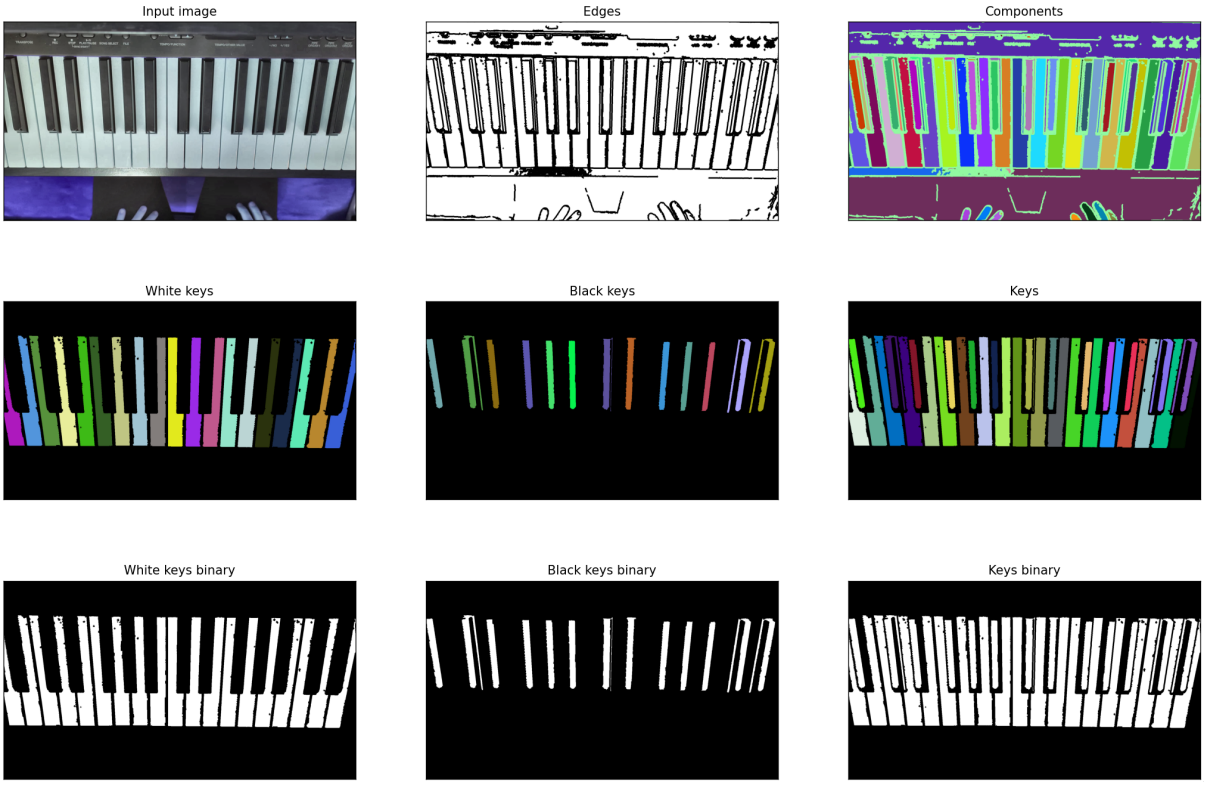
3 White and black keys detection

The first frame from the video sequence contains only the keyboard, no hands on the keyboards. Therefore, it is suitable to use that frame as the image for keys detection.

The first step to detect the keys is to extract the edges on the image. To do so, I have used Canny edge detection. Furthermore, to close the little gaps, I have thickened the edges. Next, I have obtained the number of components and the components matrix of the image using the function *connectedComponents*

from the *cv2* library.

To detect the keys, it was necessary to detect the white and black keys separately. However, the procedure for detecting the keys was the same, just the parameters where different. Based on the average intensity of the color of a component, I was able to determine whether it could be a key by color or not. Also, if a component was to small, I did not consider it at all. In this way, I have filtered out some non-key components, but there were still some left. To get rid of them, I have calculated the average size of the remaining components, and based on how much the size of each of the components differ from the average, I have selected which to remove. Eventually, I was left with only the key components.

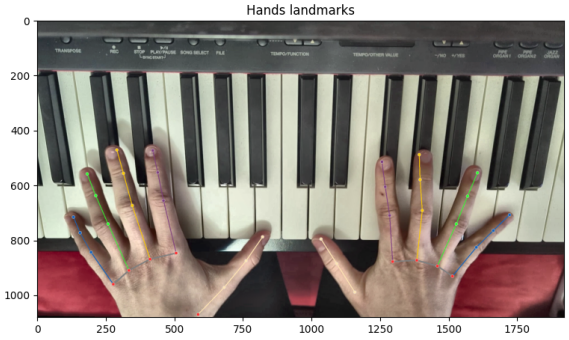


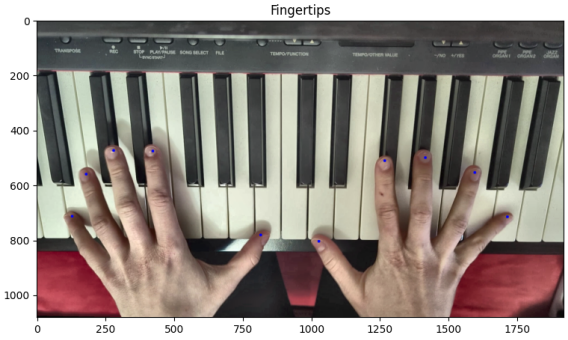
4 Hands detection

To perform hand detection, a process is implemented to isolate skin-colored regions within the input image based on color segmentation in the YCrCb color space. By establishing upper and lower bounds for skin color, a mask is generated to identify potential hand regions. Subsequent morphological operations, including erosion, dilation, Gaussian blurring, and thresholding, refine this mask to reduce noise and enhance the clarity of hand detection. Contours are then extracted from the processed mask, and the two largest contours, typically representing the hands, are selected.
These contours are superimposed onto a duplicate of the original image, visually indicating the detected hand regions. I have also created a binary image where white represents the hands, and black all the other. In such a way, it was easaier to modify the tresholds to obtain better results.

5 Fingertips detection

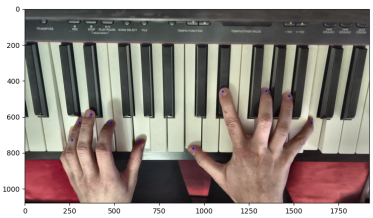
I have utilized the *MediaPipe* library in Python to detect both hand landmarks and fingertips in images. Initially, I employed the *MediaPipe Hands* model with specific configurations, such as model complexity, minimum detection confidence, and minimum tracking confidence, to accurately identify hand landmarks within the images. These hand landmarks represent key points on the hand, including joints and fingertips. Subsequently, I extracted the locations of the fingertips by iterating through the detected hand landmarks and selecting specific landmark indices corresponding to the fingertips. These indices were chosen based on the hand anatomy, where landmarks 4, 8, 12, 16, and 20 represent the fingertips of each hand.

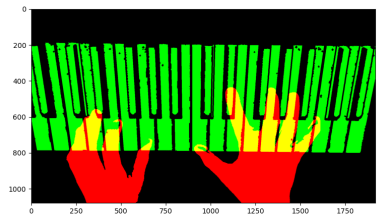


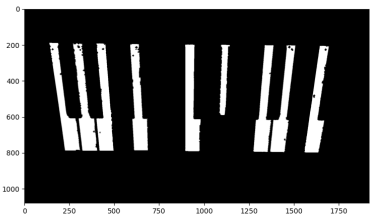


6 Pressed keys detection

To detect the pressed keys, I began by identifying candidate keys. This involved iterating through each fingertip detected in the image and examining the keys surrounding each fingertip location. Utilizing a defined proximity range, I searched for keys within the vicinity of each fingertip coordinate. If a key was found within this range, it was considered a candidate key for potential interaction.



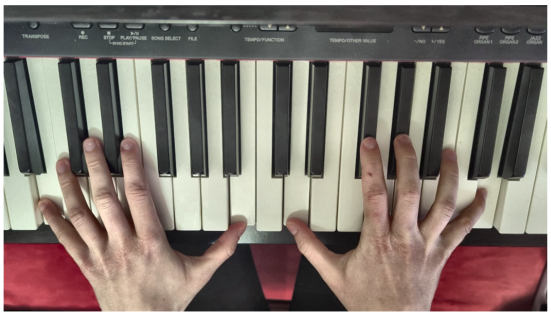


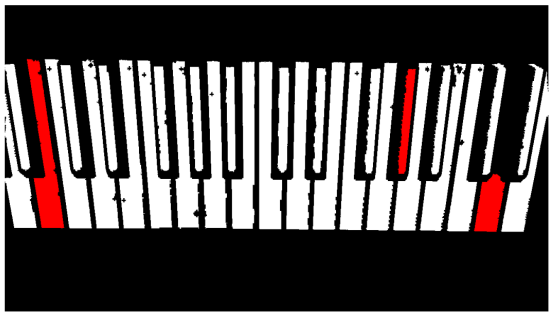


Each of the candidates is considered separately. Firstly, I get the components matrix of the current frame. The for each candidate, I filter the components that have some parts in common with the candidate (and are not too small) Then I calculate the total area of those components, and the total area where those components are overlapping with the corresponding candidate. If the ratio passes the threshold, it is considered to be pressed. The ratio is actually the difference of the shape of the key in the base frame and in the current frame.

7 Results

In the process of developing the solution, each processed frame is augmented by overlaying it with an image of the keyboard, where pressed keys are visually highlighted in red. These augmented frames are saved individually. Eventually, these augmented frames are compiled into a cohesive video. The outcome, as depicted in the provided illustration, demonstrates a satisfactory result considering the complexities encountered throughout the development phase. Despite the notable achievements, the program occasionally misidentifies a key press for a brief duration and may overlook certain pressed keys.





Above there is a frame from the result video. As it can be seen, the program missed a pressed key, but managed to detect the other pressed keys.

Conclusion

In conclusion, addressing this problem entails navigating through several intricate steps, each crucial to the overall success of the task. Any slight inadequacy in execution at any stage could significantly impact the final results. The processes involved, including frame extraction, enhancement, key identification, hand and fingertip detection, as well as the selection of key candidates, demonstrate satisfactory performance.

However, the primary area requiring improvement lies in the logic used to determine whether a key is pressed or not. While initial attempts focused on monitoring color and shape changes of the keys, these methods necessitate further refinement. It is worth considering not only the alteration in the candidate key but also the changes in its neighboring keys, as this could provide valuable contextual information for accurate key press detection. Thus, enhancing the key press detection algorithm remains a priority for achieving more precise and reliable results in this complex problem domain.

A thing to think about is to analyse the graph of the fingertip positions changing over time. The graphs peeks could be the pressing/releasing points.
Running the codes on a more powerful computer would also be beneficial. As a workaround to lengthy execution times, I had to resize the frame to a smaller size. Resizing the frame to a smaller size could diminish the quality of the images by reducing the level of detail and clarity. This reduction in image size may result in important features being lost or becoming less discernible, potentially impacting the accuracy and effectiveness of subsequent analysis or processing tasks.

References

[1] S. Kang, J. Kim, S. Yoon (2019). Virtual Piano using Computer Vision. https://arxiv.org/pdf/1910.12539.pdf

[2] Y. Baabyof (2018). PyPiano. GitHub. https://github.com/yehudabab/PyPiano/tree/master

[3] M. Cavalcanti (2022). Hand Tracking and Finger Counting in Python with MediaPipe. Geekering. https://www.geekering.com/categories/computer-vision/marcellacavalcanti/hand-tracking-and-finger-counting-in-python-with-mediapipe/

[4] P. Suteparuk (2014). Detection of piano keys pressed in video. Department of Computer Science, Stanford University. https://stacks.stanford.edu/file/druid:bf950qp8995/Suteparuk.pdf