**Virtual Piano**

**Team: The DayDreamers**

Pranav Dhakras(20162303)

Sahil Chelaramani(20162051)

1. **Abstract**
2. **Problem Statement**
3. **Methods Employed**
4. **Results**
5. **Future Work**
6. **Conclusion**
7. **References**

**Abstract**

**The virtual piano is an interface that allows users to simulate a musical instrument by printing a template on a sheet of paper. The user then ’plays’ the virtual instrument as if it were a real one. The user records this with a video camera, feeds the video to the application, and appropriate music is generated by the application. This is achieved by calibration of the template, detecting the position of fingers, identifying the hit position and playing notes corresponding to that position.**

**Previous work & Challenges**

Suteparuk[3] leverages the change in activity maps caused by the depression of actual piano keys to detect keypresses. The problem with this approach is that in the case of a virtual piano, we have no key depression to leverage to detect keypresses. This scenario is much harder to detect as touches to the paper could be for a few microseconds and hence our system needs to work in real time. Rastogi and Joshi explored the idea of virtual musical instruments for multiple instruments with intriguing results[1]. They use a marker and detect the position of the marker tip to achieve a keypress. The problem with that is that a. We are restricted to the number of markers one person can hold to play b. The learning of playing this virtual instrument does not contribute to the case where one plays an actual piano. One idea that is explored in this paper is playing the instrument using fingers instead of using a specialized marker. Our work focuses on improving the constraints of this system by allowing users to play virtual piano without any form of specialized markers, with their bare hands.

**High level flowchart**

The high level tasks for this project include:

1. Get a calibration template of the piano
2. Detect the black and white keys of the piano
3. Create a logical map of these keys to the musical notes
4. Track user’s hands and detect the keypresses and their locations
5. Play musical notes mapped to the keys being played.

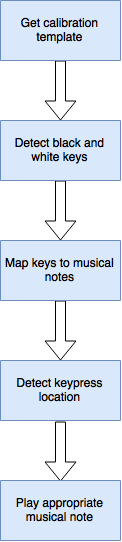


Figure : High level flowchart

Let us look at the different methods applied to try and achieve the mentioned tasks.

**Methods Employed**

**1. Calibration template**

To detect the keypress we first need to correctly locate the keys of the piano and store their positions for future reference. For doing this, we first require an occlusion free video frame with the paper piano in foreground. The assumption is that piano’s position in the video should remain static. Thus, neither should the paper piano movie nor should the viewpoint of the camera change anytime during the footage. We extract the calibration template by simply picking the first frame of the video(assuming that the user. We then pass this video frame to the key detection engine.

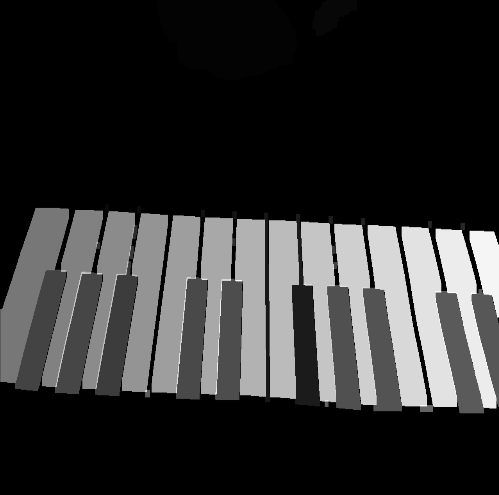


Figure : Sample keymap



Figure : Sample Keymap

**2. Key detection and mapping to musical notes**

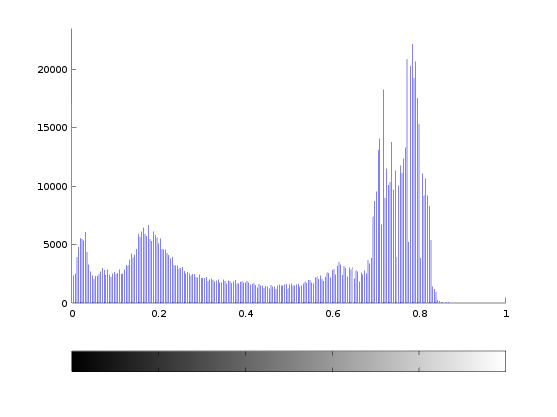
Once we have the calibration frame we proceed to locating the black and white keys from the template and storing their positions. For this purpose we use extended Otsu’s method for thresholding which returns multiple thresholds as specified. We have used two thresholds for this purpose, the lower threshold indication the maximum gray level value for black keys while the higher threshold indicates the minimum gray level value for white keys (and paper). Thus, of the three segments that by these two thresholds create from the histogram, the first (low gray level values) segment represents the black keys, the third segment, comprising of high gray level values, represents the white keys and piano sheet while the second segment includes the relatively less important components of the frame which we eliminate by setting those pixels’ gray level values to zero. We follow this up by performing a connected component analysis. This returns us all the black and white keys as separate components. Additionally, we also get, as the largest component, the sheet of paper on which the piano is printed. We eliminate the unwanted sections of sheet of paper by setting its pixels to zero. This is done easily since it is the single largest component from the thresholded frame. This leaves us with with only the black and white keys as components. We then assign a unique gray level value to each of these components (0-100 for black keys and 100-255 for white keys). This completes our task of detecting keys and we store this result as a special keymap reference frame which is used later.

Figure : Histogram of intensity channel. The three distinct peaks represent black regions, background regions and white regions.

The only remaining part is to map each key with a musical note. Since we had already assigned a unique gray level to each key, we can do a simple one-to-one mapping from gray level to a musical note. In case we encounter more keys than notes we assign musical notes in a cyclic fashion to the extra keys.

**3. Automatic Homography from Feature Points**

The idea behind this method is that we want to detect keypresses in a given frame. If we can detect a keypress with a high degree of accuracy in the original frame, we could remap the keypress onto a piano template using an Affine transformation, to get which key has been pressed. We start off by selecting keypoints from the given frame and trying to match them to key points of the template. We do this by using SIFT keypoints, and use the KD-Tree algorithm to match the feature points[12]. After finding the best matches between our template and the given frame, we use a subset of matched points (say 20 best matches) to compute what is known as the homography matrix which is just an affine matrix in this case which gives us the translation from our frame to the template.

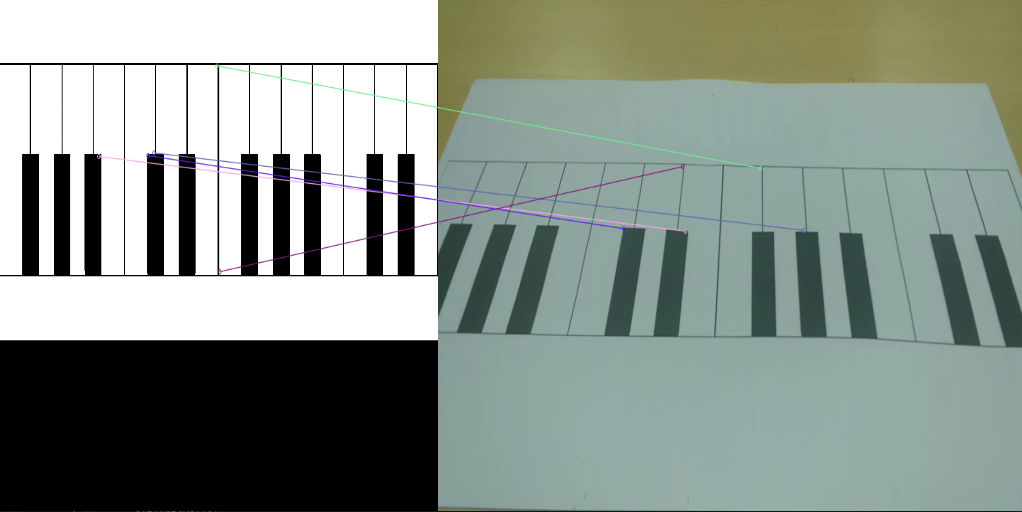


Figure : Homography for video frame and original template

As we can see in the figures shown, we have drawn a line to show the matched keypoints in the template to the frame. The keypoints are not being matched correctly. This is because of the inherent symmetry in present in the each of the key points. Due to the noisy matching, the homography matrix computed becomes noisy and results in suboptimal results. This method was abandoned due to the same reason. In the upcoming sections, we attempt to solve this problem from a different angle.

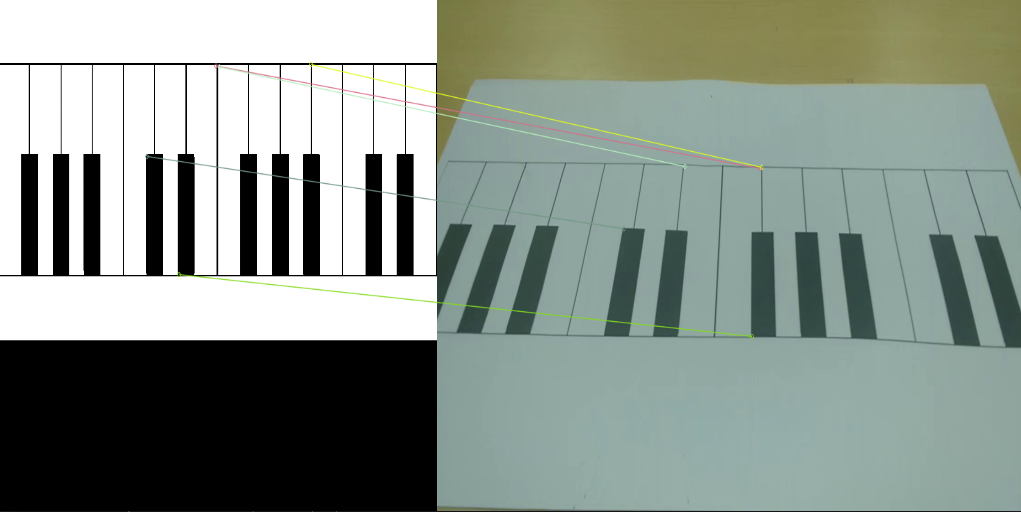


Figure : Homography for video frame and original template

**4. Contour Fitting for Skin Color**

One approach that we tried was to extract the boundary of hand and subsequently detect the fingertips of the hand. For this purpose we applied threshold in HSV colour space to match the colour of the hand and generate a mask containing only the hand. This was followed by finding out the contours of shapes present in this mask and empirically marking the two largest contours as the contours of the user’s hands. Later we found out the convex hull of these contours. From each convex hull we extracted the point with highest ‘y’ coordinate. It can be observed this point will correspond to the a fingertip. Knowing the location of the fingertip we can now detect which key is being pressed by that finger by finding out with key it occludes the most. We also keep track of the y-coordinate of the point and update a highest y-point only when the change between the current and previous highest y-point is significantly large.



Figure : Mask of hand

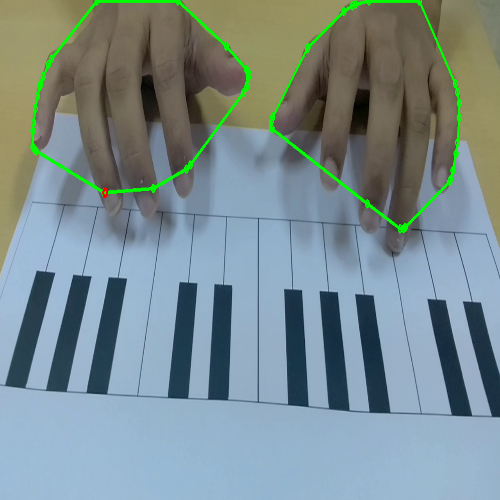


Figure : Convex hulls for hands and the corresponding highest y-points

The above procedure would work only when the user uses only one finger of each hand to play the keys. It is clear that simply extracting the point with highest ‘y- coordinate from a convex hull of hand will fail miserably when the user presses multiple keys simultaneously. In addition, this method is very closely bound with the exact extraction of hand contour and thus extremely sensitive to factors such as user’s skin tone, whether the user has applied nail paint etc. Another major problem with this method is that it will detect a ‘keypress’’ even when the user is simply hovering his fingers over a key and not actually pressing them. The reasoning behind this flaw is that to detect a ’keypress’ we are simply finding out the occlusion of a key by a finger and not looking at any special interaction between the finger and the paper at the time of a true keypress. Thus, due to its major limitations we decided to abandon this approach and looked at other, more promising, methods.

**5. Detecting white keys**

Since we want to somehow capture the interaction between paper and the user’s fingertips we thought of applying Sobel filter on each frame and look out for interesting patterns if any. On doing this we realized that we get distinct crescent shaped response whenever a finger actually touches the a white key on the paper. The reason for this behavior may be that the shadow of the finger on the paper darkens as it moves closer to the paper and when the finger touches the white key this shadow is darkest, resulting in a sharp difference of intensities between the white keys and the shadow. We decided to exploit this behaviour for detecting when a finger touches the piano. It is important to note that this works only for white keys and a different approach is required to detect black keys.



Figure : Sobel operator applied for white key detection

The Sobel operator response turns out to be noisy and we need to eliminate the gradient response seen for fingers and keys. It was observed that the gradient of a keypress was larger in magnitude than many other parts of the image. One improvement was to apply thresholding on the Sobel response map smartly by dynamically changing the threshold such that all values less than that of the keypress gradient magnitude are eliminated.

**6. Activity Map**

The problem with the above two methods is that they produce noisy results, which capture not only the color difference of the fingers v/s the paper, but also the difference between the every other combination of colors in the frame, which depending on illumination and other factors may have a stronger intensity than the finger and the paper. Now we make the assumption that other than the movement of the hands, nothing else causes significant change in the intensity values within each frame. Now, we can use the concept of an activity map to eliminate any edge responses from the sobel operator that do not change across frames. The idea of the activity map is that we find the median of the last N frames, and if the difference between the pixel in consideration and the median value is above a threshold, we keep it, otherwise we zero it out. This method eliminates any pixels that have not moved across the frames, hence any static color changes from the sobel output or background get eliminated.

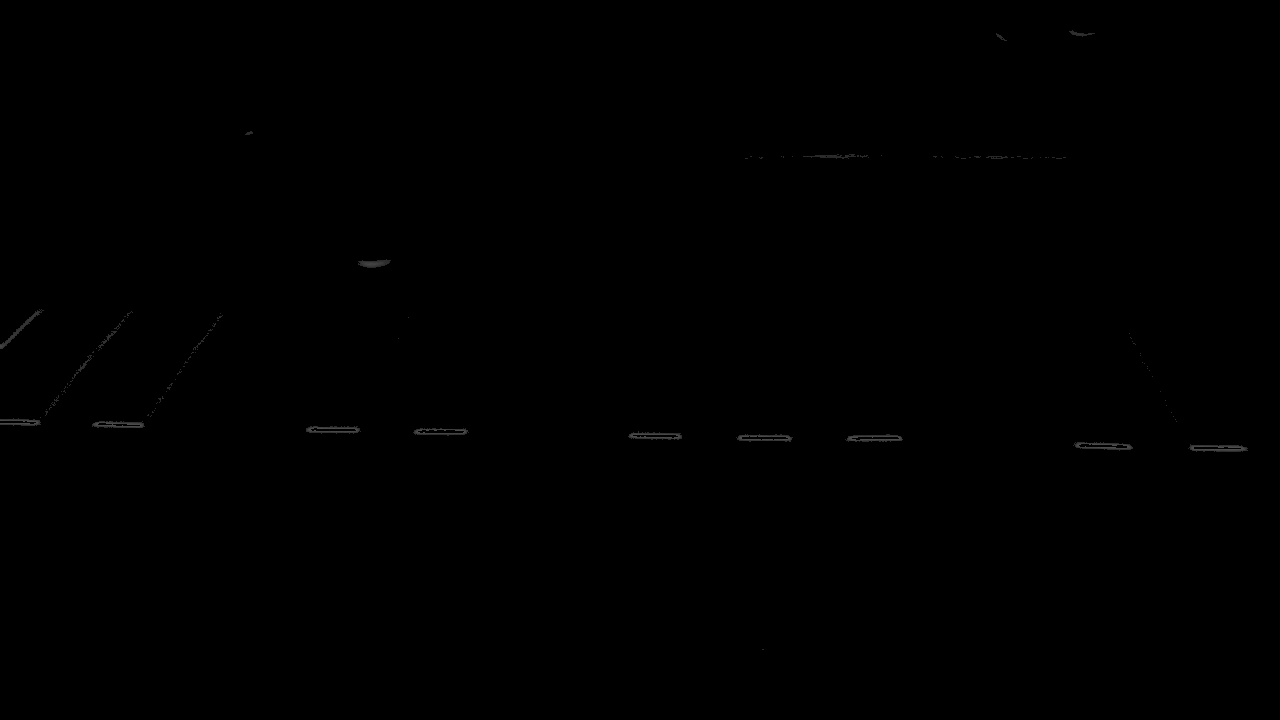


Figure : White key detection after applying activity map and smart thresholding

**7. Black key detection**

For detecting black keys we applied activity map for the saturation channel of the image. It was observed that the activity map for this channel mostly remains unchanged except when the user’s fingers move. It was also observed that when the user presses a black key there is a sudden change in intensities of the activity map for the region surrounding the keypress. We have leveraged this observation and combined it without smart thresholding as well as Sobel operator to detect when and where black keys are being pressed.



Figure : Activity map for saturation channel



Figure : Sobel operator applied over the above activity map

**8. Filtering out duplicate key presses from adjacent frames**

One problem of using activity map for detecting keypresses is that it tends to duplicate key press to adjacent frames in the video. The reason for this anomaly is that activity maps have a context of multiple frames and compare the difference for all frames within this context. Thus, an intensity difference corresponding to a keypress is propagated to nearby frames and the same press is detected in multiple frames. To eliminate this anomaly we maintained a queue structure which holds a recent history (for N frames) of keys pressed. A keypress is considered valid only if it is not already present in the queue. Otherwise the key press is ignored as duplicate. For every frame, if a valid keypress is detected it is added into the queue while the oldest keypress is removed from the queue. This technique filters out many duplicate keypresses detected by the original algorithm.

**Results**

The results for the experiment are as shown below. Our system successfully identifies not only single touch cases, but also multi touch cases as well. The white key presses have been highlighted in blue, while the black key presses have been highlighted in red.

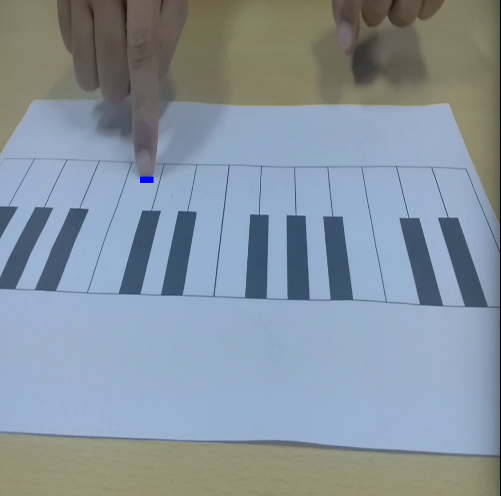


Figure : White key pressed

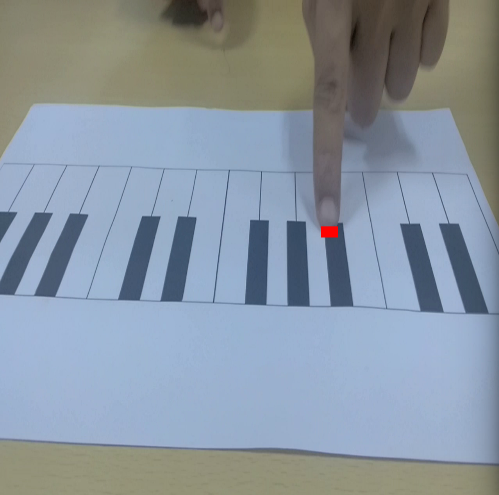


Figure : Black key pressed

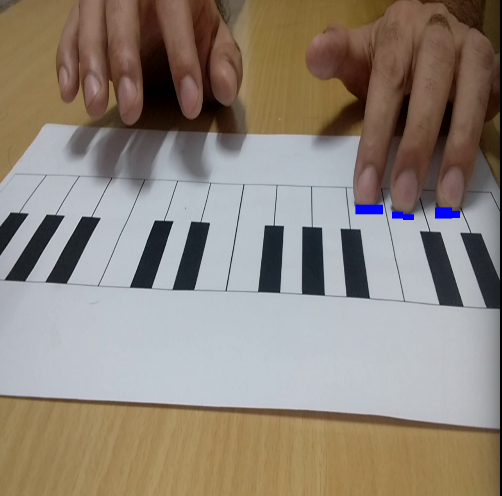


Figure : Multiple white keys pressed

The following are couple of failure cases that our system either missed due to the touches being too small and the paper being moved respectively.



Figure : False negative

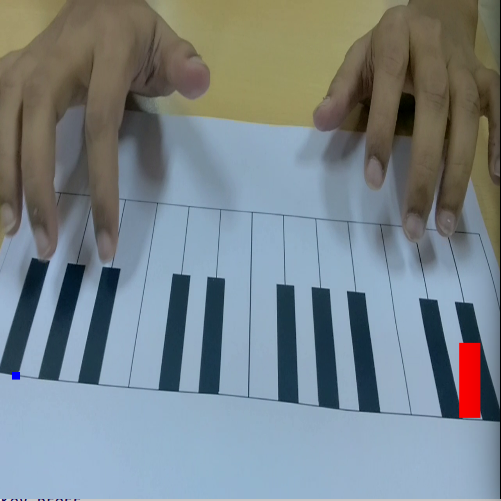


Figure : False positives

**Future Work**

1. **Skin Detection**

One of the biggest problems with earliest method for contour fitting was that the HSV color range used to extract skin color was not good enough to handle all skin tones. Imagine moving from country to country, the variance in skin tone is high, and hence selecting a range of color, is an intractable problem. Hence, we attempted to use machine learning to learn the skin tones. UCI has a skin segmentation dataset[7], and we use this dataset as an initial seed to learn skin tones using a decision tree classifier. Additionally, we propose a bootstrapping technique to improve skin selection. This method works by using our initially trained decision tree classifier to select pixels in the frame that it thinks are skin pixels and gives us a mask for the same. We follow this by running multiple morphological closing operations on the mask. The newly added pixels now are also possibly skin colored, and hence are added back to the training set and the classifier is retrained with the new set of pixels. We do this process for a multitude of hand-tones, and hence get a much richer dataset for skin tone selection.



Figure : Skin colour detection using Decision Trees

**2. RGB to Depth predictions**

The intuition behind this approach was that to establish the touch point accurately, we need depth information[8]. The problem with this approach is that we need depth sensors which are expensive and too much effort for users to calibrate correctly. To make this approach more adaptable by the masses, we turn to work done in Deep learning to predict depth maps from a single RGB image[9]. We use this work in conjunction with the hand pose dataset[6] to try and predict depth maps from single RGB cameras. This method would allow mobile devices to accurately detect keypresses because, the depth of the extracted finger and the table would be the same. We use a 5-layer deep convolutional neural network with alternating convolution and max pool layers, and rectified linear activation units. We feed this network our RGB images and try and predict the given depth maps. We use ADAM[10] to optimize our loss function.



Figure : Depth map computed using neural network



As can be clearly seen, the depth maps generated are very blurred, and hence can’t be used.

**3. Predicting key-presses using CNN**

We also try a method of breaking the image into multiple non-overlapping windows, and feeding each segment into a convolutional network. This network simply tries to predict whether a press occurs in this segment. If a press does occur then we assume that the press occurs at the center of the window. To actually build such a system, we need an immense amount of data. For this task, we take a majority of 10 videos of average 20 seconds each at 32 FPS. Next we manually annotate these pictures for places where presses occur, and store them in an index with frame to keypress coordinates. This being a relatively small dataset, we first flip each of the obtained frames, and double the size of our dataset(Adjusting obviously for the flipping about y-axis by computing the new affine coordinates of the press). To make our system invariant to lighting conditions, We use openCV’s implementation of Robertson’s[11] tone mapping operator to simulate various lighting conditions for the frames. We select 4 different exposure times, to simulate the low, lower-mid, mid, and high ranges of illumination. We use a CNN with a similar architecture as discussed in the last section, except that we have a fully connected layer at the end of the deep network to predict whether a press happened in the window or not. We use a 50x50 window size.

**Conclusion**

We present a system that is extremely efficient since it uses simple digital image processing constructs to detect keypresses. Additionally, our system has been built with the consideration for allowing users to learn playing the piano without having to actually buy the instrument, hence minimizing the cost barrier for learning musical instruments. This is accomplished in our system, by letting users play the piano with multiple presses in complex chord shapes. It can be shown that the technique learned on the virtual piano is directly transferable to cases where users move on to actually playing the piano. In addition to this, adding a feedback mechanism to improve learning can easily be achieved to improve the learning experience for users without having an explicit need for tutor. This becomes formative for beginners who need special mentoring to get good at playing the piano.

**References**

[1] “Virtual Musical Instruments” <http://web.stanford.edu/class/ee368/Project_Spring_1415/Reports/Rastogi_Joshi.pdf>

[2] “Digital Waveguide Architectures for Virtual Musical Instruments” - Julius O. Smith III <https://ccrma.stanford.edu/~jos/asahb04/asahb04.pdf>

[3] “Detection of Piano Keys Pressed in Video” - Potcharapol Suteparuk

[4] “Augmented Piano Reality” - Ihab Zaqout1 , Samar Elhissi2 , Aya Jarour3 and Heba Elowini4 <http://www.sersc.org/journals/IJHIT/vol8_no10_2015/13.pdf>

[5] “Detection and tracking of pianist hands and fingers” - Dmitry O. Gorodnichy1 and Arjun Yogeswaran2 <http://www.videorecognition.com/doc/publications/2001-2007/06-crv-see-pianist-last-0.pdf>

[6] “NYU Hand pose Dataset” <http://cims.nyu.edu/~tompson/NYU_Hand_Pose_Dataset.htm>

[7] “UCI Skin Segementation”

[https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation#](https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation)

[8] Barehanded Music: Real-time Hand Interaction for Virtual Piano - Hui Liang, Yong-Jin Li, Jun Luo, Ying He

[9] Depth Map Prediction from a Single Image using a Multi-Scale Deep Network - David Eigen, Christian Puhrsch, Rob Fergus <https://papers.nips.cc/paper/5539-depth-map-prediction-from-a-single-image-using-a-multi-scale-deep-network.pdf>

[10] A method for Stochastic optimization - <https://arxiv.org/pdf/1412.6980.pdf>

[11] “Estimation-theoretic approach to dynamic range enhancement using multiple exposures,” M. Robertson, S. Borman, and R. Stevenson,

[12] “FAST APPROXIMATE NEAREST NEIGHBORS WITH AUTOMATIC ALGORITHM CONFIGURATION” - Marius Muja, David G. Lowe