Virtual Musical Instruments with Python and OpenCV

Isaac Abraham Thottathil[1], S. Thivaharan[2]

[1]UG Scholar, Department of Computer Science and Engineering, PSG Institute of Technology and Applied Research, India

[2]Assistant Professor, Department of Computer Science and Engineering, PSG Institute of Technology and Applied Research, India.

E-mail: [1]21z119@psgitech.ac.in, [2]thivahar@psgitech.ac.in

Abstract

There is an increasing need for musical aspirants to have access to cheaper musical instruments. This study explores the opportunities to utilize image recognition algorithms via OpenCV to port this technology into readily available modern devices, which will enable inexpensive yet authentic methods of playing a piano. Through OpenCV and Pygame libraries, one can set up a rigid camera that will trace the player's fingers. The fingers if they cross or hover over a specific coordinate of a key, the piano note (.wav file) will be played by Pygame's mixer module. This simple yet inexpensive option might help first-time musical aspirants experience music in an affordable and accessible way. Furthermore, this article explores the future scope of accommodating other musical instruments.

Keywords: Social media analysis, decision-making, computational intelligence, machine learning.

1. Introduction

Music is an expression of art that is evolving with technology. There is a growing technology that deals with developing affordable and portable devices. However, beginners in musical endeavors are hesitant to purchase expensive instruments. Hence, this article utilizes the available technologies to improve musical instruments and make it readily available to musical aspirants, whilst aiming to give the same experience while playing a live musical instrument.

DOI: https://doi.org/10.36548/jucct.2023.1.001
Received: 27.12.2022, received in revised form: 27.1.2023, accepted: 13.2.2023, published: 23.2.2023
© 2023 Inventive Research Organization. This is an open access article under the Creative Commons Attribution-NonCommercial International (CC BY-NC 4.0) License

In India, approximately 40% of the Indian population belongs to the middle-class economy and the burgeoning cost of musical instruments can force young children to despise their interest in playing musical instruments. A vast majority of rural schools have no basic infrastructure to accommodate such growing interests. Playing a Keyboard, Piano, Guitar might be a distant dream and families find it difficult to afford this skill. However, technology has offered us keyboards and virtual instruments that are restricted in small smartphones, but fail to promise us the authenticity of playing the instruments in reality. The paper provides a virtual piano that can detect fingers via finger cap (using HSV recognition) and by approximating the fingers to the 3D space it is located in, the Pygame runs the corresponding wav file of the key.

2. Literature Survey

This research work has attempted to create a virtual piano by utilizing Augmented Reality (AR) and image recognition technology. With AR technology, several research work have developed fingerings to generate data and predict data in real time with the help of a HMM Model to decide the note currently being played.[30] In order to simulate a real-life model, a 3D rendering of the current position of the fingerings is displayed to increase the calibration.

Some techniques have attempted to achieve depth sensing data or the magnitude of how much the fingers have been raised, to play a specific note. Pre-developed software like Leap Motion has been utilized to provide data feed of the finger movement which in turn is processed to play musical notes [5] [17][18][31].

By capitalizing on gesture detection, by differentiating finger from palms, using convex hull algorithms or by using a CNN various hand movements can be classified.[22][24] Proposed at segmenting images before usage or further detections, papers [26][27][28] have pointed to the use of techniques that are not same as traditional Gaussian Blur, that would create noise. They also call out for better edge detection techniques in data preprocessing. To improve the data fed to the algorithm for playing a note, several attempts [23][25] have been aimed at creating object tracking either via comprehensive models or through OpenCV and HSV detection. This can be further fed as input to the program for playing a note.

In another instance to subtract skin-based subtraction from background [29], the camera feed is converted to “YCrCb” scheme instead of RGB format. To isolate a moving hand, it

first identifies moving pixels, and then checks if the moving pixels fall under aYCrCb threshold. Though it is appropriate for moving objects but cannot be effective for still cases.

As a piano is being played simultaneously it requires to be run on lightweight applications. Several versions have looked into utilizing the full potential of Computer Vision techniques. In attempts of utilizing Computer Vision techniques [4], there is a use case of creating a replica of a Piano image. This piano image is applied as the prototype or the base image, on which our fingers when placed can be identified. Since one's fingers covers a note, that finger is verified by masking and then the note covered is played. This technique uses masking of the finger color, to identify the note being currently played or the note that is being currently hidden. As laid out, this model is subject to insufficient lighting that can cause improper note detection. Moreover, this model is based on the base image or the prototype, which will have to be restricted or be predetermined by the developer.

This is why there is a need to have a base image that can be of any size and structure with respect to one's camera's field of view. This paper shows a plausible means of finding a finger from a live camera feed by utilizing the masking technique and it is independent of the base image/prototype. This paper tries to achieve an algorithm that can run on any user generated base image. In this paper, freedom of selecting the base piano is provided to the user. By manually affixing the anchor points, one can create a virtual base image for their template piano, and are free to hover over their preferred base image/templates.

Hence, by combining techniques to improve finger detection by HSV on a lightweight application, while still providing flexibility to user for an easier template, this paper tries to achieve the various complexity of a virtual piano.

3. Proposed Work

The paper divides the Python program into different modules that solve different tasks. The following shows the modular division of the program where piano.py acts as the entry point with a main function. It imports built-in modules like Pygame, OpenCV and Numpy. Pygame is a platform to play notes and show a sample of the keyboard. OpenCV uses the module to enable HSV recognition.

Numpy is the medium for data processing of each frame of the image.

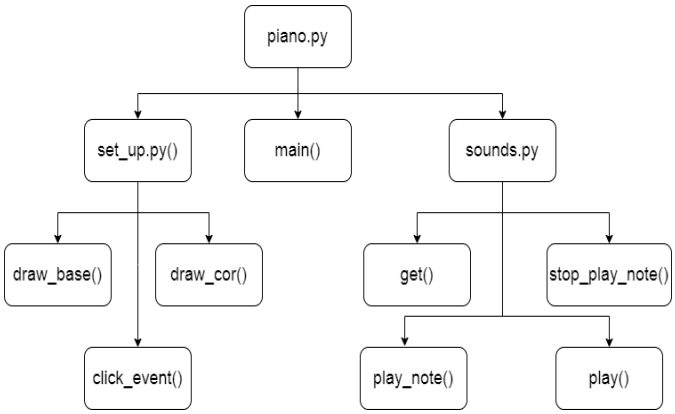

Figure 1. Module Division

3.1 Main Module - piano.py

The piano.py imports two files called as set_up.py and sounds.py. Moreover, piano.py contains the *main()* function. Built-in Modules like OpenCV, Pygame and Numpy are imported. The *main()* function runs a while loop indefinitely to get the frames from the camera via OpenCV. It calls the various functions under sounds.py and set_up.py. The reason it runs an indefinite loop is so as to capture the frames from the camera again and again.

The *main()* is responsible for finger detection via HSV recognition and if the finger of the prescribed contour size is chosen, then play() function in sounds.py is fired. The *main()* function under its indefinite while loop, which runs till camera stops feed, also calls the *draw_base(), draw_cor(), click_event()* functions under the set_up.py module.

3.2 Module to Play Notes- sound.py

The sound.py module has all the essential methods and algorithms to run the corresponding wav file of each key, when the key is detected by the *main()* function. The reason to choose wav files (Waveform Audio Files) is because they retain quality during compression and if these files are chosen losses in quality will be minimal. To expand this project via multithreading to play various files in many channels, compression is essential, and combining the waveforms are common. Hence, in applications of expanding this paper the wav files remain suitable to scale up.

This module makes sure to load all Sound objects or the wav files of all the Piano notes from C1 scale to C6 Scale. Moreover, it has *play()* function which receives the note as input and selects them from loaded Sound Objects and passes it to *play_note()*. If another note is being played it calls the *stop_play_note().*

The *play_note()* function ensures that mixed sound of the *Pygame.mixer* property is free and allows loading of wav files. The get() function is called initially whenever play() is called. It is to ensure that the global constants are set. These global variables are 7 integers (representing C,D,E,F,G,A,B) that change value to 1 when the corresponding key is being played.

3.3 Module to For Initial Setup of the Blueprint- set_up.py

The set_up.py has three functions that are called under *main()*. The *draw_base()* sets up the *Pygame* frame with a Keyboard. If a note has been detected, over the detected note there will be a circle to reassure the user that the correct note is detected.

The *draw_cor()* is a function that draws out the x and y coordinates of the blueprint onto the camera. This draws the coordinates indefinitely over the OpenCV, after a new frame is received.

The *click_event()* handles all the left and right clicks on the OpenCV window. A left click can set up a new end point of a key. The user will have a blueprint over which he will play the piano. A camera opposite to the user's hand, tracks the movement of his fingers over the blueprint. The blueprint contains all the keys of the pianos.

The vertex of each of the keys when seen from the camera is added using the left click. By right clicking all the previous vertices are removed and can be added from start. This refresh will be essential in perfectly pinpointing the vertices.

4. Setting Up the Device

To ease the mobility of studying a Piano/Keyboard with an authentic experience and providing an inexpensive option, a Piano is created that uses the help of a desktop or Smartphone affixed with a Camera. This innovation has been developed entirely on a Desktop with the help of an old camera that has basic colour recognition hardware specifications. An A4 Sheet or a Paper is laid out on a Table. This acts as a blueprint for the player to move their fingers on a set of keys. Laying the paper in landscape, the C Scale of Piano is drawn (the White Keys of the Keyboard) by spacing each key in equal intervals onto the paper.

Place a Camera opposite your paper/blueprint, such that it is facing towards you: making it easy to record your hand movements. Connect the Camera with a USB Cable to a Desktop that has the developed Python file installed.

In this paper, to attain a higher image and eased finger recognition, we have used bright colourful finger caps made with help of colorful sheets (that covers your nails), as the Camera will be able to easily detect your finger gestures. Users can determine the colour of the finger cap they would want to use and change the HSV value range to an appropriate level[21][22][23]. Users can also restrict setting up the device with skin recognition by utilizing the HSV range as follows: Hue Range (0-20), Saturation Range (30-249), and Value Range (60-255).[14][15][16].

This can vary based on the lighting as well as the camera's hardware specifications. However, it is essential to fine tune these ranges to optimize recognition of fingers without shadows. [19][27]. Configure the program by pointing out where each key of the C scale lies in your camera by right clicking at the desired location. By analyzing where your key starts and ends, find the starting and ending point of each key of a scale (C,D,E,F,G,A) on the camera. And then click its coordinates onto the OpenCV window. This will allow the program to predetermine where the keys are placed with respect to the user's paper (blueprint) and user's space. Fig.2 shows a blueprint that is stuck onto a table. This blueprint (an A4 sheet) has the keys marked for a single scale. This is the blueprint onto which the user strikes their hand, which will be recognized by an image opposite to him. To define each key in the 2D frame, the red points will be referred to as vertices of the Piano Keys in this paper. These are the points on the blueprint that are being marked onto the OpenCV window, and these can also be called as anchors for this piano.

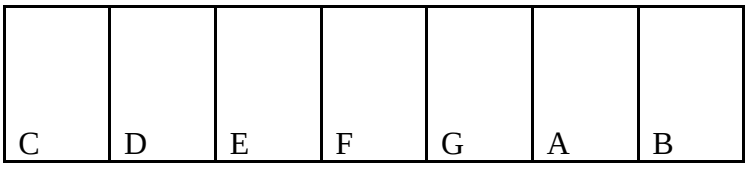

Figure 2. Blueprint of the Keyboard

This blueprint acts as the reference points for further image detection. Increasing the width of the blueprint and increasing the field of view of the camera, can aid in playing multiple scales at a single time. Due to using only a single scale, the blueprint has the general notes, without its mention of the keys.

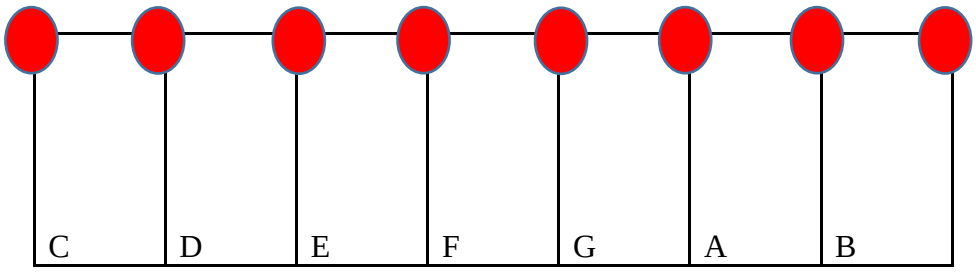

Figure 3. Blueprint of the Keyboard with Vertices (Red in Colour)

In Figure 4, the left window is a *Pygame* window that shows a keyboard over which a note will be struck if a finger is detected. This *Pygame* window is made to reflect the blueprint. Camera View, on the right, shows a view opposite to that of where the paper is placed. The user pinpoints the start and end of the keys (vertices of each key) to mark it onto the Camera Window (OpenCV window).

In Figure 4, the user is seen using a pen to locate his vertices in real life and mark it onto the OpenCV window by a left click. This camera angle requires this process. For various other musical instruments, various angles and differences in defining anchor points are the unique differences among various instruments.

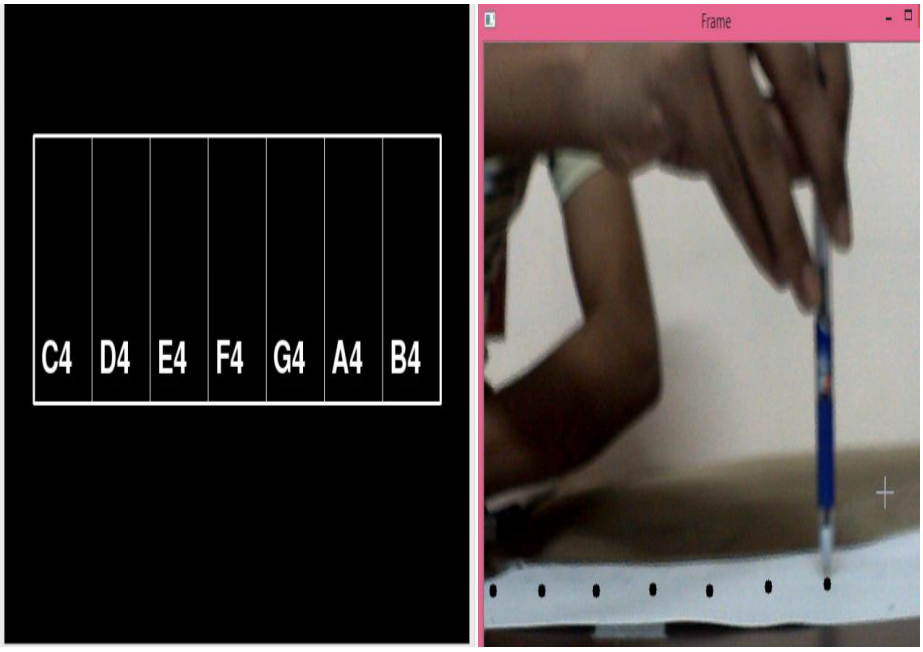

Figure 4. Setting up the Device with Vertices

5. Algorithm and Working

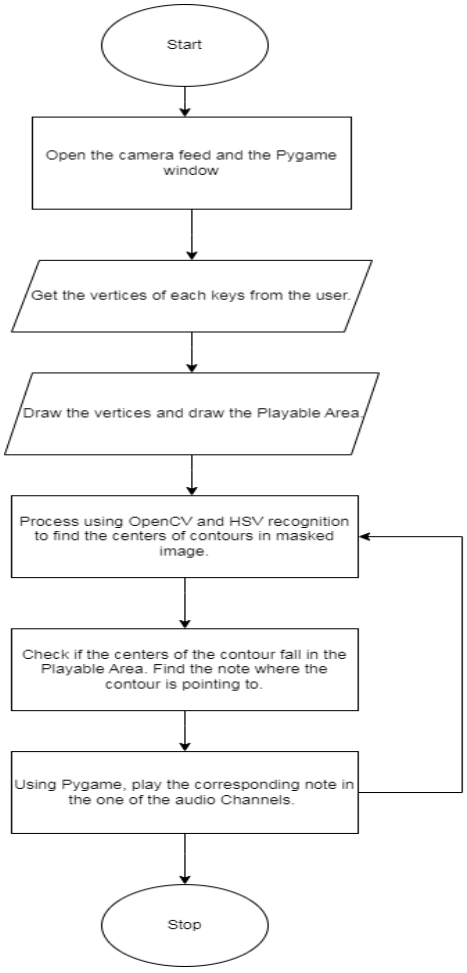

Figure 5. Simplified Flow of the Entire Program

The algorithmic flow between the modules has been simplified to a flowchart as in Fig 5. The major indefinite while loop runs till no frame is being returned. This can be seen in the jump from the step that deals with playing the wav files to the step that deals with processing the next frame. This jump runs over and over as it processes each of the frames. Despite this visualization, these processes run in the background. The playing of a wav file runs continuously without any disturbance when reading and processing a frame for contours. This is an example wherein the built-in packages efficiently take care of the running of wav files automatically on the audio channels. Hence, this representation is the flow of how the code is written, but its execution is not limited to it.

5.1 Initialization

The algorithm to set up the 2 windows, a *Pygame* window and an OpenCV window is handled by the module set_up.py. This has functions like *draw_base(), draw_cor* and

click_event(). All these functions are called during each iteration of the while loop in *main().* They are being called to draw an image onto the 2 windows, and handle functions in these windows. The *draw_base()* will be called at the start of the indefinite loop, and will be used to set up the Python Frame. Then a polygon is drawn, with several lines to create a sample keyboard [12].

```
screen=pygame.display.set_mode((640, 480))
```

The screen object is onto which the *blit()* and draw methods are being called. The *blit()* function writes down the key name for each of the Piano Key and its values change when moving up and down a scale. The OpenCV window is initialized globally under the main.py and onto that window, in each iteration the points are being drawn. The *click_event(event,x,y,flags,parm)* takes care of handling events in the OpenCV window. These handled events include left and right click. During the initial setup, it is essential for the user to define his space or region, wherein the keys are placed with respect to his camera. The x and y received as parameters by *click_event()* are the coordinates on the OpenCV window.

```
if event==cv.EVENT_LBUTTONDOWN:
ifnumth==0:
        p1x,p1y=x,y
numth+=1
elifnumth==1:
        p2x,p2y=x,y
numth+=1
```

The sample code shows the if clause handling 2 vertices, wherein if the user pinpoints their cursor to the appropriate vertex of a key, and then left clicks, the above code will be executed. The variables *numth, p1x, p1y, p2x, p2y* are global and they are being manipulated on a left click. If the first vertex is clicked, then p1x, p1y are assigned values of x and y and the value of *numth* is incremented. Hence on the next left click, the if clause modifies the p2x and p2y values. Since the *draw_cor()* function is being called along with this function, the defined points are marked onto the OpenCV window. This idea can be extrapolated to points 8 to mark the 8 vertices on a single scale. This same function also handles the EVENT_RBUTTONDOWN, which resets all the coordinates of points 1 to 8 and sets *numth* to 0. This provision is enabled to allow users to correct their mistakes. The *draw_cor(frame)* is being called along with *click_event()*. This function helps us visualize these vertices onto the OpenCV window. This is done with the help of *cv.cricle()* method. This function redraws all of the squares onto the frame supplied by the camera. The parameter frame refers to the new frame received from the camera input from main.

5.2 Finger Recognition

The main aim of the Image recognition program is to approximate when a color that suits the predetermined HSV [2][3] range crosses a threshold area.

The y coordinates of the points that decide the start and end of each key is averaged out and around 40 pixels above the average ordinates is set to the limit where finger recognition is possible. If a finger is noticed within the average ordinate and 40 pixels above the average ordinate, the corresponding note is played.

```
cam=cv.VideoCapture(0)
ret,frame1=cam.read()
```

This sets *cam* as the Video object that reads from the 0th Camera (usually the built-in webcam of the laptop). The *cam.read()* method returns ret, which is a Boolean that determines the validity of the loop. The loop or the program runs indefinitely till ret is True or till camera can get feed. Frame1 is an *NumPy* array that has an RGB value of each pixel of each frame. The number of frames is determined implicitly or explicitly. To enable proper HSV recognition and color recognition, a frame that consists of HSV values is created from frame1.

```
frame1_hsv=cv.cvtColor(frame1,cv.COLOR_BGR2HSV)
min_b=np.array([lh,ls,lv])
max_b=np.array([uh,us,uv])
mask=cv.inRange(frame1_hsv,min_b,max_b)
```

Here the mask variable consists of pixels that are set to 1 if the source image has pixels that are between the pixels of the *min_b* and *max_bnumpy* array. Their values are determined from the lower and upper limit HSV values of the finger or the finger cap of the user. The following formula is utilized to allocate masks within the given range [11].

$$dst(I) = \ lowerb(I)0 \ \leq src(I)0 \leq upperb(I)0 \wedge \ lowerb(I)1 \leq src(I)1 \leq \ upperb(I)1$$

Then the current frame is operated bitwise and with the mask created. The bitwise and makes sure that each pixel if it falls under the HSV range (as defined in the mask) will only allocate a pixel value. The HSV value in each pixel is converted to white (or value 1) while the rest falls to 0. This is enabled by converting the *thto* Grayscale and then converting each pixel value only to 0s and 1s (by *Thresh_Binary*) only if it falls under the given threshold.

```
th=cv.bitwise_and(frame1,frame1,mask=mask)
th=cv.cvtColor(th,cv.COLOR_BGR2GRAY)
th=cv.threshold(th,0,255,cv.THRESH_BINARY)
dilated=cv.dilate(th,kernel,iterations=3)
```

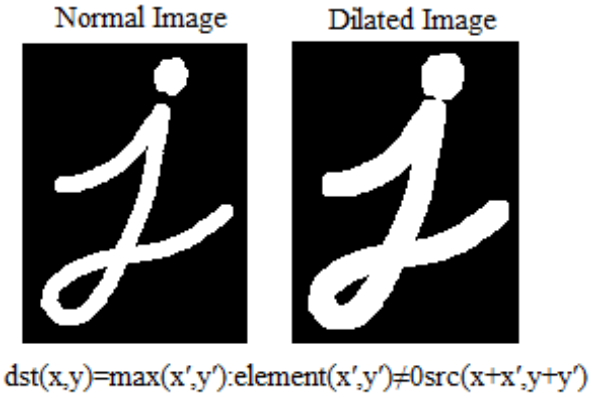

Figure 6. Dilation of an Image to enhance the masked regions before contour detection [8]

Moreover, to dilate the masked regions, a dilated function is iterated 3 times to enhance the regions that are already bright. From this dilated region, it is easier to find Contours or the keys over which the fingers hover. In a dilated image, the value of the output pixel in a is determined as the maximum value of all pixels in its neighborhood. The reason we convert it to a binary image, is because the pixels are set to 1 if any of the neighboring pixels have the value 1. Hence the masked regions will be visible and performing dilation on it creates a contour/shape. Morphological dilation fills in irregularities by covering small holes. This allows lines to appear thicker and contours to be filled [13]. Hence it is now easier to pinpoint and select the center of a large contour than divided up contours. However, before dilation noise removal is necessary which is evident in the Fig.7. Dilating a noisy frame that masked black objects, created a frame wherein the shadows are also dilated. This might prove a hurdle to HSV recognition. This is why to obtain better results, a background that is white in color and a high contrast color is to be used for a finger detection. This is why\ in this paper, a finger cap is being utilized. This will provide a contrast color to the white background, and since it is not black it doesn't fade in with the shadows [6][7].

In Figure 7, the left window shows an example of Black Color HSV recognition. The right frame is masked with appropriate HSV (for the range of black, that do not accept grey shadows). Then a *bitwise_and* is performed between them. The result is turned into Gray Scale, then Binary Gray Scale. This image has values that are either 1 or 0, and is easy for

contour detection. It is then dilated to produce the image. Finally contours/areas within the 2 red lines are identified and a green dot is added to show the start of the contours.

In Figure 7, the contour's center is not considered to be the point which acts as a reference for the contour. However, the contour is accessed by its border value that could be found in the contours list. This would seem an inefficient way for contour detection spanning a large region. Moreover, these edge points seem to oscillate much larger to external noise.

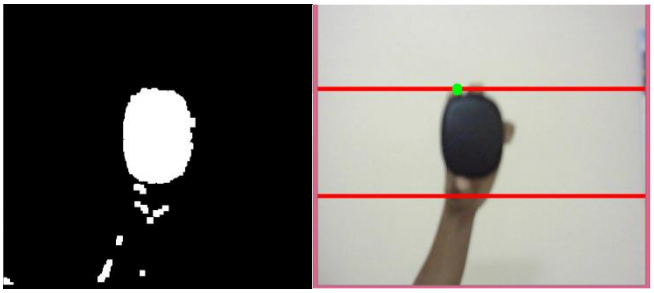

Figure 7. Contour Detection for a Black Object in a Playable Area

Contours [1][19] are boundaries of a shape that have the same intensity. Contours contain the x and y coordinates of the boundaries of the shape. However, to optimize space complexity, OpenCV removes redundant points and compresses the memory. This is taken care of by the parameter *cv.CHAIN_APPROX_SIMPLE*. The second parameter is called the contour Retrieval Mode, which is taken care of by the property *cv.RETR_TREE*. This ensures that the hierarchy list contains the hierarchy of parent contours and child contours [9]

```
contours,hierarchy=cv.findContours(th,cv.RETR_TREE,cv.CHAIN_APPROX_SIMPLE)
```

In figure 8, on the left, a white contour that is in the shape of a rectangle is found by *cv.CHAIN_APPROX_NONE*. This property returns a contour list that has over 729 points. On the other hand, the white contour on the right has just 4 points. OpenCV removes the rest of the redundant points with help of *cv.CHAIN_APPROX_SIMPLE*. This is how space efficiency is attained, and can aid in faster processing [10].

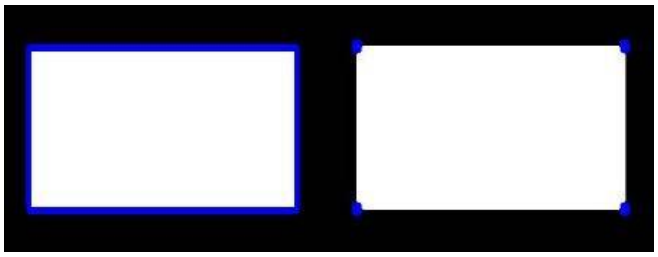

Figure 8. Setting edge Points for Contours in 2 ways [9]

```
for i in contours:
M=cv.moments(i)
if M['m00']!=0 and M[‘m00’]>100:
y=int(M['m01']/M['m00'])
if y>=avg_y-40 and y<=avg_y+10:
x=int(M['m10']/M['m00'])
cv.circle(frame1,(x,y),5,(0,250,0),-1)
play(x)
```

Furthermore, the next step is to iterate over the list of contours, find their areas and check if their areas cross a certain threshold, which is set as 100. If it is true, the y coordinate of the centroid of the contour is found using the relation M[‘m01]/M[‘m00]. If this y coordinate is within a certain region of the so called “Playable Area” in the frame, then the x coordinate of the centroid is found. The “Playable Area”, refers to the region where if the user moves his/her fingers will be recognized by OpenCV. If a user strikes their fingers on the upper region of the camera frame, it will not be accepted. The region of acceptance is defined to a “Playable Area”. The Playable Area is defined by finding the average of all the ordinates of the edges of the keys, and extending 40 pixels to the top and bottom from this average. This region is marked by two lines and it is within this region where the Camera tracks movement. After this, with the centroids of each contour, a circle is plotted which is Green in color. The abscissa of the centroid is passed to the function *play()*, where it is further processed to play a note. The reason for selecting the centroids of each of the moments is not for the reason of symmetry alone. But using the abscissa and ordinate of the center, in real runtime, avoids the fluctuation of the coordinates. The coordinates of the centroid chosen are more or less constant throughout the entire loop, compared to any coordinate on the contour boundary.

The formula below is applied to find the centroid of a contour or a moment.

$$C_x = \frac{M_{10}}{M_{00}} C_y = \frac{M_{01}}{M_{00}}$$

Accuracy and speed of the finger recognition algorithm is calculated when the fingers are either close or far from the anchor points. Speed is relatively same as they apply same techniques with little change. Variation in accuracy is proved due to critical changes in lighting conditions.

Notes Played	Accuracy	Speed
When centroid is in mid of the region (>5pixels from border)	98.15%	0.00708s

When centroid is near to borders (<5 pixels from border)	87.23%	0.00800s

5.3 Playing the Note

Pygame's Mixer Module is helpful for controlling playback and loading sound objects. The mixer module must be initialized at the beginning of the sounds.py module. The *pygame.mixer.init()* function is run as the first function and takes arguments to control the playback rate and sample size. This can be decided with respect to the user's hardware resources or audio resources. Once a user installs the wav files of some notes of the piano on their local machine, they will have to be created as sound objects.

```
c1=mixer.Sound('C1.wav')
d1=mixer.Sound('D1.wav')
e1=mixer.Sound('E1.wav')
f1=mixer.Sound('F1.wav')
g1=mixer.Sound('G1.wav')
a1=mixer.Sound('A1.wav')
b1=mixer.Sound('B1.wav')
```

The above code shows creating Sound objects with wav files of the keys of the first Scale. In this paper, this preliminary initialization is done for 6 scales, and each of the Sound objects are loaded into a list. This is also extrapolated to lists c, d, e, f, g, a, b.

```
c=[c1,c2,c3,c4,c5]
```

With help of a global integer *ind*, that keeps track of the current scale, the user will select a sound object from the list. The play(x) function has an if-else ladder to check if the abscissa of the centroid is between any one of the 7 keys (C, D,E,F,G,A,B) that was already predetermined by the user. If it falls between point 1 and point 2, the C note of the current scale will be played. The sample if clause for catching the first key (C key of any scale) is given below.

```
iftemp_X>=p1x and temp_X<p2x :
    play_note(c[ind])
```

The function play(x) copies the abscissa of where the finger is placed into a variable *temp_X*. Then the defined integers of p1x and p2x, all the way up to p8x are checked to see if the current finger is between them. This if-clause can be extrapolated to 8 points. To play the note wherein the current finger is placed, a *play_note()* user defined function is utilized. This function checks if the *Pygame* mixer, a module to load Sound objects, is currently running. If not, the current Sound object is played. The *get_busy()* method checks if the mixer is busy

mixing any channels. If the mixer is standing idle, the method returns False. This will prevent cutting a running wav file in the background abruptly.

```
ifmixer.get_busy()==False:
<Sound Object>.play(0)
```

Else, if the current mixer is already running, which means a previous note is currently playing. Then the mixer is stopped and the new Sound Object is played. The next block of code comes under the else block of the previous code. The *get_num_channels()* returns the total number of active playback channels. The <*Sound Object*> variable refers to the parameter received for the *play_note()* function. There is a specific reason why there is another if clause to avoid performing *themixer.stop()* if the active number of playback channels is 1. This is because when a frame is being played in OpenCV, these functions in sounds.py module are being called indefinitely. Hence when playing a note, if the note's corresponding wav file is already called, it is essential not to stop the wav file running, and to avoid restarting. Hence the program has to check if the current Sound Object is playing or not. If it is playing, it indicates that the user's hand is still over the key else, it shows that the key was never played in a recent set of frames.

```
ifmixer.Sound.get_num_channels(<Sound Object>)==1:
pass
else:
mixer.stop()
<Sound Object>.play(0)
```

In Figure 9, the finger when playing/hovering over the key E1(as seen for the user in blueprint), is being recognized as key E1 in the *Pygame* window. The note E1 is shown in the *Pygame* window. Moreover, the corresponding wav file of E1 is being played in the background. Notice the green circle on the center of the finger that is currently being played. This explicitly tells that the image pre-processing, HSV recognition and contour detection algorithms have all worked with zero errors, to point to the center of each moment.

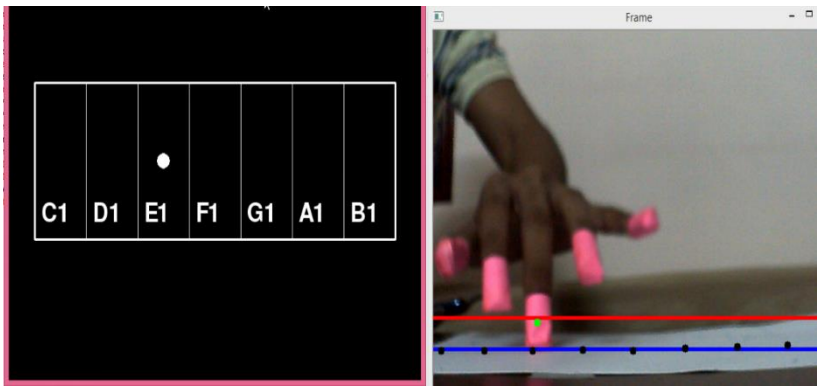

Figure 9. Setting edge Points for Contours in 2 ways [9]

As mentioned, to move across various scales the user will have to traverse the scales using the right and left keys. This event is being handled by *Pygame* in main, as it waits for events in frame.

```
forev in pygame.event.get():
ifpygame.KEYDOWN==ev.type:
ifev.key==pygame.K_RIGHT:
ind+=1
ifev.key==pygame.K_LEFT:
ind-=1
```

The *pygame.event.get()* method listens for a key in the keyboard to be in down state. If the corresponding event is the same as the right button being pressed down, then the global variable *indis* incremented. Similarly, the *indis* decremented if the event is a left button being pressed down. Incrementing *ind* will affect the values like *blit()* methods for the *Pygame* Screen and the different sound objects will be accessed according to the *ind* value. Changes in the *indvalue* will be reflected at the start of the next while loop.

6. Conclusion

The main objective of the paper was to cultivate an environment for musical aspirants to play virtual musical instruments with easy accessibility and affordability. This paper aimed to solve the issue with OpenCV working to identify fingers in each frame. This is done by HSV color conversion, masking the frame within a HSV range. Then the obtained mask is converted to a Grayscale image, then to a frame with binary values. Finally, it is dilated. On this dilated image, contours are found whose centroids fall within the region where the user has defined his keys. On finding a finger hovering over a key, a note is played with help of *Pygame* and by loading the wav files of keys of the piano with the help of a mixer. Hence this continues till the user closes the window. With help of *Pygame*, a miniature Piano keyboard is created that highlights a key when the key/ note is recognized by OpenCV. *Numpy* module plays a major role in processing each frame and converting them to grayscale and HSV. The *numpy* array is the medium or the data type utilized to process the frames for color recognition. The major achievement of this paper is its attempt to utilize a lightweight Computer Vision technique to allow it to be deployed on various platforms. Amongst techniques that solely utilize HSV recognition this paper provides users to define their template or their workspace on how their piano is set up. This independence is achieved by allowing users to determine their anchor points, and this is crucial in extending to well

beyond to other instruments. This is a step towards achieving affordability and accessibility to new aspirants.

6.1 Future Scope of Improvement

There are methods to improve this virtual musical instrument. The software can be easily upgraded with a camera of larger aperture to track a larger field of view. This means more keys can be tracked and at a time more scales in the Piano can be played. Currently, to move from one scale to another, the up and down buttons are being used. To play a Grand Piano, in an authentic sense, this can be achieved by using a camera with larger aperture and increased clarity, by setting up two or three A4 Sheets on the Table. By freely moving across the ends of the paper, users can play up and down the scales.

Moreover, this work can be applied to an Android application by running an image recognition algorithm on the backend. This will definitely increase affordability for musical aspirants in developing countries who have entered into the technological realm. Musical Instruments need not be restricted to single instruments like a piano or a keyboard. Guitars and other String Instruments too could be developed that require different camera angles, and slightly modified programs to identify each note. This could be customized to various other musical instruments. The customization for a drum might be to change the viewing angles, and have a new anchor. A paper or a blueprint acted as an anchor for this virtual instrument. This can be extended to a drum by having wooden plates/planar objects as anchors that when struck, or when a stick comes to the vicinity of such a planar object, there is a wav file played. Hence, a similar fashion of color recognition can open doors to increasing portability of bulky instruments and efficiently improve affordability.

References

[1] A. S. Konwar, B. S. Borah and C. T. Tuithung, "An American Sign Language detection system using HSV color model and edge detection," 2014 International Conference on Communication and Signal Processing, 2014, pp. 743-747, doi: 10.1109/ICCSP.2014.6949942.

[2] Vladimir Vezhnevets, VassiliSazonov and AllaAndreeva, "A survey on pixel-based skin color detection techniques", Proc. Graphicon, vol. 3, pp. 85-92, 2003.

[3] D. S. Y. Kartika and D. Herumurti, "Koi fish classification based on HSV color space," 2016 International Conference on Information & Communication Technology and Systems (ICTS), 2016, pp. 96-100, doi: 10.1109/ICTS.2016.7910280.

[4] A. Saxena, "VIANO-the virtual piano," 2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT), 2017, pp. 1-4, doi: 10.1109/CIACT.2017.7977332.

[5] L. W. Campbell and A. E. Bobick, "Recognition of human body motion using phase space constraints", Proceedings of the 5th IEEE International conference on Computer Vision, pp. 624-630, 1995.

[6] A. Saner, A. Sharma, A. Patil, A. Soni and P. More, "Dynamic Color Recognition for Video Game Controller," 2021 International Conference on Computing, Communication and Green Engineering (CCGE), 2021, pp. 1-3, doi: 10.1109/CCGE50943.2021.9776403.

[7] P. Singh, B. B. V. L. Deepak, T. Sethi and M. D. P. Murthy, "Real-time object detection and Tracking using color feature and motion," 2015 International Conference on Communications and Signal Processing (ICCSP), 2015, pp. 1236-1241, doi: 10.1109/ICCSP.2015.7322705.

[8] R. Hua and Y. Wang, "Skin color detection based super pixel," 2017 3rd IEEE International Conference on Computer and Communications (ICCC), 2017, pp. 1756-1760, doi: 10.1109/CompComm.2017.8322841.

[9] Thivaharan.S, Srivatsun.G, "Keras Model for Text Classification in Amazon Review Dataset using LSTM", Journal of Artificial Intelligence and Capsule Networks (IROAICN), June 2021, Vol.03, Issue.02, pp.72-89, ISSN: 2582-2012, https://doi.org/10.36548/jaicn.2021.2.001

[10] docs.opencv.org,Contours: Getting Started, Available: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html , [Accessed : 15-October-2022]

[11] docs.opencv.org: Operations on Arrays, Available: https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce98, [Accessed: 15-October-2022]

[12] pygame.org/docs: Setting Display Modes,Available : https://www.pygame.org/docs/tut/DisplayModes.html

[13] Thivaharan.S, Srivatsun.G, "Maximizing the Prediction Accuracy in Tweet Sentiment Extraction using Tensor Flow based Deep Neural Networks", IRO Journal of

Ubiquitous Computing and Communication Technologies (IROUCCT), June 2021, Vol.03, Issue.02, pp.61-79, ISSN: 2582-337X, https://doi.org/10.36548/jucct.2021.2.001

[14] Rehanullah Khan, Allan Hanbury, Julian Stöttinger and Abdul Bais, "Color based skin classification", Pattern Recognition Letters, vol. 33, no. 2, pp. 157-163, 2012.

[15] RabiaJafri, Syed Abid Ali, Hamid R Arabnia and Shameem Fatima, "Computer vision-based object recognition for the visually impaired in an indoors environment: a survey", The Visual Computer: International Journal of Computer Graphics, vol. 30, no. 11, pp. 1197-1222, 2014.

[16] Praveen Kakumanu, Sokratis Makrogiannis and Nikolaos Bourbakis, "A survey of skin-color modeling and detection methods", Pattern recognition, vol. 40, no. 3, pp. 1106-1122, 2007.

[17] Siddharth S. Rautaray and Anupam Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey", Artificial Intelligence Review, vol. 43, no. 1, pp. 1-54, Jan 2015.

[18] Hui Liang, Jin Wang, Qian Sun, Yong-Jin Liu, Junsong Yuan, Jun Luo, et al., "Barehanded music: real-time hand interaction for virtual piano", Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 87-94, 2016.

[19] VA Oliveira and A Conci, "Skin detection using hsv color space", H. Pedrini& J. Marques de Carvalho Workshops of Sibgrapi, pp. 1-2, 2009.

[20] Dengsheng Zhang and Guojun Lu, "Review of shape representation and description techniques", Pattern recognition, vol. 37, no. 1, pp. 1-19, 2004.

[21] L. Barba-Guamán, C. Calderon-Cordova and P. A. Quezada-Sarmiento, "Detection of moving objects through color thresholding," 2017 12th Iberian Conference on Information Systems and Technologies (CISTI), 2017, pp. 1-6, doi: 10.23919/CISTI.2017.7975755.

[22] P Lalitha Surya Kumari, C.H.Sarada devi, S. Thivaharan, K Srinivas, Avula Damodaram, A Resilient Group Session Key Authentication Methodology for Secured Peer to Peer Networks using Zero Knowledge Protocol, Optik, 2022, 170345, ISSN 0030-4026, https://doi.org/10.1016/j.ijleo.2022.170345. (https://www.sciencedirect.com/science/article/pii/S0030402622016035)

[23] A. S. Silva, F. M. Q. Severgnini, M. L. Oliveira, V. M. S. Mendes and Z. M. A. Peixoto, "Object Tracking by Color and Active Contour Models Segmentation," in

IEEE Latin America Transactions, vol. 14, no. 3, pp. 1488-1493, March 2016, doi: 10.1109/TLA.2016.7459639.

[24] G. Li, H. Tang, Y. Sun, J. Kong, G. Jiang et al., "Hand gesture recognition based on convolution neural network", Cluster Comput, vol. 22, pp. 2719-2729, 2019.

[25] A. Yilmaz, O. Javed and M. Shah, "Object Tracking: A Survey", ACM Computer Survey, vol. 38, no. 4, pp. 1-45, 2006.

[26] F. Russo, "An image enhancement technique combining sharpening and noise reduction", IEEE Transactions on Instrumentation and Measurement, vol. 51, no. 4, pp. 824-828, 2002.

[27] Yuheng Song and Hao Yan, "Image segmentation techniques overview", 2017 IEEE Asia Modelling Symposium (AMS), pp. 103-107, 2017.

[28] Susmita Sahu, Himadri Sarma and Dibya Jyoti Bora, "Image segmentation and its different techniques: An in-depth analysis", 2018 IEEE International Conference on Research in Intelligent and Computing in Engineering (RICE), pp. 1-7, 2018.

[29] Mr. Thivaharan S, Dr. G. Srivatsun, Mr. KARTHIKEYAN A S, Dr. R. Santhosh, Portable assistant to read and interpret braille paper print to aid visually challenged persons in Tamil language, Status: Published, Application No: 202241038275, Journal No: 27/2022 (Part2), Date of Filing: 04.07.2022, Publication Date: 08.07.2022, Page No: 43144, No of pages: 8, Claims: 9, Indian patent.

[30] R. Guo, J. Cui, W. Zhao, S. Li and A. Hao, "Hand-by-Hand Mentor: An AR based Training System for Piano Performance," 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Lisbon, Portugal, 2021, pp. 436-437, doi: 10.1109/VRW52623.2021.00100.

[31] W. Qiao, R. Wei, S. Zhao, D. Huo and F. Li, "A real-time virtual piano based on gesture capture data," 2017 12th International Conference on Computer Science and Education (ICCSE), Houston, TX, USA, 2017, pp. 740-743, doi: 10.1109/ICCSE.2017.8085592.