Detection of Piano Keys Pressed in Video

Potcharapol Suteparuk
Department of Computer Science
Stanford University
neungs1@stanford.edu

***Abstract*—This paper presents a a method for detecting and tracking the keys that are pressed in a piano playing video. The proposed pipeline uses image differences between video frames, as well as other image processing techniques that include Hough transform, morphological dilation and erosion, and region labeling.**

I. Introduction

There are various tools and software in the piano/keyboard community that helps one learn the keystrokes of the songs. Some software such as Synthesia also visualizes the sequences and duraion of keystokes in the piece. Others create music sheets from Youtube video with no sheets provided. However, several tools either are not accurate or do not show hand movements and key tracking that would be helpful for learners.

Previous works including tracking hands and fingers to help learners better play the piano. Various algorithms in Computer Vision and image processings are used to detect hands including skin colors detector or hand shape detector. The result is, however, still not accurate as pianists' hand shapes are different from normal hand gesture in that the fingers are not clearly protude and often occlude each other. To fix this issue, some work such as [2] design a crevice detector that utilize the drop in luminance in the areas between fingers to detect the side of the fingers instead. To make the matter more complicated, the five fingers are even harder to distinguish due to their similarity across the hands. Simple solution as in [3] is to use markers to differentiate each of the finger. Others such as [1] requires a depth sensor to detect the positions of the fingers. The results are proven to be quite positive.

Few previous works, on the other hand, have tried using image processings techniques to detect the actual keystrokes on the piano. [1] uses a depth sensor to detect the differences in depth for keys. However, as not everyone has access to the depth sensor, our goal is to develop a new pipeline that uses only simple image processings techniques. In this paper, we will utilize the image differences between frames that pressed keys make due to light reflection and shadow to determine the correct keys. The result is not perfect but hopefully it will be more robust when combined with the fingering algorithms presented in the previous works.

II. Input Video and Tool Setup

We will use a video camera to capture the piano playing. To simplify the main goal, we will restrict the position of the camera to be fixed directly above the piano, clearly expose the entire piano region used to play the particular piece. The choice of using a direct view is to eliminate the possibility of keys occluding each other. In addition, we will require the user to take the picture of the bare piano where no hands are occluding any keys. This will be used in the detection of black keys and white keys. See Figure 1.

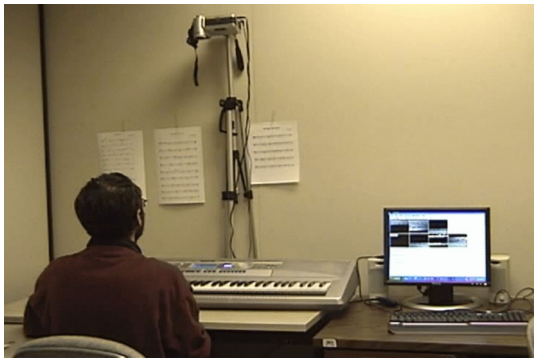


Fig. 1. Ideal setup for input videos. Note that this is not the actual input for this project. Source: [2]

Fig. 2. Example of input videos used for this paper.

For this project, we did not use the actual setup described above. Rather, we use a set of Youtube videos, which is recorded at 30 fps, with camera setup above the piano as our inputs. See Figure 2.

III. Methodology

Figure 3 illustrates the proposed processing pipeline. We first split the videos into frames. The preprocessing stage rotates and crop out the key section. This is simple since the piano view is clear and the keys have certain patterns to be recognized. Next, we extract the hands out. For this paper, we will use the skin color to create an MAP detector that will differentiate the color of the hand from the piano keys. Having the hands in the image will confuse our algorithm since a hand above a key does not mean that the key is pressed. We mask out the hands and then detect the black and white keys positions using the region labeling. This is where we use the bare piano

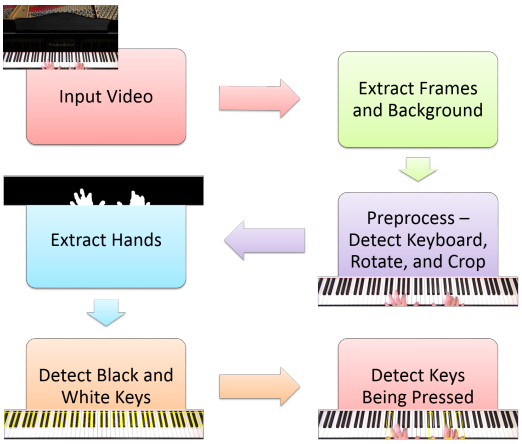


Fig. 3. Pipeline for our proposed algorithm.

image we ask the user to take before playing the video. The main task of the algorithm is to do image difference between nearby frames. We will describe each process in further details below.

A. Pre-processing

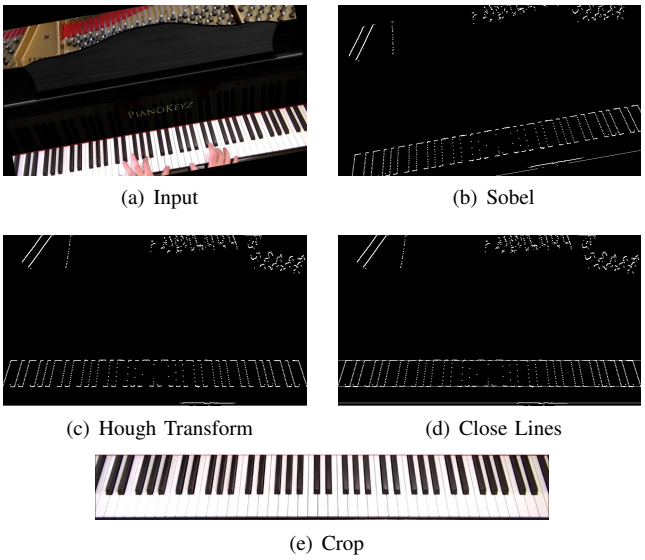
To make the rest of the computation easier, we will crop out only the black and white keys of the piano out from each frame. To do this, although the camera is fixed above the piano, we still need to rotate the frames to align with the piano keys direction. We will assume that the area around the white keys are not white. With this assumption, it is then easy to extract the main part of the piano out.

1) Rotate: We first apply a gradient operator to the grayscale image with certain threshold to find where the grayscale values jump significantly. For this purpose, we use a Sobel operator. There are other ways to achieve this. One might use a Canny edge detection or zero-crossings of Laplacian of Gaussian to find strong edge in the frame. We choose to apply a Sobel filter mainly because it is simpler and faster than other edge detection techniques, perfect for a process that does not need to be overly robust. We clean up the binary image by removing small regions using region labeling technique. We then apply a Hough transform to find the dominant angle/slope of the orientation and rotate the image according to that angle. Since the orientation of the piano keys are very dominant, Hough transform will almost surely correctly rotate the frames. Figure 4(b) and 4(c) show the results of the rotation.

2) Crop: After we get the correct orientation, we can see the three dominant lines in the gradient image clearly: the top of the black keys, the bottom of the black keys, and the bottom of the white keys. we use morphological image processing to further enhance these lines. It is now easy to detect the lines (simply find the rows with highest white pixels). See Figure 4(d) and 4(e).

B. Hand Detection

Having a hand over a key does not mean that key is being pressed. While it is useful to know the position of the hands, for the purpose of this paper we will remove them from the main pipeline and only use it as helper information later in the

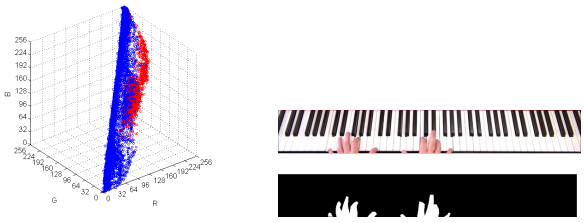


(a) Input (b) Sobel

(c) Hough Transform (d) Close Lines

(e) Crop

Fig. 4. Pre-process stage



(a) Training pixel RGB values in 16 bins. The red points are hand pixels and the blue ones are non-hand pixels.

(b) Mask resulting from MAP detector. Dilate the result a little to include area around the hands as well.

Fig. 5. Hand extraction stage

algorithm. Since the hands can be easily differentiated from the keyboard by the color of the skin, we train a MAP detector base on the pixel color in RGB. Using 10 training pairs of RGB images cropped from the input frames and the hand-drawn mask of the hands, we bin the pixel RGB values into 16 bins in each channels. Figure 5(a) clearly shows the separation between non-hand and hand pixels. Note that after we get the mask of the hands in each frame, we dilate the area by a 5x5 structured element to include any pixels that might be affected by the hands or their shadows.

C. Black Keys Detection

The next step is to determine the location of black keys and white keys. This is where we need the user to take a picture of the piano without the hands first. We start with the black keys, which is easier. Since the keyboard contains mainly black and white colors, we apply the region labeling to the negative image of the binarized image. Note that we have to remove a portion of rows from the top and bottom to remove any outliers that connect the black keys. Figure 6 shows the result. Note that the binarization of the cropped input image also needs to be cleaned up first by removing small regions/noise.

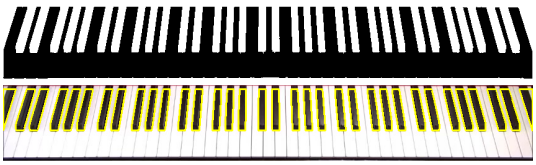
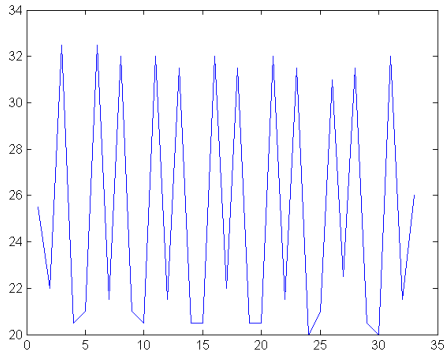
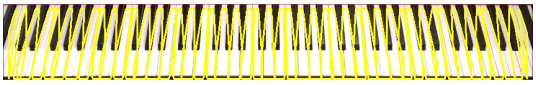


Fig. 6. The black keys detected using region labeling.



(a) The horizontal distances between the center of each black keys. The peaks in the graphs determine the large gaps where two white keys reside.



(b) White keys regions illustrated by the convex hull of all the pixels in the region. Note that the black keys regions are removed.

(c) Erode the black regions to create buffer between black and white keys.

Fig. 7. The white keys detection stage.

D. White Keys Detection

The white keys are harder to detect due to light reflection that sometimes blend nearby keys together as one. This poses difficulty in using region labeling or edge detection techniques like Canny edge detection. It is not impossible, however, since the keyboard has a repetaed pattern of black and white keys. So even if we cannot separate all white keys from each other, we can still separate only certain part that has clear separation and then extrapolate the separation to other areas using the key pattern.

The above method might improve the performance of our pipeline, but for the purpose of this paper, we will leave it as a future work. In this paper, we detect white keys by scaling the nearest black keys to fit the size of the white keys, which has a standard ratio against the black keys. First, we detect the pattern of the black keys and determine where the larger gaps are. Figure 7(a) shows the horizontal distances between the center of each black keys. After we know the pattern of the keys, we can now determine for each white key which black key is the one closest to it. We scale and translate the regions to fit the white key size and then remove the areas that overlapped with black keys. We also dilate the white region

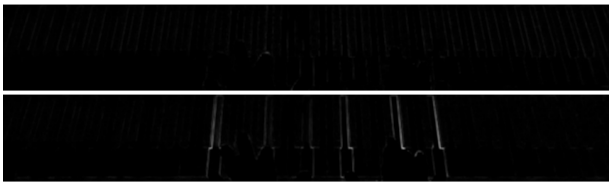


Fig. 8. Image difference between close grayscale frames. (Top) No keys are pressed. (Bottom) A D-major chord is played.

by a structure element of 3x3 to cover more area as detecting changes in the next section will need to detect changes around the keys as well. Finally, we erode the previously found black regions by a structured element of 5x5 to leave some room between black and white keys. See Figure 7.

E. Key Pressed Detection

The main effects of pressed keys that we utilize in this paper are the changes in light reflection and shadows, as well as the side of neighboring keys that is exposed when the current key is pressed. To detect such changes, we will find differences between grayscale frames via mean of subtraction.

1) Reference Frame Selection: At first, it is logical that we try to find the image difference between the current frame and the background keyboard. We need to mask the hands off the current frame using the mask learned in Hand Detection section. However, this does not prove to be robust and the results are very noisy. This is due to the fact that the camera being moved a little while playing. Although, we align the frames to reduce the MSE, the errors still persist, which probably stem from the camera view being pushed and tilted little by little to distort the view. Therefore, the last frame and the first frame is very misaligned.

To circumvent this issue, we instead find the difference between the current frame and a few previous frames. Frames that are close in time are closely aligned as well, although we still need to align the frame by minimizing the MSE. In addition, the hand positions are alike. All in all, frames that are close in time are suitable to be used as a reference frame for image differentiation. But we still do not know how close the frames should be since keys are not pressed with the same speed throughout the video. The answer is simply differentiate the current frame with a few previous frames and find the maximum one out of the candidates. This ensures we detect all the changes without skipping or repeating any of them.

In this paper, we differentiate each frame from the frames that are $1/3^{\text{rd}}, 1/6^{\text{th}}, 1/10^{\text{th}},$ and $1/15^{\text{th}}$ seconds before it. We discard the union area of the hands in both frames in the differentiation and only consider the changes on the actual keyboard.

2) Detecting Changes: In Figure 8, we can see the changes detected in the image difference. Note that we need to align the frames before differentiate them. Also after the differentiation, we suppress noise by removing small regions, as well as blur it using a low-pass filter (in this paper we use a box filter of size 3).

Now, since we already know the regions of each key, detecting where the changes occur is straightforward. Note

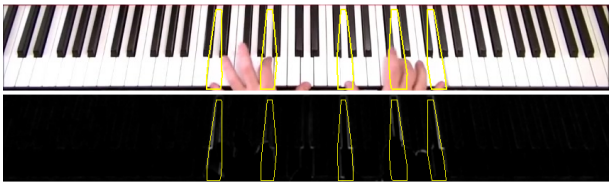


Fig. 9. Keys detected as being pressed.

that to detect a pressed white key, not only do we need to look at the shadow from the nearby black keys, but the side of neighboring white keys that are exposed is also a sign of change. This is why we dilate the white key regions in the White Keys Detection section. On the other hand, the changes in black keys mainly stem from the changes in light reflection on the key itself. We do not want to misinterpret the changes occur on the neighboring white keys to affect the changes in the black key. That is why we erode the black key regions. For each key region, we find the sum of the changes occurred in that region (use Figure 8 for summation). If the sum of changes is above a certain percentage of the area of the region, then we detect that there is a significant change in that key. In this paper, we use a threshold of 3% for the white keys and 1% for the black keys. Note that the black keys rely heavily on the light reflection, which is hard to detect, and hence we need to lower the threshold.

Having changes detected does not mean that the key is pressed. The key might be pressed in the reference frame and is now released. To keep track of keys pressed, we maintain a binary sequence P_i for each frame i where each bit represents each key on the keyboard. If a bit is 1, the corresponding key is being pressed in the respective frame. Now for each frame, we create another binary sequence C_i where a bit of 1 means that we detect changes in the corresponding key. Simply xor P_r and C_i where frame r is the reference frame of frame i, and we will get the binary sequence P_i of frame i. Figure 9 shows the result of a frame when D-major chord is played.

3) Refinement: As seen in Figure 9, we miss the D key in the middle of the keyboard. The reason is that the camera is right above the keys in the middle and therefore it is very hard to see the changes even for human eyes. One refinement that we make is to use different threshold for different part of the keyboard. For white keys, we use a linear scale from 4% at the left and right border to 2% in the middle. For black keys, this is from 1.5% to 0.5%.

Another refinement is to clean up some of stray detection. Because we maintain a binary sequence fully based on the previous one, if there is a mistake in the previous one, we might be pressing a key for the rest of the video. To remove such outliers, we detect if all the detected keys share a certain amount of its area with the hands. If not, then it is impossible that such key is pressed at that moment. We simply remove it.

IV. Results and Performance

Figure 10 show some of the frames annotated by our algorithm. We use a clip of Canon in D performed in the Youtube video [4], which is a fairly slow piece. We extract 690 frames (the video is recorded at 30 fps). As we can see, the middle keys are now being detected as a result of

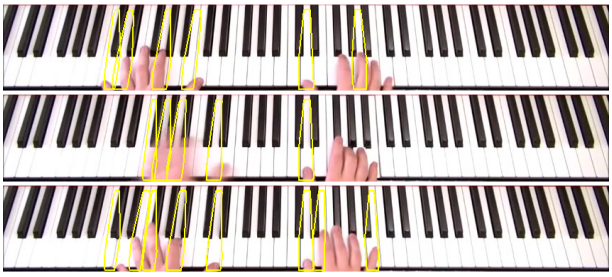
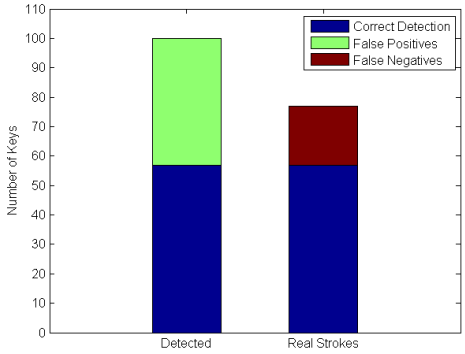
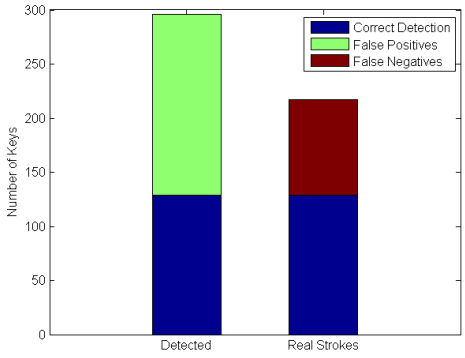


Fig. 10. A few results from our pipeline.



(a) Canon in D (slow)



(b) Let It Go (moderate)

Fig. 11. Numerical results. The blue bars represent the number of correct detections. The green bars represent the number of detected keys that are not actually being pressed (false positives). And the red bars represent the number of undetected pressed keys (false negatives).

our refinement. However, a few keys on the left hand are incorrectly detected as being pressed (false positives). This problem is due to the shadows of the hands affecting the image differences. Although we mask the hands off and even dilate the mask to include some areas around the hands, we cannot clean all of the shadows. As we can see in the middle image above, the left hand swiftly changes position, creating a blurred hand figure. Our hand detection can detect the large portion of it but a ring of shadows still remains, which results in multiple consecutive keys being pressed. On the other hand, some keystrokes are missed as in the G on the right hand in the middle image above. This is due to not enough changes are detected.

Figure 11 shows the number of keys that are detected as being pressed by our algorithm against the number of keys that is actually pressed. We collect the data from two input videos. One is the Canon in D [4], Figure 11(a), described above with 690 frames extracted and 77 keys being pressed in the entire clip. Our algorithm can correctly identify 57 of the pressed keys, leaving 20 false negatives. This gives a 74% recall. But our algorithm detects 43 false positives, i.e. it detects these keys as being pressed while they are actually not. This gives a 63.3% precision and a F1 score of 0.68. Our precision is low because of the reason described above. Hand shadows and swift movement interfere with our algorithm, causing changes that should not be there. Moreover, without precise calculation of the white key regions, we end up having too large white keys. These large areas sometimes detect the changes in other keys and presume to theirs. On the other hand, the recall is okay. We can further increase the recall by experimenting more with the adaptive threshold described in the Refinement section.

The results get worsen when we use the algorithm on a more moderate-pace piece. We use Disney's Let It Go [5], Figure 11(b), with 545 frames extracted and 217 keys being pressed in the entire clip. Our algorithm can correctly identify 129 of the pressed keys, leaving 88 false negatives. The recall now falls to 59.4%. The algorithm also detects 167 false positives, giving a precision of 43.6% and an F1 score of 0.50. Since the piece is faster, the problem exacerbates because the hand movement is faster. Moreover, the player sometimes does not press the key to its fullest depth so as to soften the sound. This interferes with our algorithm and makes it miss some keys.

V. Conclusion and Future Work

Clearly, the proposed algorithm does not perform well, although it does provide a promising starting point. There are many steps in our pipeline that can still be improved:

(1) The hand shadows are probably the main culprit in our low precision. To discard more keys, we need to design a new MAP detector that can detect the shadows and mask them off as well.
(2) The white keys detection as of now is not precise. As described in the section, using edge detection techniques or Harris corner detection might improve the performance.
(3) Experiment more with the adaptive threshold on different parts of the keyboard. The problem of keys near the middle being hard to detect can possibly be improved by tilting the camera setup a little.

We also have not tested the robustness of our algorithm against different illumination, scales, image quality, etc. We will leave that as a future work.

Another future work, and one of the main goal of this project, is to combined this pipeline with finger detection techniques such as detailed in [2]. If we can incorporate the locations of the fingers, we might be able to reduce the number of false positives in our algorithm. Perhaps in the future, we might be able to use only videos of piano playing and automatically produce the sheet music. Such tool will be very helpful for online learners who learn playing piano by watching video online.

Acknowledgment

The author would like to thank Professor Bernd Girod of Stanford University's Department of Electrical Engineering for his outstanding teaching of the Image Processing class, including all the topics used in this project. The author also would like to thank mentor Matt Yu and the EE 368 teaching team for their guidance throughout the project.

References

[1] A. Oka and M. Hashimoto, *Marker-Less Piano Fingering Recognition using Sequential Depth Images.* The 19th Korea-Japan Joint Workshop on Frontiers of Computer Vision, pp. 1-4, Jan 30-Feb 1, 2013.

[2] O. D. Gorodnichy and A. Yogeswaran, *Detection and tracking of pianist hands and fingers.* CRV '06 Proceedings of the 3th Canadian Conference on Computer and Robot Vision, pp. 63, 2006.

[3] Y. Takegawa, T. Terada, and S. Nishio, *Design and Implementation of a Real-Time Fingering Detection System for Piano Performance.* International Computer Music Conference Proceedings, vol. 2006, 2006.

[4] https://www.youtube.com/watch?v=wny1ojJli8k

[5] https://www.youtube.com/watch?v=q0yVJUuSZ10