import cv2  
import pickle  
import numpy as np  
import pyautogui as gui  
  
with open("range.pickle", "rb") as f: # range.pickle is generated by range-detector.py  
 t = pickle.load(f)  
cam = cv2.VideoCapture(1)  
if cam.read()[0]==False:  
 cam=cv2.VideoCapture(0)  
hsv\_lower = np.array([t[0], t[1], t[2]])   
hsv\_upper = np.array([t[3], t[4], t[5]])  
width = cam.get(cv2.CAP\_PROP\_FRAME\_WIDTH) # width of video captured by the webcam  
height = cam.get(cv2.CAP\_PROP\_FRAME\_HEIGHT) # height of the video captured by the webcam  
  
def get\_keys():  
 *"""  
 this function is used to design the keyboard.  
 it returns the 4 parameter that are needed to design the keys.  
 they are key label, top right corner coordinate, bottom left corner coordinate, and center coordinate  
 """* max\_keys\_in\_a\_row = 11 # max number of keys in any row is 10 i.e the first row which contains 1234567890'backspace'  
 key\_width = int(width/max\_keys\_in\_a\_row) # width of one key. width is divided by 10 as the max number of keys in a single row is 11.  
   
 row0\_key\_width = key\_width \* 11 # width of zeroth or numeric row of keys  
 row1\_key\_width = key\_width \* 10 # width of first row  
 row2\_key\_width = key\_width \* 9 # width of second row  
 row3\_key\_width = key\_width \* 7 # width of third row  
 row4\_key\_width = key\_width \* 5 # width of space  
 row\_keys = [] # stores the keys along with its 2 corner coordinates and the center coordinate  
  
 # for the zeroth row  
 x1, y1 = 0, int((height - key\_width \* 5) / 2) # 5 is due to the fact that we will have 5 rows. y1 is set such that the whole keyboard has equal margin on both top and bottom  
 x2, y2 = key\_width + x1, key\_width + y1  
 c1, c2 = x1, y1 # copying x1, x2, y1 and y2  
 keys = "1 2 3 4 5 6 7 8 9 0 <-"  
 keys = keys.split(" ")  
 for key in keys:  
 if key == "<-":  
 row\_keys.append([key, (x1, y1), (x2, y2), (int((x2+x1)/2) - 25, int((y2+y1)/2) + 10)])  
 else:  
 row\_keys.append([key, (x1, y1), (x2, y2), (int((x2+x1)/2) - 5, int((y2+y1)/2) + 10)])  
 x1 += key\_width  
 x2 += key\_width  
 x1, y1 = c1, c2 # copying back from c1, c2, c3 and c4  
  
 # for the first row  
 x1, y1 = int((row0\_key\_width - row1\_key\_width) / 2) + x1, y1 + key\_width   
 x2, y2 = key\_width + x1, key\_width + y1  
 c1, c2 = x1, y1 # copying x1, x2, y1 and y2  
 keys = "qwertyuiop"  
 for key in keys:  
 row\_keys.append([key, (x1, y1), (x2, y2), (int((x2+x1)/2) - 5, int((y2+y1)/2) + 10)])  
 x1 += key\_width  
 x2 += key\_width  
 x1, y1 = c1, c2 # copying back from c1, c2, c3 and c4  
  
 # for second row  
 x1, y1 = int((row1\_key\_width - row2\_key\_width) / 2) + x1, y1 + key\_width # x1 is set such that it leaves equal margin on both left and right side  
 x2, y2 = key\_width + x1, key\_width + y1  
 c1, c2 = x1, y1  
 keys = "asdfghjkl"  
 for key in keys:  
 row\_keys.append([key, (x1, y1), (x2, y2), (int((x2+x1)/2) - 5, int((y2+y1)/2) + 10)])  
 x1 += key\_width  
 x2 += key\_width  
 x1, y1 = c1, c2  
  
 # for third row  
 x1, y1 = int((row2\_key\_width - row3\_key\_width) / 2) + x1, y1 + key\_width  
 x2, y2 = key\_width + x1, key\_width + y1   
 c1, c2 = x1, y1   
 keys = "zxcvbnm"  
 for key in keys:  
 row\_keys.append([key, (x1, y1), (x2, y2), (int((x2+x1)/2) - 5, int((y2+y1)/2) + 10)])  
 x1 += key\_width  
 x2 += key\_width  
 x1, y1 = c1, c2  
  
 # for the space bar  
 x1, y1 = int((row3\_key\_width - row4\_key\_width) / 2) + x1, y1 + key\_width  
 x2, y2 = 5 \* key\_width + x1, key\_width + y1   
 c1, c2 = x1, y1   
 keys = " "  
 for key in keys:  
 row\_keys.append([key, (x1, y1), (x2, y2), (int((x2+x1)/2) - 5, int((y2+y1)/2) + 10)])  
 x1 += key\_width  
 x2 += key\_width  
 x1, y1 = c1, c2  
  
 return row\_keys  
  
  
def do\_keypress(img, center, row\_keys\_points):  
 # this fuction presses a key and marks the pressed key with blue color  
 for row in row\_keys\_points:  
 arr1 = list(np.int0(np.array(center) >= np.array(row[1]))) # center of the contour has greater value than the top left corner point of a key   
 arr2 = list(np.int0(np.array(center) <= np.array(row[2]))) # center of the contour has less value than the bottom right corner point of a key   
 if arr1 == [1, 1] and arr2 == [1, 1]:  
 if row[0] == '<-':  
 gui.press('backspace')  
 else:  
 gui.press(row[0])  
 cv2.fillConvexPoly(img, np.array([np.array(row[1]), \  
 np.array([row[1][0], row[2][1]]), \  
 np.array(row[2]), \  
 np.array([row[2][0], row[1][1]])]), \  
 (255, 0, 0))  
 return img  
  
  
def main():  
 row\_keys\_points = get\_keys()  
 new\_area, old\_area = 0, 0  
 c, c2 = 0, 0 # c stores the number of iterations for calculating the difference b/w present area and previous area  
 # c2 stores the number of iterations for calculating the difference b/w present center and previous center  
 flag\_keypress = False # if a key is pressed then this flag is True  
 while True:  
 img = cam.read()[1]  
 img = cv2.flip(img, 1)  
 imgHSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
 mask = cv2.inRange(imgHSV, hsv\_lower, hsv\_upper)  
 blur = cv2.medianBlur(mask, 15)  
 blur = cv2.GaussianBlur(blur , (5,5), 0)  
 thresh = cv2.threshold(blur, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)[1]  
 contours = cv2.findContours(thresh.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)[1]  
  
 if len(contours) > 0:  
 cnt = max(contours, key = cv2.contourArea)  
   
 if cv2.contourArea(cnt) > 350:  
 # draw a rectangle and a center   
 rect = cv2.minAreaRect(cnt)  
 center = list(rect[0])  
 box = cv2.boxPoints(rect)  
 box = np.int0(box)  
 cv2.circle(img, tuple(np.int0(center)), 2, (0, 255, 0), 2)  
 cv2.drawContours(img,[box],0,(0,0,255),2)  
   
 # calculation of difference of area and center  
 new\_area = cv2.contourArea(cnt)  
 new\_center = np.int0(center)  
 if c == 0:  
 old\_area = new\_area  
 c += 1  
 diff\_area = 0  
 if c > 3: # after every 3rd iteration difference of area is calculated  
 diff\_area = new\_area - old\_area  
 c = 0  
 if c2 == 0:  
 old\_center = new\_center  
 c2 += 1  
 diff\_center = np.array([0, 0])  
 if c2 > 5: # after every 5th iteration difference of center is claculated  
 diff\_center = new\_center - old\_center  
 c2 = 0  
   
 # setting some thresholds  
 center\_threshold = 10  
 area\_threshold = 200  
 if abs(diff\_center[0]) < center\_threshold or abs(diff\_center[1]) < center\_threshold:  
 print(diff\_area)  
 if diff\_area > area\_threshold and flag\_keypress == False:  
 img = do\_keypress(img, new\_center, row\_keys\_points)  
 flag\_keypress = True  
 elif diff\_area < -(area\_threshold) and flag\_keypress == True:  
 flag\_keypress = False  
 else:  
 flag\_keypress = False  
 else:  
 flag\_keypress = False  
  
 # displaying the keyboard  
 for key in row\_keys\_points:  
 cv2.putText(img, key[0], key[3], cv2.FONT\_HERSHEY\_DUPLEX, 1, (0, 255, 0))  
 cv2.rectangle(img, key[1], key[2], (0, 255, 0), thickness = 2)  
  
 cv2.imshow("img", img)  
   
 if cv2.waitKey(1) == ord('q'):  
 break  
  
 cam.release()  
 cv2.destroyAllWindows()  
  
main()