SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

IMAGE ANALYSIS AND COMPUTER VISION

PROJECT 2023 – 2024

Supervisor:

Professor Vincenzo Caglioti

Students:

Jakub Tomasz Stępniak (246371)

Jagoda Różalska (246526)

Academic year: 2023/2024

Table of contents

1. Introduction

The goal of our project was to detect the finger movements of a keyboard player based on a video sequence. We choose a video depicting a musician playing the piano.

To reach that goal we devide one big problem into several smaller problems like:

- Hand detection
- Fingers detection (in their three articulations)
- Keyboard Detection
- Detection of Pressing keyboard key by single finger

2. Dependencies

- Python – programming language
- OpenCV – library to for image analysis algorithms
- Numpy – library for numerical operations in Python
- Mediapipe – library with deep learning models

3. Proposed Algorithm

We are proposing below algorithm:

1. Reading video with someone who plays on intrument with keyboard.
2. Processing every single frame ts fast as possible (Real Time Processing as a goal).
3. Return processed frame according to arguments passed by user (marking keyboard, hand, fingers and pressing by finger key) and displaying it.

4. Assumptions

We are introducing some assumptions connected to our project:

- Detecting keyboard is done on specific keyboard (white and black keys on black piano) with specific orientation (camera view is from overhead of player)
- Camera can move but cannot strongly change its orientation
- All keyboard keys should be visible all the time (except these keys behind hands)
- At least first frame where whole keyboard is detected should be without visible hands

5. Structures used in program

- Pipeline – main class of processing pipeline which include Processor and all frames
- Processor – class responsible for making all processing on received frame. It includes classes which represents KeyboardMaster class and Hands class
- KeyboardMaster – class which is responsble for detecting keyboard and return its object to the processor
- Keyboard – class which represents detected keyboard
- Hands – class which instance should hold two objects of both hands, also this class is responsible for correct detection of the hands on the received frame.
- Hand – class which represent single hand. Every instance should have objects which represents all fingers of hand
- Finger – Base Class from which there are inherit all fingers classes

6. Hand and Finger Detection

These 2 steps we were be able to solve in the same step using the same tool. To do that we are using Deep Learning library ‘mediapipe’ and its ready model to detect hands. We also create instances of classes Hands and Hand for create objects which will hold all importatnt from our point of view informations received from the model.

Informations which are mostly needed by us are:

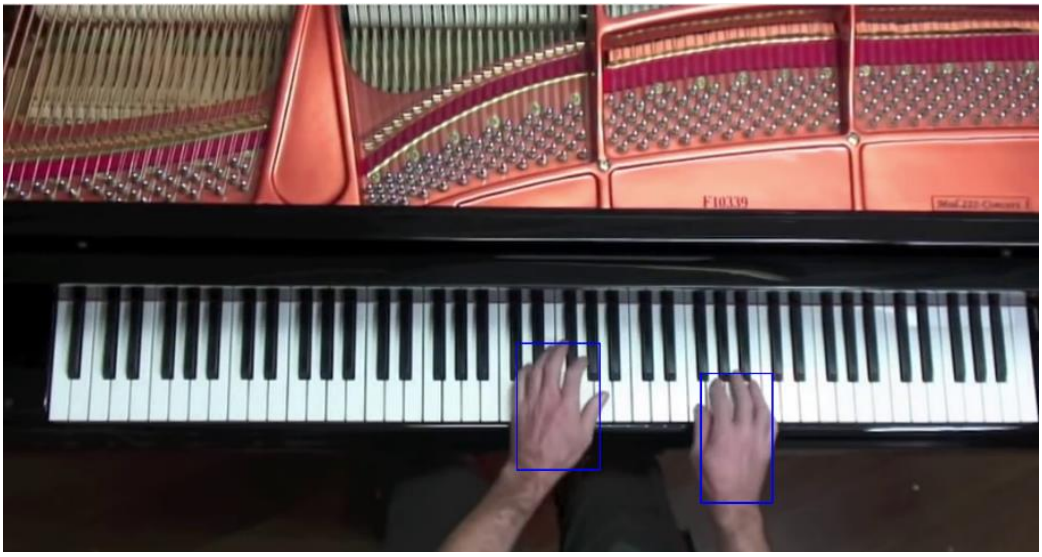
- Bouding boxes of every hand
- Side of detected hand (left or right)
- Landmarks which tells us about current position of fingers and hand

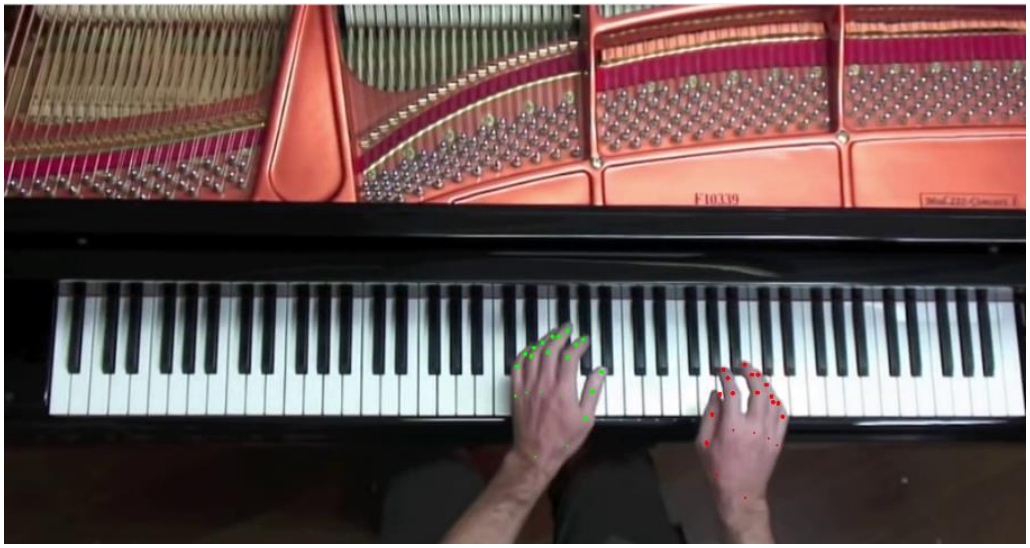
From model mediapipe.solutions.hands we are receiving landmarks and side of hand, but bounding box of the hand we have to calculate ourselves using extreme landmarks.

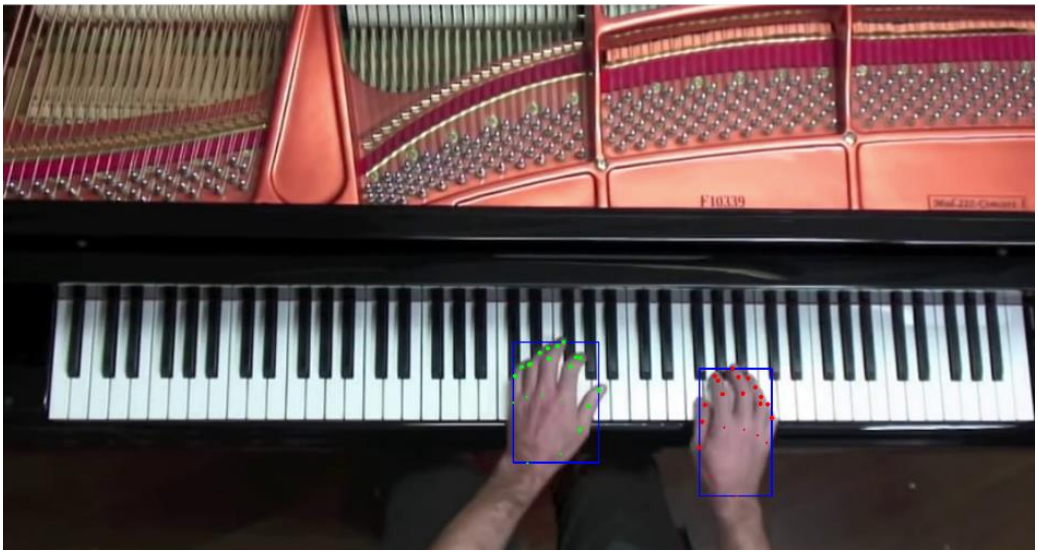
```
def calculate_bounding_box(self, landmarks):
    x_max = 0
    y_max = 0
    x_min = float('inf')
    y_min = float('inf')

    for lm in landmarks.landmark:
        x, y = lm.x, lm.y
        if x > x_max:
            x_max = x
        if y > y_max:
            y_max = y
        if x < x_min:
            x_min = x
        if y < y_min:
            y_min = y

    return BBox.create_from_points(
        x_min*self.frame_shape[1],
        y_min*self.frame_shape[0],
        x_max*self.frame_shape[1],
        y_max*self.frame_shape[0]
    )
```

Received landmarks are in range of 0 to 1 so we have to multiply them with frame shape.

Results:





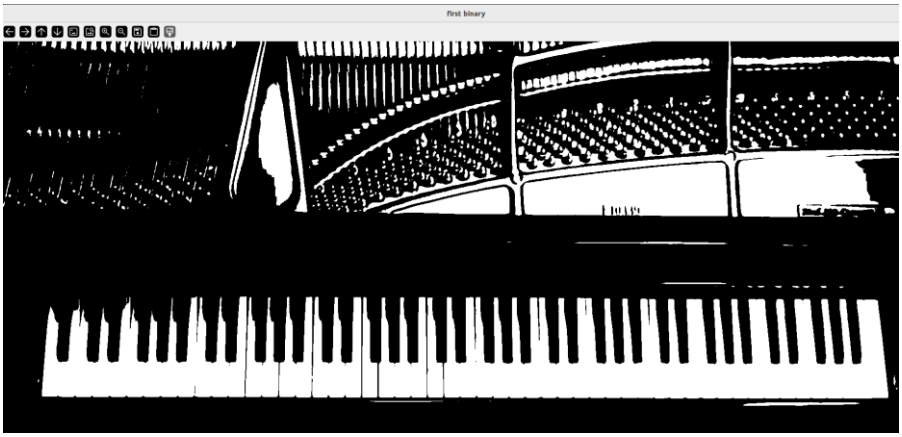


7. Keyboard Detection

We couldn't find any ready-made deep learning model for detecting keyboards, and we also didn't have enough time and resources to train our custom model. We decided to create a custom algorithm that will detect the keyboard from overhead. Initially, we wanted to make it as universal as possible, but with a moving camera, different types of keyboards, and different keyboard orientations, it's impossible to create such a robust algorithm without using deep learning techniques.

Proposed algorithm which works for given video:

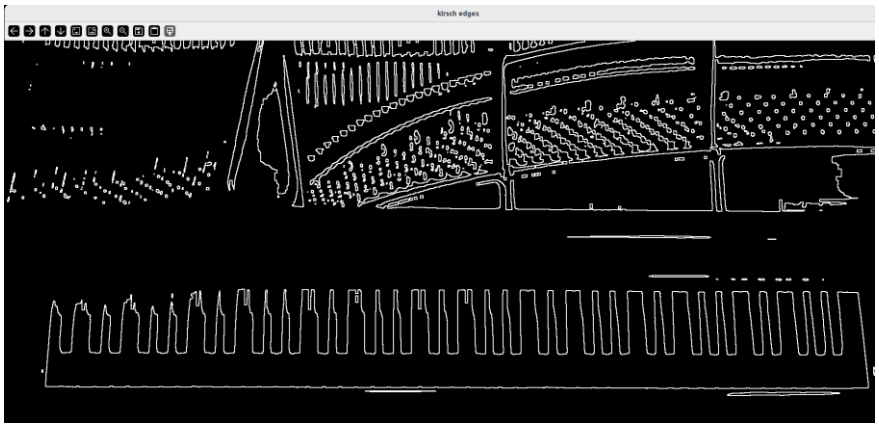
1. Changing Processing Frame Color Space From BGR to Grayscale
2. Threshold of the image on thresh equal to 130



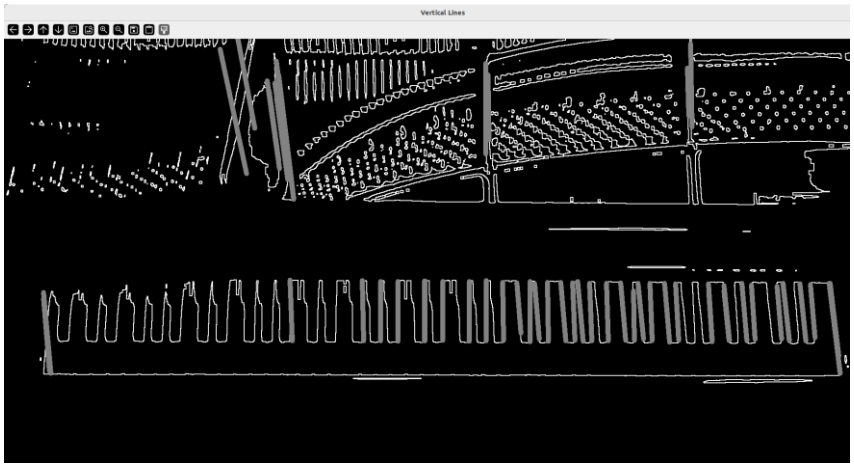
3. Operation of closing (morphological operation) for connection some regions which are very closed to each other after thresholding but without any changes in size of the objects

4. Applying custom version of Kirsch algorithm for detection of the edges

```python
def kirsch(img, divider=3):
    KIRSCH_K1 = (np.array( object: [[5, -3, -3], [5, 0, -3], [5, -3, -3]], dtype=np.float32) / divider)
    KIRSCH_K2 = (np.array( object: [[-3, -3, 5], [-3, 0, 5], [-3, -3, 5]], dtype=np.float32) / divider)
    KIRSCH_K3 = (np.array( object: [[-3, -3, -3], [5, 0, -3], [5, 5, -3]], dtype=np.float32) / divider)
    KIRSCH_K4 = (np.array( object: [[-3, 5, 5], [-3, 0, 5], [-3, -3, -3]], dtype=np.float32) / divider)
    KIRSCH_K5 = (np.array( object: [[-3, -3, -3], [-3, 0, -3], [5, 5, 5]], dtype=np.float32) / divider)
    KIRSCH_K6 = (np.array( object: [[5, 5, 5], [-3, 0, -3], [-3, -3, -3]], dtype=np.float32) / divider)
    KIRSCH_K7 = (np.array( object: [[-3, -3, -3], [-3, 0, 5], [-3, 5, 5]], dtype=np.float32) / divider)
    KIRSCH_K8 = (np.array( object: [[5, 5, -3], [5, 0, -3], [-3, -3, -3]], dtype=np.float32) / divider)
    first_edges = np.maximum(
        cv.filter2D(img, -1, KIRSCH_K1),
        np.maximum(
            cv.filter2D(img, -1, KIRSCH_K2),
            np.maximum(
                cv.filter2D(img, -1, KIRSCH_K3),
                np.maximum(
                    cv.filter2D(img, -1, KIRSCH_K4),
                    np.maximum(
                        cv.filter2D(img, -1, KIRSCH_K5),
                        np.maximum(
                            cv.filter2D(img, -1, KIRSCH_K6),
                            np.maximum(
                                cv.filter2D(img, -1, KIRSCH_K7),
                                cv.filter2D(img, -1, KIRSCH_K8),
                            ),
                        ),
                    ),
                ),
            ),
        ),
    )
    return first_edges
```

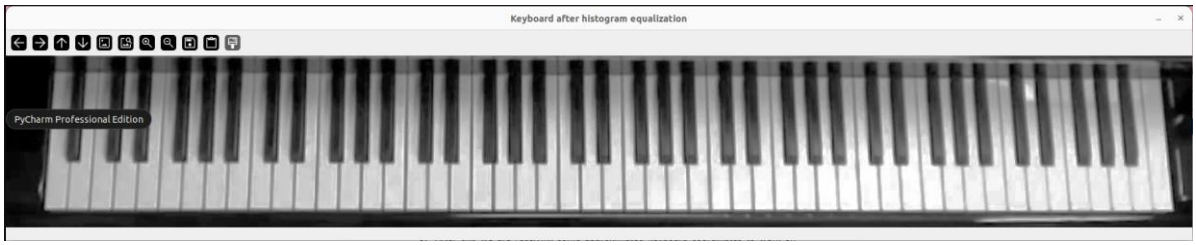


5. Another thresholding operation also on thresh equal to 130
6. Applying Hough Transformation implemented in OpenCV library
7. Looking for only vertical lines (with some margin) in order to receive lines of the black keys on keyboard. We are doing this to get coordinates of the keyboard with some margin

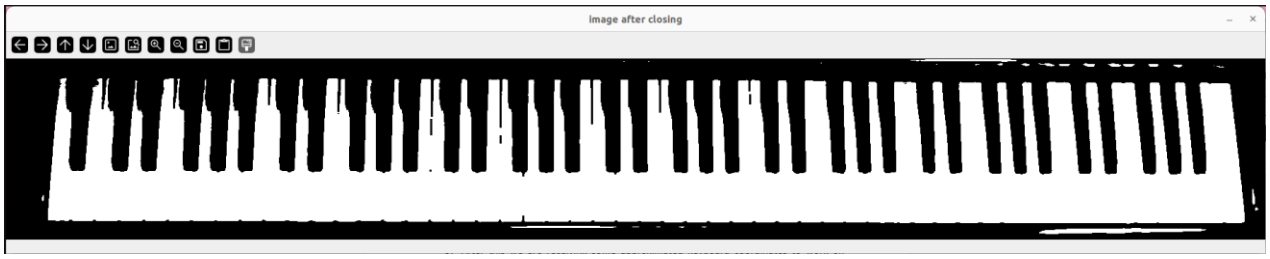


8. To recognize lines from black keys from other lines except filtering them by angle, we are filtering them also by their position on the image. We are creating several bins to which lines at certain level are classified based on their y coordinate. After classification of the all lines we are counting all bins and choose bin with the biggest number of lines as a bin with black keys lines.
9. After this we are receiving some approximated keyboard coordinates to work on much smaller image with almost only keyboard

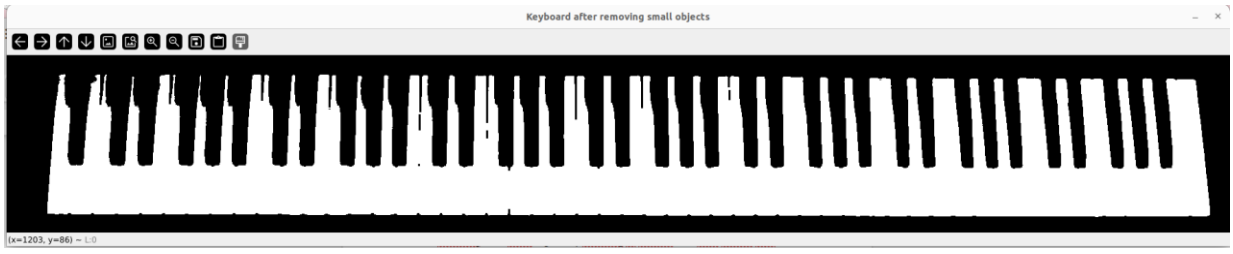
10. Equalizing histogram of the imageto receive more equalized white and blacks



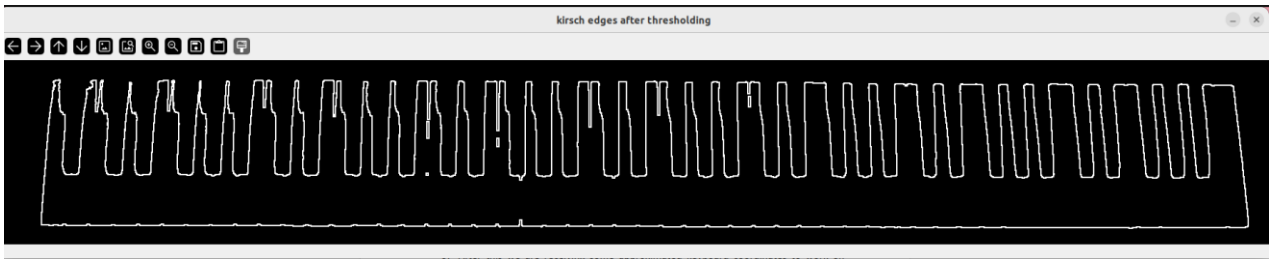
11. Otsu Threshold which will find correct thresh by analyzing histogram
12. Vertical Close Operation (structural element only in one, horizontal dimension)



13. Removing small white regions by removing all conours with fileld smaller than 1500

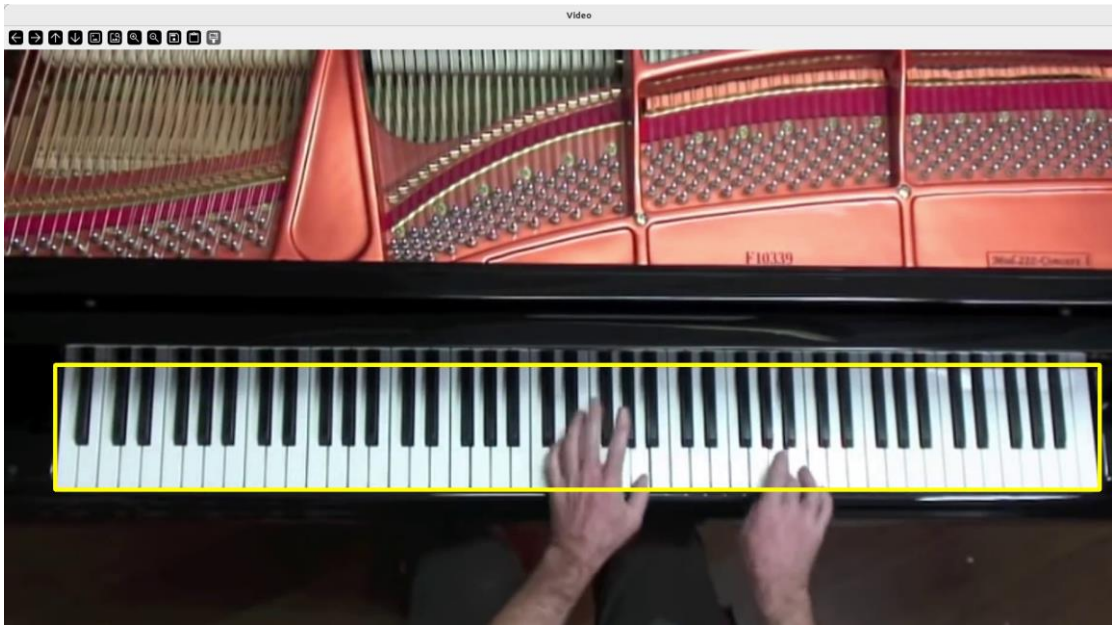


14. Kirsch Operator to get edges
15. Otsu Binarization



16. Summing all variables in horizontal direction on the image to receive 1D vector with sum of all columns of the image
17. Find the first and last big enough edge which will be left and right coordinate of keyboard
18. Hough Transform to get horizontal lines
19. Find the longest line which is down line of the keyboard
20. return only keyboard from whole original image

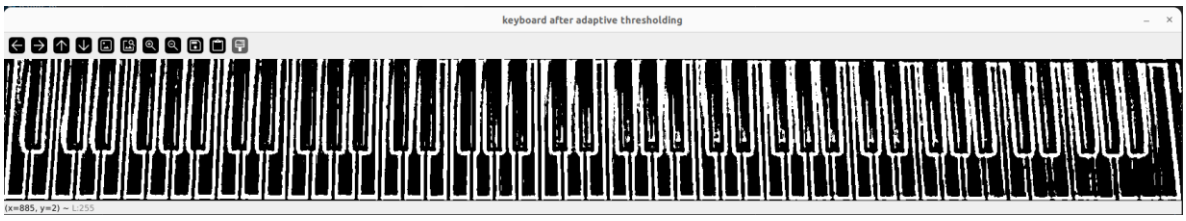
Final Result:



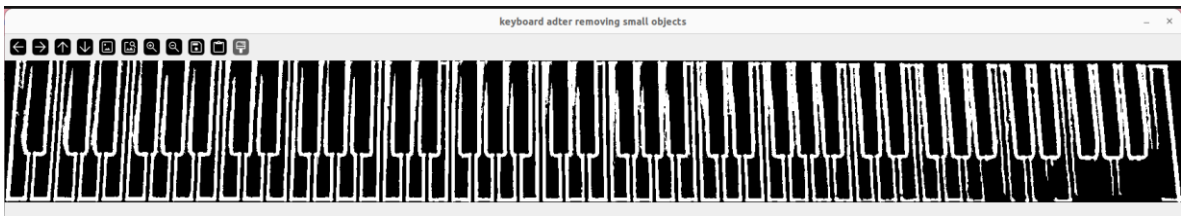
8. Pressing Key Detection

To detect a key press, we agreed to use a reference keyboard created from the frame without any visible hands. The main idea is to subtract the received frame of a single key from its corresponding key on the reference keyboard. Thanks to this, we can calculate the change between these two images and check if a key was pressed by noticing that its position has changed. It seems like the best way to achieve this because analyzing the finger from this camera perspective is pointless as we don't have information about the third dimension. To make this work, it's very important to have a keyboard image created as a Keyboard Object. To make this work, we must have an image of the keyboard processed in a way that maximizes every change of key. To do this, we are proposing the following algorithm:

1. Convert Colorspace from BGR into Grayscale
2. Perform adaptive thresholding to deal with changing in lights and to make important lines between keys (after pressing this breaks becoming bigger)

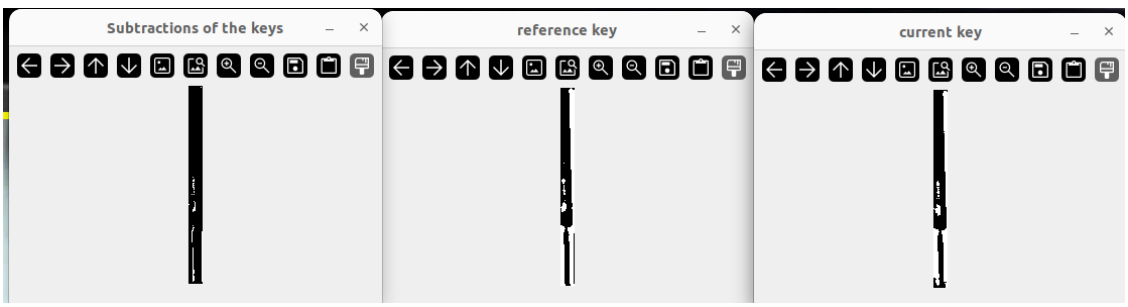


3. Removing noise by removing smaller contours than 500.



We are proposing below algorithm to perform pressing detection on key:

1. Calculating values of bounding box of the checking key based on last landmark of the finger (fingertip), bounding box of the hand (we don't want to have in our comparing region other fingers) and bounding boxes of keyboard
2. Create Region Of Interest of the reference key and current key
3. Calculate their field
4. Subtract both keys
5. Count all black (or white) pixels in subtraction ROI
6. Check if ratio between black pixels and field is bigger than threshold set by ourselves. If yes we detected pressing of the key



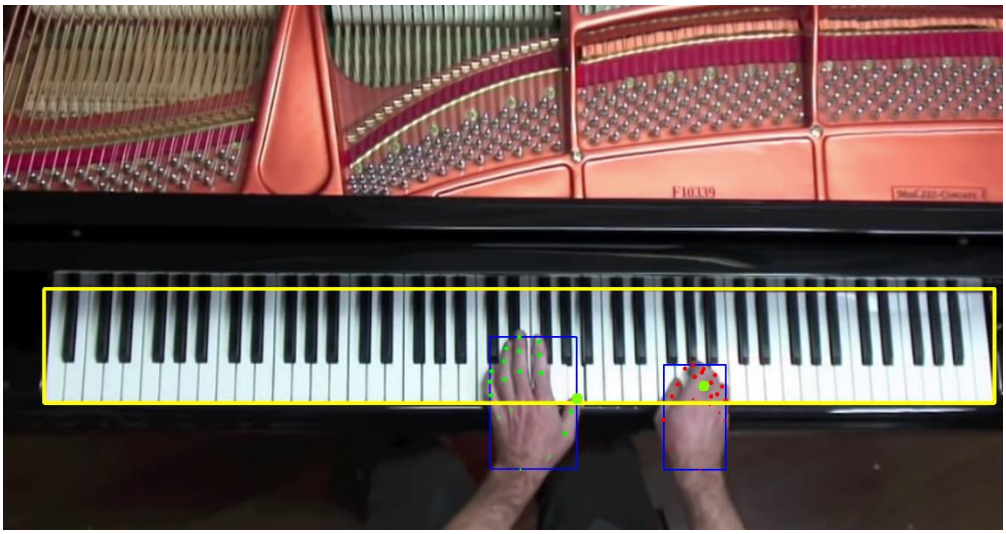
Results:

9. Tests

To run app paste this command into command line (the best is in concrete virtual environment with specific requirements)

`python src -v bach.mp4 -hm true -km true -fp true -fm true`

Results which we are receiving:



Keyboard, hand, and finger detection works very well. Pressing detection has more troubles. There are many false positives and sometimes pressing is not detected.

Nevertheless, most of the pressings are detected correctly. A big disadvantage is the fact that the processing time on our local laptop is not real-time and the processed video is much slower than the original.

10. Conclusion

It is very hard to achieve perfect pressing detection with the current approach. The first thing to get detection at a good ratio is perfect keyboard detection with the accuracy of a single pixel. This is so important because of the subtraction of the two images. If there is any shift, it can greatly influence the result. Another difficulty is the lack of features from this camera perspective which can be used for pressing detection. The only noticeable change is the increasing gap between keys for white keys and a smaller field for black keys, which are very hard to detect. Big problems also arise from fingers which disturb the examined keys of other fingers.

Model for hands and fingers detection works very well on all conditions which we were testing. Unfortunately this model makes real time processing imposible and processed video was much slower than original one.

Future improvements of this project would include creating a custom deep learning model which would detect the keyboard in real time in different scenarios. It would require a complicated dataset with specific ground truth bounding boxes of keyboards on images, which would be very time-consuming to build.

Another improvement could be the quantization of the model for detecting hands to improve its performance because we can see that it is working quite slowly compared to the incoming frames.

Also applying more preprocessing especially for keys subtraction could results more resistant for camera noices and small shifts of detected keyboard.

Good idea would be further research about different techniques of feature extraction to compare pressed and not pressed key which would include for example frequency filters like FFT or gabor filter.