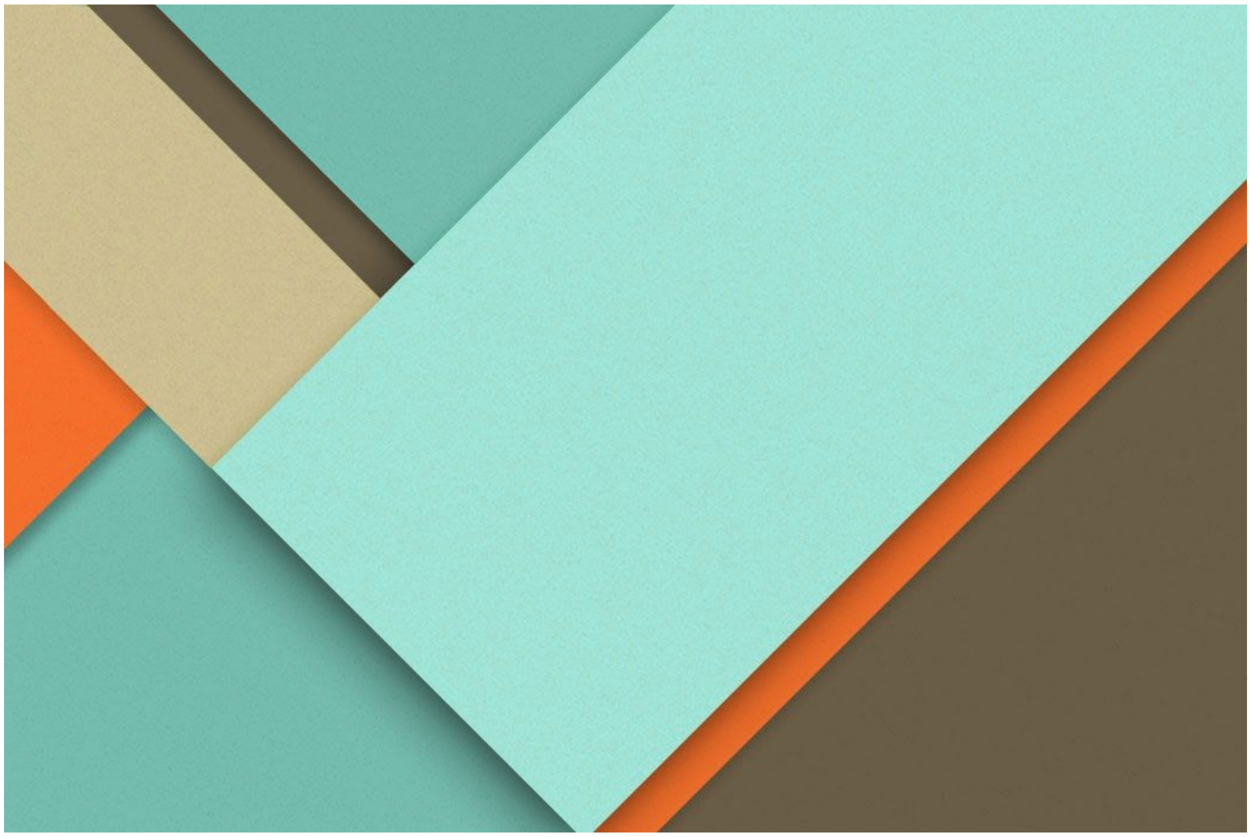


AIR CANVAS

IMAGINING THE FUTURE

Proposed by:

1. Ayush Ranjan, Team ERA, IIT-Kanpur
2. Sidhartha Watsa, Team ERA, IIT-Kanpur

Date: 12-05-2020

Introduction

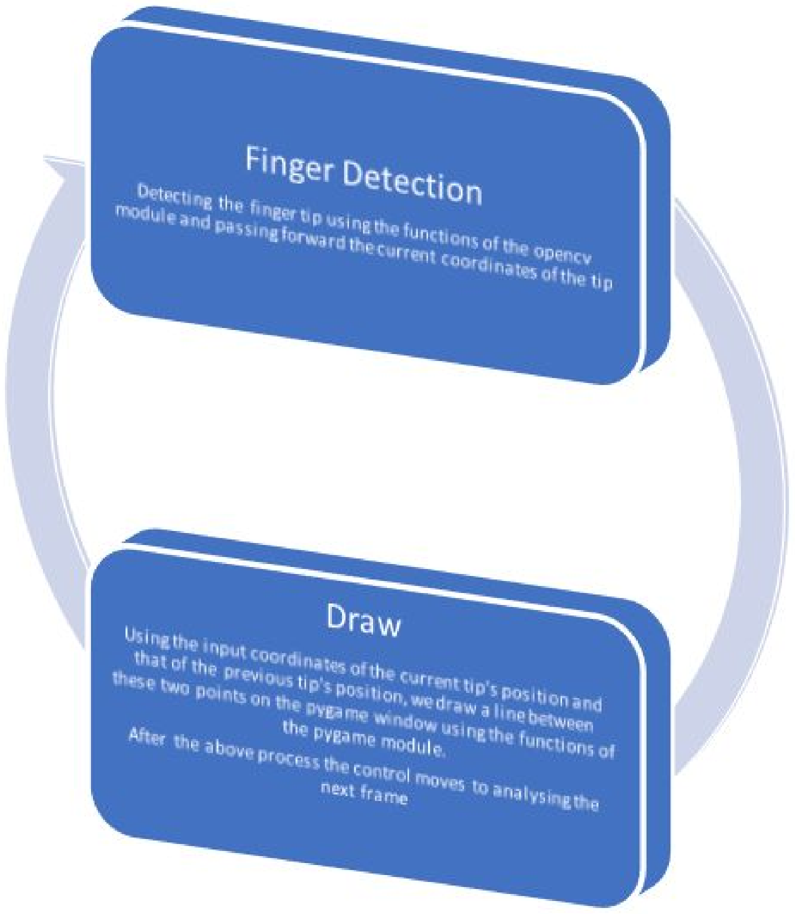
Often you might have seen geniuses doing calculations by just waving their fingers in the air. Human brain is advanced enough to imagine the pattern drawn before his eyes and could store them in his memory quite for some time, but just imagine how it would have been that whatever we draw with our fingers in the air gets actually drawn on some surface and you could have been able to see that drawing with your eyes rather than pressurising your brain to imagine and make a picture in your brain. Well, you would be able to see it as long as you want. Just imagine that you would be able to draw anything without using pens, pencils, brushes, etc and save it to your smartphone or PC for future use. Just imagine that you wave your finger in the air and a message is sent to your peers without typing that message manually. This project is a way to convert all your imaginations into reality where anything can be drawn on the screen by just waving your fingers in the air.

Model

The project presents a software application which, on launching, displays a drawing platform a window showing the frames captured by the camera (attached to the PC) in succession. When you draw anything using your forefinger in the air before the camera, that drawing will be drawn on the drawing platform beside. You can also change the color and thickness of the curves drawn on the screen by suitable keys.

Structure

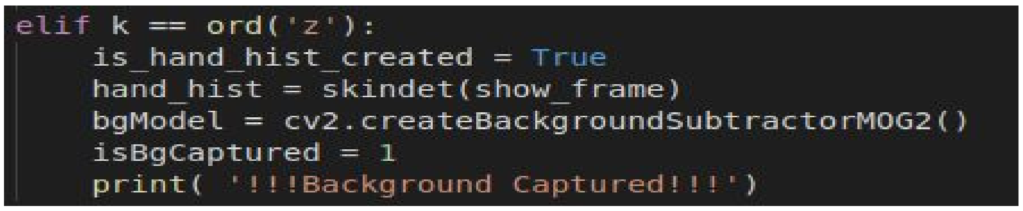
The program for this application contains the basic structure as depicted in the loop diagram below:


Finger Detection
Detecting the finger tip using the functions of the opencv module and passing forward the current coordinates of the tip
Draw
Using the input coordinates of the current tip's position and that of the previous tip's position, we draw a line between these two points on the pygame window using the functions of the pygame module.
After the above process the control moves to analysing the next frame

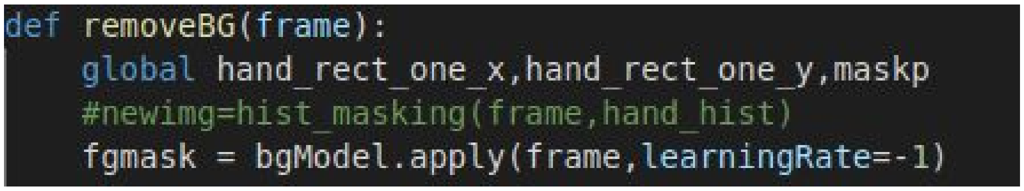
I. FINGER DETECTION

The fingertip detection was the first part in the process of making the AIR CANVAS. Manual fingertip detection is done in several steps. This included background subtraction with skin detection and then use of a convex hull on the contour with maximum area in the masked image.

1. **Background Subtraction** -- Background Subtraction method is used for detecting moving objects in a video. In this method, the reference frame is compared with the next frames and the moving objects are thus detected. We used the cv2.createBackgroundSubtractorMOG2() method for this task. This alone worked well in still very backgrounds. It was decided that Skin detection should be done along with this process.



```
elif k == ord('z'):
    is_hand_hist_created = True
    hand_hist = skindet(show_frame)
    bgModel = cv2.createBackgroundSubtractorMOG2()
    isBgCaptured = 1
    print( '!!!Background Captured!!!')
```



```
def removeBG(frame):
    global hand_rect_one_x,hand_rect_one_y,maskp
    #newimg=hist_masking(frame,hand_hist)
    fgmask = bgModel.apply(frame,learningRate=-1)
```

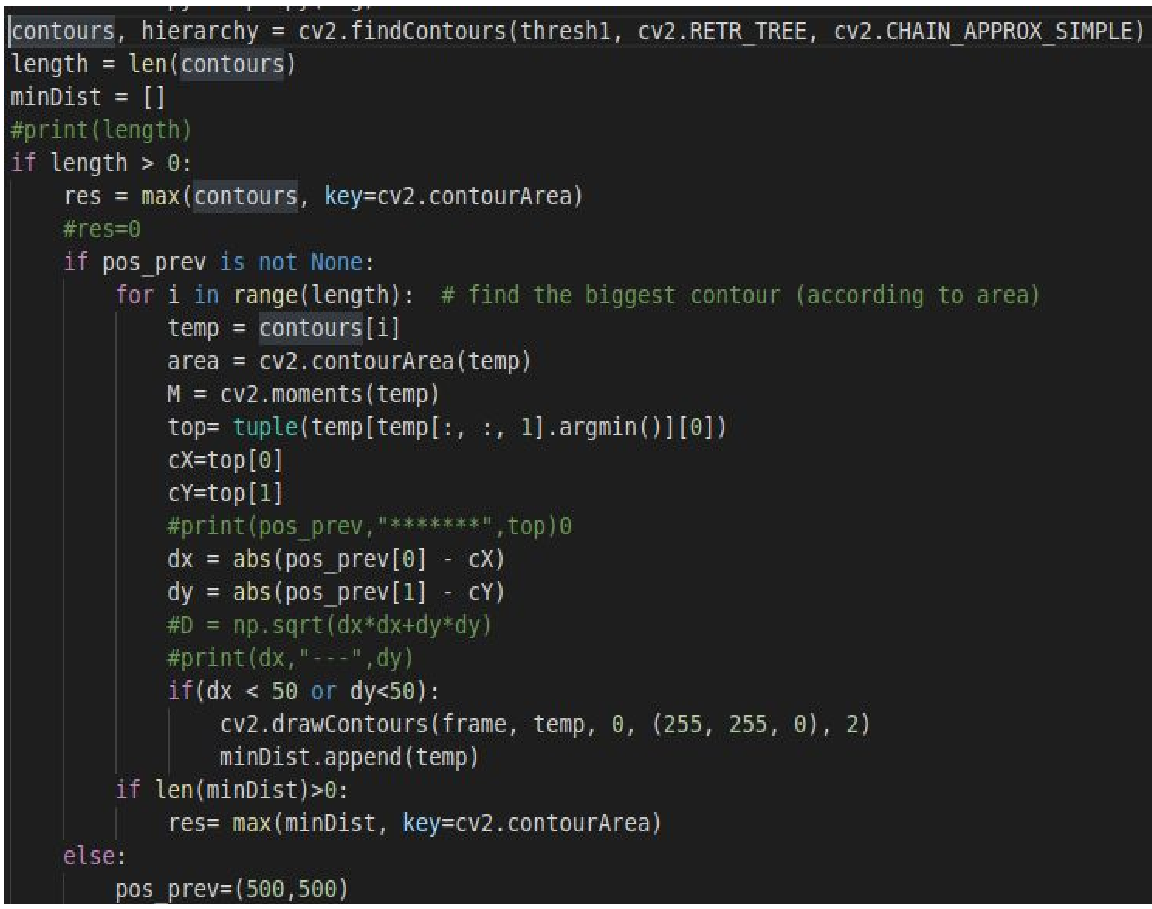
2. **Skin Detection**

- *By YCrCb colorspace mask*

The YCrCb color space was initially used to mask the region lying in the range of skin color. It was effective to some extent but the finger used to get detached from the fist in the masked image in some frames. This caused the tip to shift from the fingertip to the fist as it had a greater area.

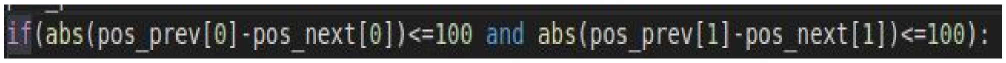
Several checks were implemented to avoid such large fluctuations.

- Not the largest contour but the largest contour in a certain range of the previous position of the tip was taken into consideration.



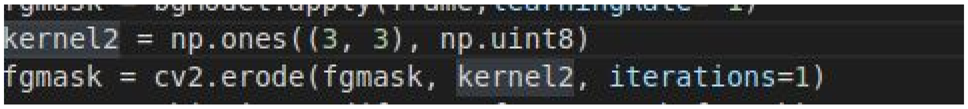
```
contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
length = len(contours)
minDist = []
#print(length)
if length > 0:
    res = max(contours, key=cv2.contourArea)
    #res=0
    if pos_prev is not None:
        for i in range(length):  # find the biggest contour (according to area)
            temp = contours[i]
            area = cv2.contourArea(temp)
            M = cv2.moments(temp)
            top= tuple(temp[temp[:, :, 1].argmin()][0])
            cX=top[0]
            cY=top[1]
            #print(pos_prev,"*******",top)0
            dx = abs(pos_prev[0] - cX)
            dy = abs(pos_prev[1] - cY)
            #D = np.sqrt(dx*dx+dy*dy)
            #print(dx,"---",dy)
            if(dx < 50 or dy<50):
                cv2.drawContours(frame, temp, 0, (255, 255, 0), 2)
                minDist.append(temp)
        if len(minDist)>0:
            res= max(minDist, key=cv2.contourArea)
    else:
        pos_prev=(500,500)
```

- Secondly, the brush was dropped in case of a large fluctuation.

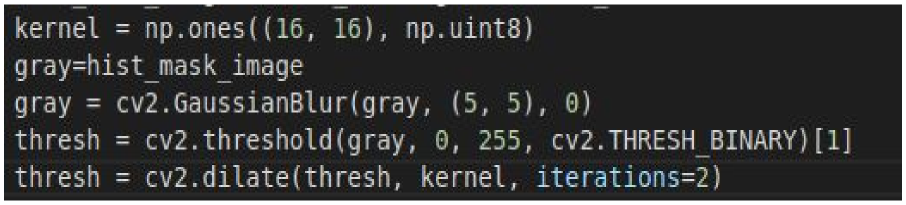


```
if(abs(pos_prev[0]-pos_next[0])<=100 and abs(pos_prev[1]-pos_next[1])<=100):
```

- The masked image was eroded with a small kernel of size [3,3] but dilated with a kernel of size [16,16].



```
kernel2 = np.ones((3, 3), np.uint8)
fgmask = cv2.erode(fgmask, kernel2, iterations=1)
```

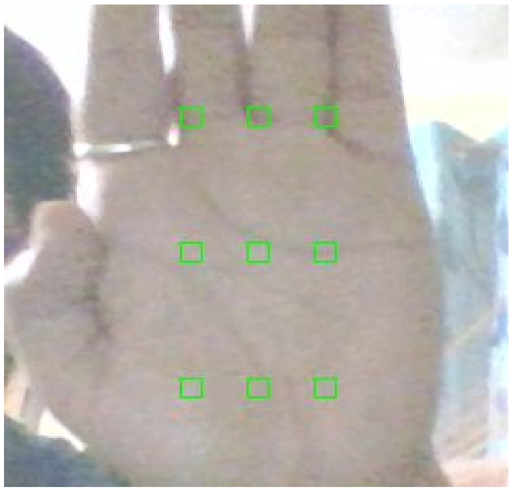


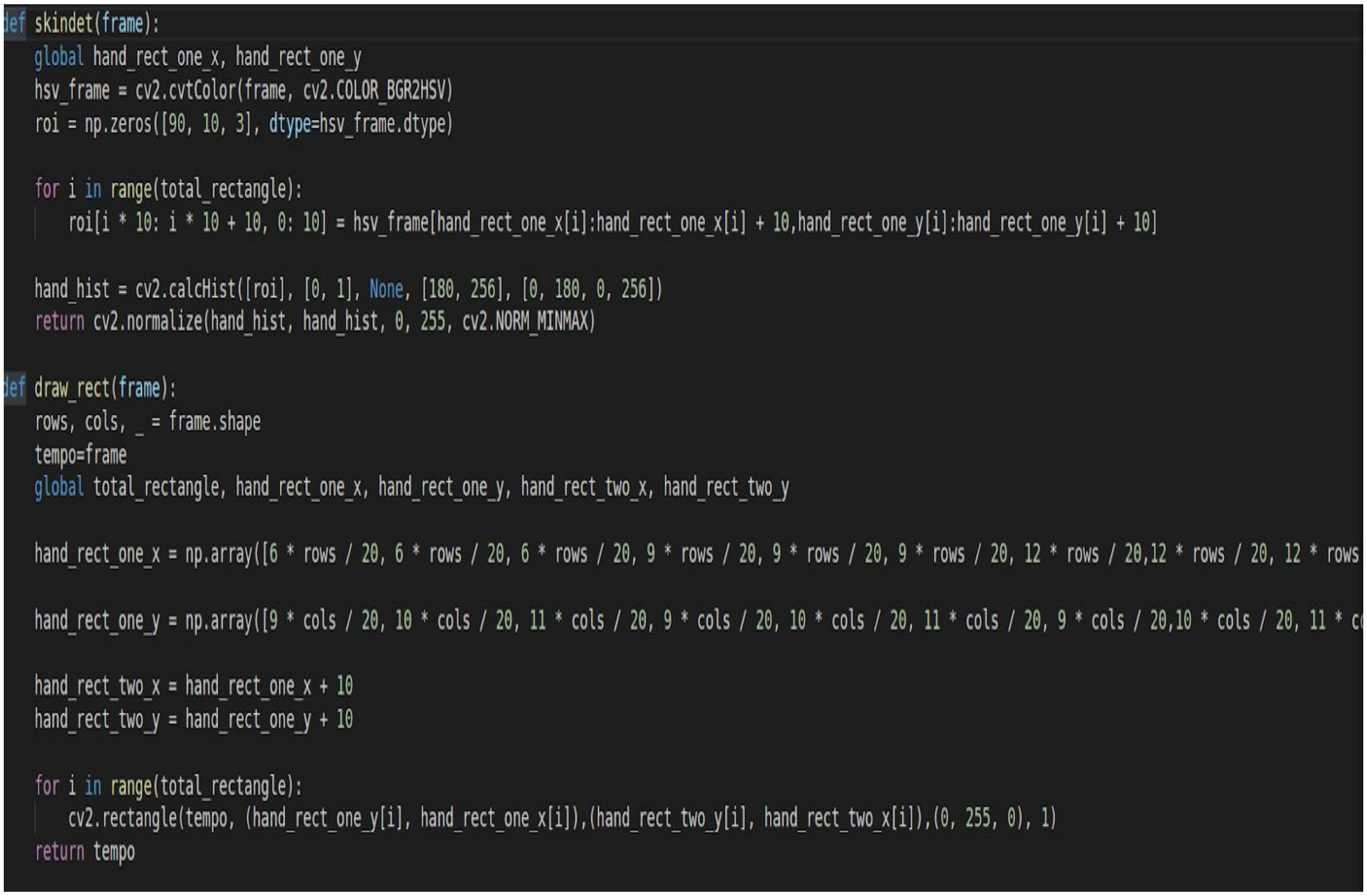
```
kernel = np.ones((16, 16), np.uint8)
gray=hist_mask_image
gray = cv2.GaussianBlur(gray, (5, 5), 0)
thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]
thresh = cv2.dilate(thresh, kernel, iterations=2)
```

- *By Histogram Backprojection*

This method proved to work well because of the code not needing any predefined ranges for masking the skin region. The histogram of the color of the skin of the user is calculated at the very start along with storing the background. The rest of the frames are masked by back-projecting this histogram. This process worked better and a lot of issues were resolved.

- First, nine rectangles are drawn on the screen, and the user has to place his palm such that it covers them and then press ‘z’. Now the histogram is calculated for the color lying in those rectangles.

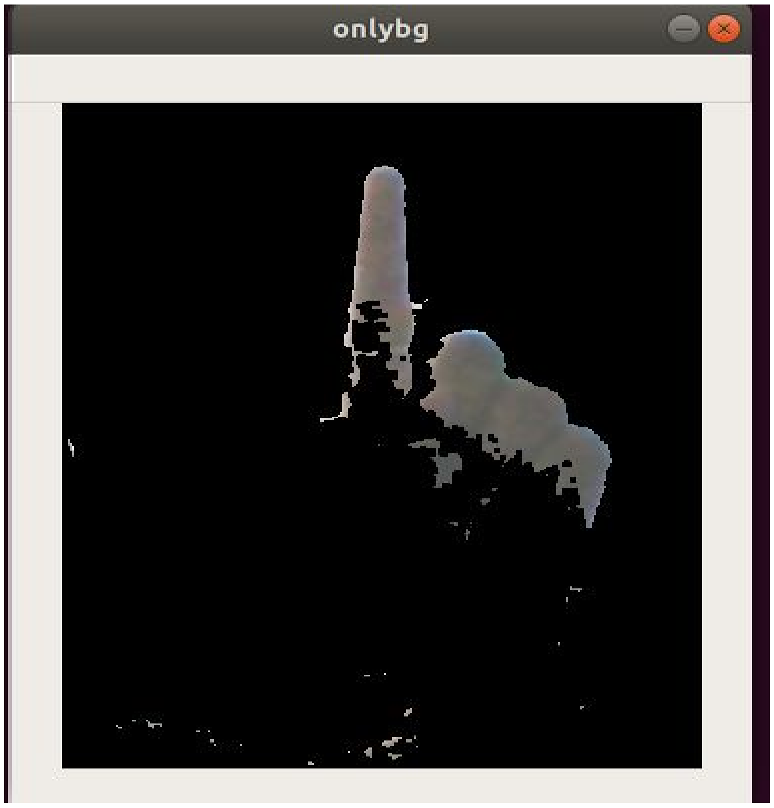




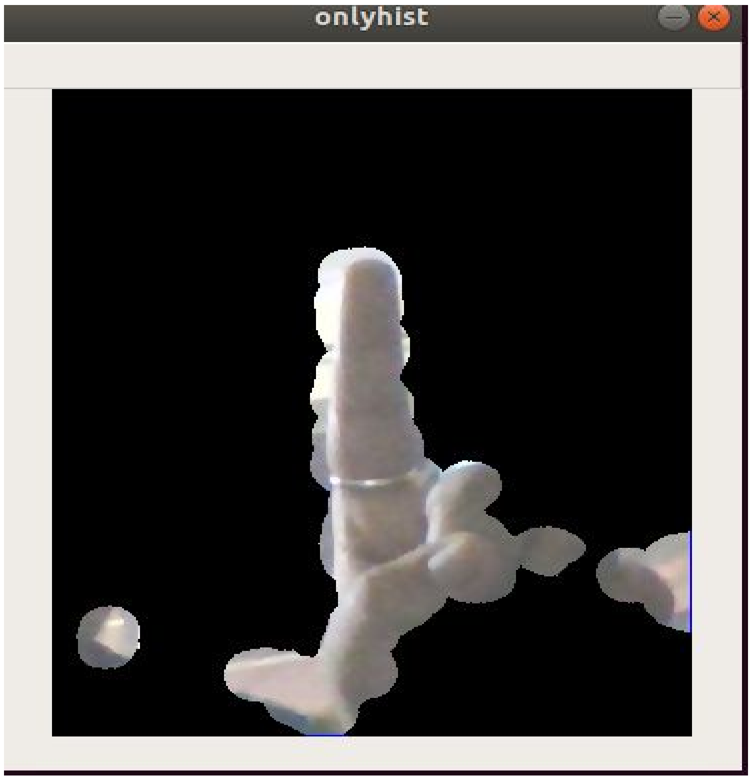
```
def skindet(frame):
    global hand_rect_one_x, hand_rect_one_y
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    roi = np.zeros([90, 10, 3], dtype=hsv_frame.dtype)

    for i in range(total_rectangle):
        roi[i * 10: i * 10 + 10, 0: 10] = hsv_frame[hand_rect_one_x[i]:hand_rect_one_x[i] + 10,hand_rect_one_y[i]:hand_rect_one_y[i] + 10]

    hand_hist = cv2.calcHist([roi], [0, 1], None, [180, 256], [0, 180, 0, 256])
    return cv2.normalize(hand_hist, hand_hist, 0, 255, cv2.NORM_MINMAX)

def draw_rect(frame):
    rows, cols, _ = frame.shape
    tempo=frame
    global total_rectangle, hand_rect_one_x, hand_rect_one_y, hand_rect_two_x, hand_rect_two_y

    hand_rect_one_x = np.array([6 * rows / 20, 6 * rows / 20, 6 * rows / 20, 9 * rows / 20, 9 * rows / 20, 9 * rows / 20, 12 * rows / 20,12 * rows / 20, 12 * rows

    hand_rect_one_y = np.array([9 * cols / 20, 10 * cols / 20, 11 * cols / 20, 9 * cols / 20, 10 * cols / 20, 11 * cols / 20, 9 * cols / 20,10 * cols / 20, 11 * c

    hand_rect_two_x = hand_rect_one_x + 10
    hand_rect_two_y = hand_rect_one_y + 10

    for i in range(total_rectangle):
        cv2.rectangle(tempo, (hand_rect_one_y[i], hand_rect_one_x[i]),(hand_rect_two_y[i], hand_rect_two_x[i]),(0, 255, 0), 1)
    return tempo
```

Now the following images depict different cases.

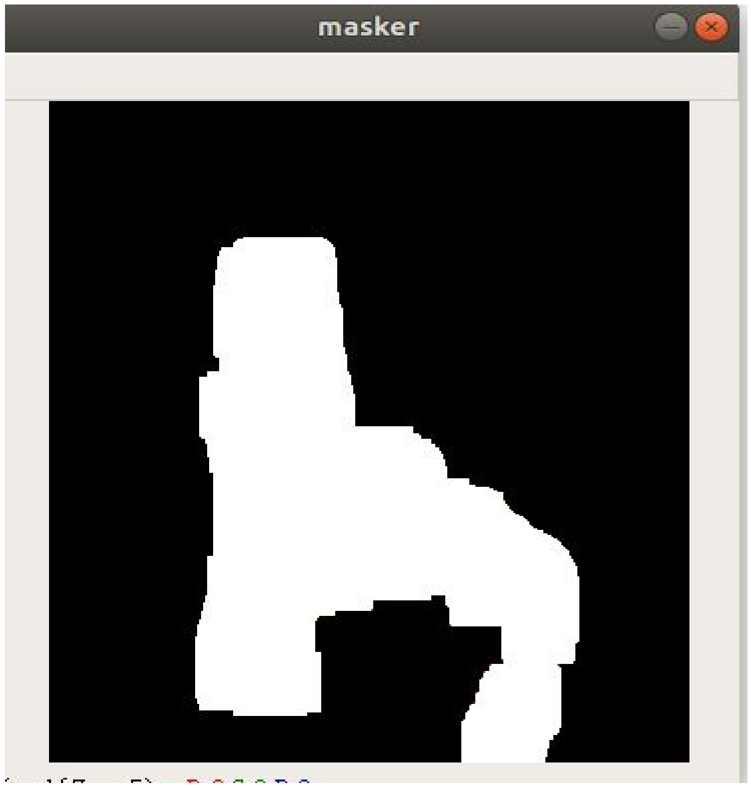
1. *Only background Subtraction*



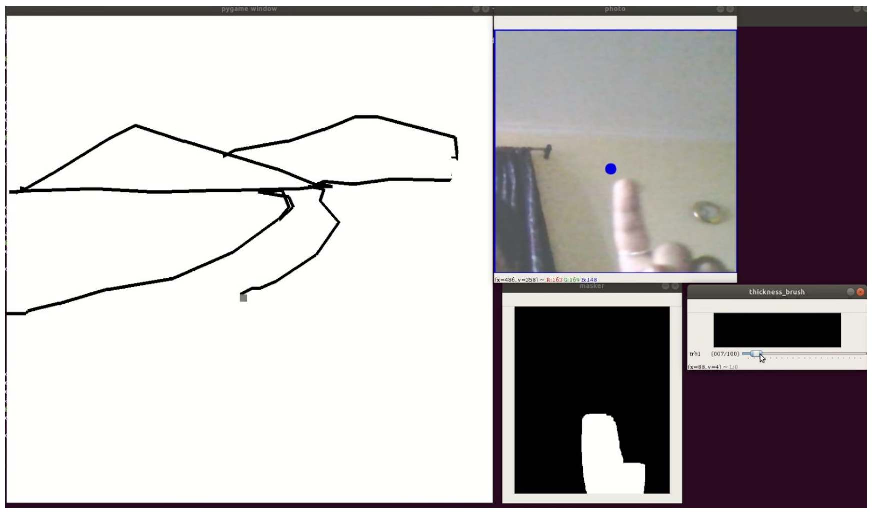
2. *Only Histogram Backprojection*



3. *Combined mask*



4. *Finally the pointer is located.*



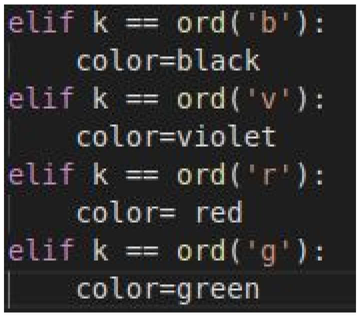
II. DRAWING

Pygame is an amazing library that allows us to build small gui wherein we can draw anything, insert images, vedios, icons and so on. It constantly loops over the gui's screen and thus through simple logical code, we can easily make our drawings move in some particular pattern. It defines a term 'event' which basically refers to any activity done by the mouse or keypad. Thus we can easily incorporate various basic features like mouse clicks, key press, etc. We can draw numerous shapes using very easy functions provided in this module. Having done the fingertip detection, now we know two coordinates, the current and the previous one, and these are enough to draw a line between them using the **pygame.draw.line()** function using the values of the selected color and thickness. This process iterates over frames and we see a continuous line drawn on the pygame screen.

III. ADDITIONAL FEATURES

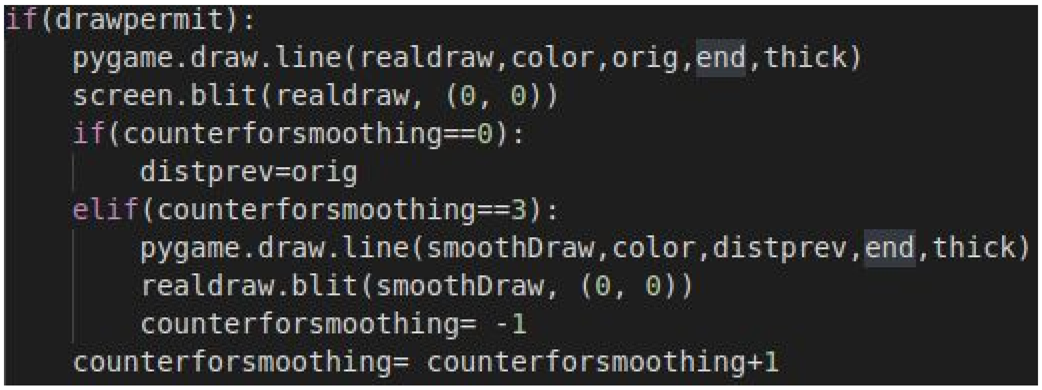
Besides simply drawing, we have also incorporated various other features like

- Color changing using various keys on the keyboard.



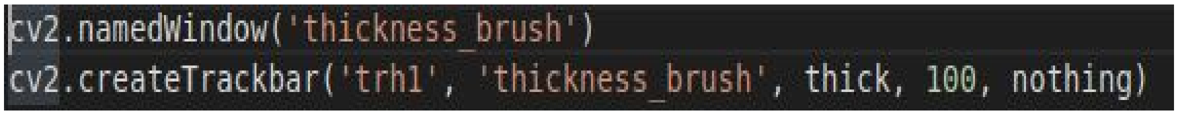
```
elif k == ord('b'):
    color=black
elif k == ord('v'):
    color=violet
elif k == ord('r'):
    color= red
elif k == ord('g'):
    color=green
```

- The Drawing is smoothened by joining the every fourth point and this reduces unnecessary fluctuations of the finger tips.

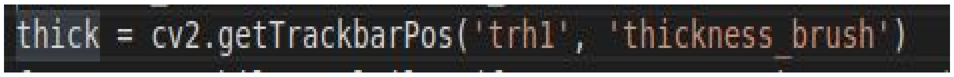


```
if(drawpermit):
    pygame.draw.line(realdraw,color,orig,end,thick)
    screen.blit(realdraw, (0, 0))
    if(counterforsmoothing==0):
        distprev=orig
    elif(counterforsmoothing==3):
        pygame.draw.line(smoothDraw,color,distprev,end,thick)
        realdraw.blit(smoothDraw, (0, 0))
        counterforsmoothing= -1
    counterforsmoothing= counterforsmoothing+1
```

- Changing the thickness of the line drawn using the trackbar window.



```
cv2.namedWindow('thickness_brush')
cv2.createTrackbar('trh1', 'thickness_brush', thick, 100, nothing)
```



```
thick = cv2.getTrackbarPos('trh1', 'thickness_brush')
```

- Icon of a small pen that points the current position of your fingers tip on the imaginary 2D surface of air.

How to use it ?

- Run the code on the terminal. You will see your webcam image on the screen with 9 rectangles. Place your palm in such a way that it covers the rectangles and then press ‘z’.
- Now the pygame screen appears along with a portion cut from the webcam image which has a blue circle which you have to move with your finger. The pen on the screen moves along with your blue circle. The masked image is also shown so that errors can be recognised.
- Now once you get comfortable with moving the brush press ‘d’ to start drawing. Remember to keep your opencv window selected for the program to accept your keyboard inputs. Now you can press:

  g- green color.

  r- red color

  v- violet color

  y- yellow color

  l- blue color

  e- eraser.

  d- to toggle between drawing and moving without drawing

Future modifications

What could be probably the future modifications to this wonderful project. We could incorporate several other features like

- Control the actions like clicks (to start and stop drawing), changing colors and width of the line etc using some special gestures with our palm

- Until now we have achieved to draw something in a 2D space, but probably we could draw a 3D diagram with our fingers and visualize this 3D drawing on the computer screen.
- We could use various machine learning algorithms to convert these cursive texts drawn using this application into actual texts appearing on the screen which can further be saved in the memory, sent as a message to some guy, could be used as a command to do several tasks in the PC and so on.

Bibliography

Acknowledgement

We got extensive support and motivation from the Team ERA-IITK and we are very grateful to them for giving us this opportunity.