A VIRTUAL KEYBOARD

PROJECT REPORT

SUBMITTED BY

PRINCE JAISWAL J

312323205172

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

St.JOSEPH'S COLLEGE OF ENGINEERING

(An Autonomous Institution)

St.JOSEPH'S GROUP OF INSTITUTIONS

OMR,CHENNAI-119

ABSTRACT

The keyboard is the only component that has remained unchanged for decades despite disks and components becoming smaller. We opt for a virtual keyboard due to the difficulty of shrinking a conventional keyboard. Software that simulates a standard keyboard is our virtual keyboard. On a computer screen, a picture of a keyboard is shown, and the user can type by pointing and clicking on the pictures of the keys. To obtain the correct keystroke, a camera here tracks the typist's finger movements. Instead of hitting actual keys on a physical keyboard, a user types on or within a wireless or optically detectable surface or area. so that it can provide the user with complete keyboard access on a small handheld device, like a mobile phone or personal digital assistant (PDA). To relieve finger cramping, utilize a virtual keyboard.

Green is detected utilizing Color Detection Technology. To make the virtual keyboard function like a real one, users can tap the images of the keys utilising detecting technology, which is based on optical recognition. It is made to accommodate any typing speed. Switches that can be activated in a variety of ways use the muscles that are most suitable for each user. With virtual keyboards, people with significant mobility issues can use computers. To speed up entry, some virtual keyboards incorporate word prediction. Python's Kivy is a graphical user interface (GUI) tool that works with any platform. because it runs on Android, iOS, Linux, Windows, and other platforms. It is utilized in the Android application development process, but this does not preclude its use in desktop applications.

Keywords: virtual keyboard, colour detection, detection technology.

INTRODUCTION

Computing devices have the capacity to significantly simplify our day-to-day lives. Most of the time, we need a controller with a lot of keys to control these

machines. In the modern world, we are accustomed to using a variety of keyboards to operate digital gadgets like computers, televisions, phones, and other things. As a result, the physical keyboard is currently the most widely used device for human-machine interaction. However, due to the necessity of customizing a keyboard or controller for each machine, these physical keyboards are not the best option. Additionally, each key's functions must be learned and memorized. Additionally, body language is the foundation of natural human communication. To put it another way, we naturally use body language, postures, and gestures to convey information. Machines, on the other hand, lack these biological characteristics and are unable to comprehend our intentions. As a result, researchers are currently focusing a lot of their attention on figuring out how to create a human-machine interaction that is straightforward, natural, and effective.

Everybody is aware of how much room a keyboard takes up. A physical keyboard substitute must be found to improve portability. The most common solution is a touch screen virtual keyboard.

Software that mimics a standard keyboard is called a virtual keyboard. On a computer screen, a picture of a keyboard is shown, and the user can type by pointing and clicking on the pictures of the keys. Switches that can be activated in a variety of ways use the muscles that are most suitable for each user. With virtual keyboards, people with significant mobility issues can use computers.

Some virtual keyboards include word prediction to speed up entry. The graphical user interface (GUI) tool Kivy from Python is cross-platform. because it is compatible with Linux, Windows, Android, iOS, and other platforms. Although it is mostly used in the creation of Android applications, desktop programs can also make use of it.

Instead of hitting actual keys on a computer keyboard, users can type on a wireless or optically detectable surface or area. The term "virtual keyboard" applies to this. For someone utilizing a small, portable device like a mobile phone or personal digital assistant, such a system may allow full keyboard access (PDA). One technology kind optically transfers the keyboard onto a flat surface. As the user presses a key, the optical device detects it and communicates the keystroke to the computer.

A futuristic device would theoretically project the keyboard into space, enabling users to type by hurling their fingers into the air.

A virtual keyboard is a term that is occasionally used to describe a soft keyboard that displays as an image map on a screen. Sometimes, a software-based keyboard can be altered.

LITERATURE SURVEY

The layout and implementation of the virtual keyboard have been the subject of numerous suggestions, the bulk of which called for the incorporation of hardware [1]. One proposal suggested using a CMOS camera to implement a keyboard, which added complexity and defined a virtual keyboard for Android-powered smartphones. The K-means algorithm or manual learning is used by the keyboard to locate keys, although slight variations have an impact on the output that is displayed [2,14].

Nevertheless, once the hand is very close to the keyboard, the shadow vanishes and the camera is unable to see it, therefore the desired outcome is not accomplished. The shadow analysis technique was applied in another paper for gesture identification. regulates the use of a virtual keyboard and determines the precision of input recognition [1]. It is challenging to consistently acquire the appropriate results because the finger must be held perpendicular to the keyboard to detect a click. compares the frequency of mistakes and the space between the keys across four different keyboard layouts using Fitts' Law to generate a graph. According to tests, the Tesseract device is accurate, but when the camera is angled, it gives inaccurate findings.

A system has been created that can recognize hand movements in depth and be used to operate home appliances. The device employs a depth camera to identify dynamic hand gestures by recognising hand trajectories and static postures. The characteristics of the hand are collected and utilised to classify the hand's location in static gesture recognition. A different method of capturing the 3D hand motion trajectory employed a depth camera [4]. The strategy uses k-NN classification and dynamic time warping (DTW) techniques. Due to the combination's simplicity of use and adaptability to different user types and gesture kinds, it was picked. The applicability of these techniques to virtual keyboards is unknown [4, 10].

This study will demonstrate and analyze a revolutionary virtual keyboarding method that does not require additional hardware to allow for one-handed typing. The limitations of technology and the further work required to improve it will then be discussed [16].

PROPOSED SYSTEM

We employed an HP laptop with a webcam for our experiment. This implementation can be tested on any device that has a front-facing camera, as well as the image processing and OpenCV libraries installed. Our system will not function or produce the desired output if OpenCV and Python are not installed on your laptop. After the setting is complete, the virtual keyboard, which is now ready to recognize keys, is shown.

The user performs a proper click by first moving the fingertip inward and then outward. The web camera only recognizes a gesture as a click if there is a discernible difference between the two consecutive frames. The center of each key is marked in green when a correct click is made. The distance between the click's area and its centre is determined to make sure that the key whose coordinates cover the click is recognized and displayed.

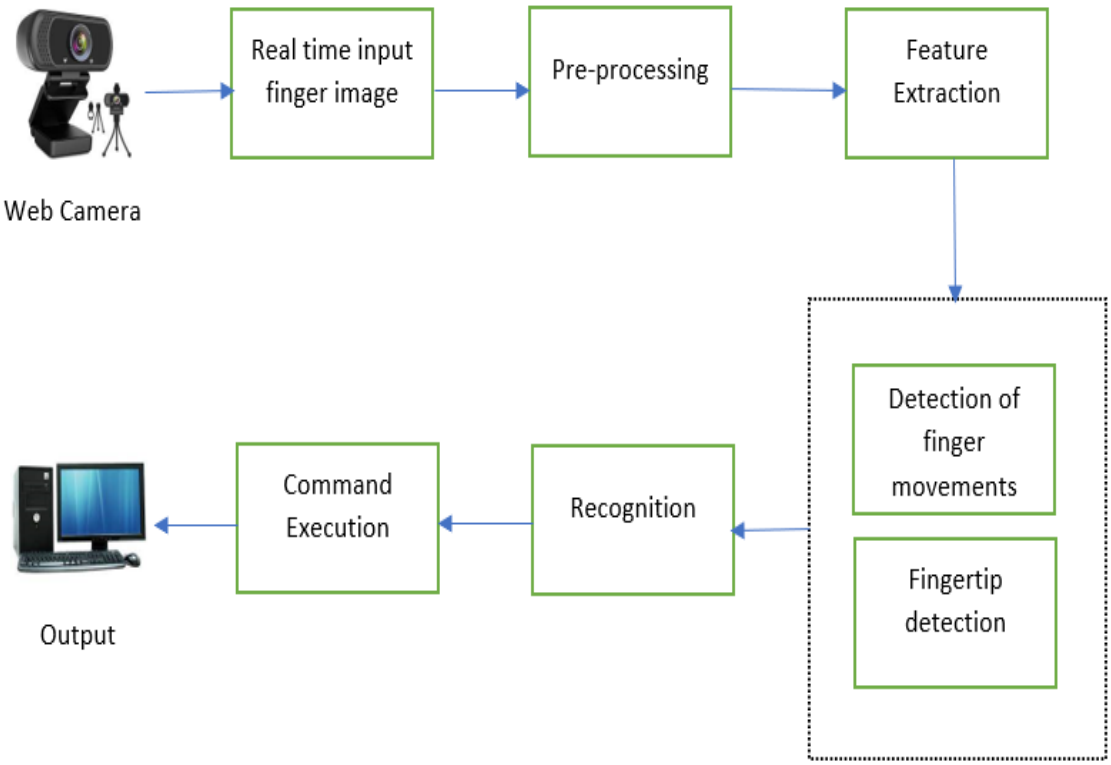
The algorithm we employed to produce this is as follows: We provide a method to tell the foreground from the backdrop before setting up the out-file for creating a movie. You should also know how to determine the Euclidean distance between any two points. Now, type a letter while looking at the virtual keyboard and reading a camera frame. The webcam captures the frame, which is then examined. Locate the contour of the click gesture if there is a noticeable contour difference; otherwise, simply click the keyboard alphabet again.

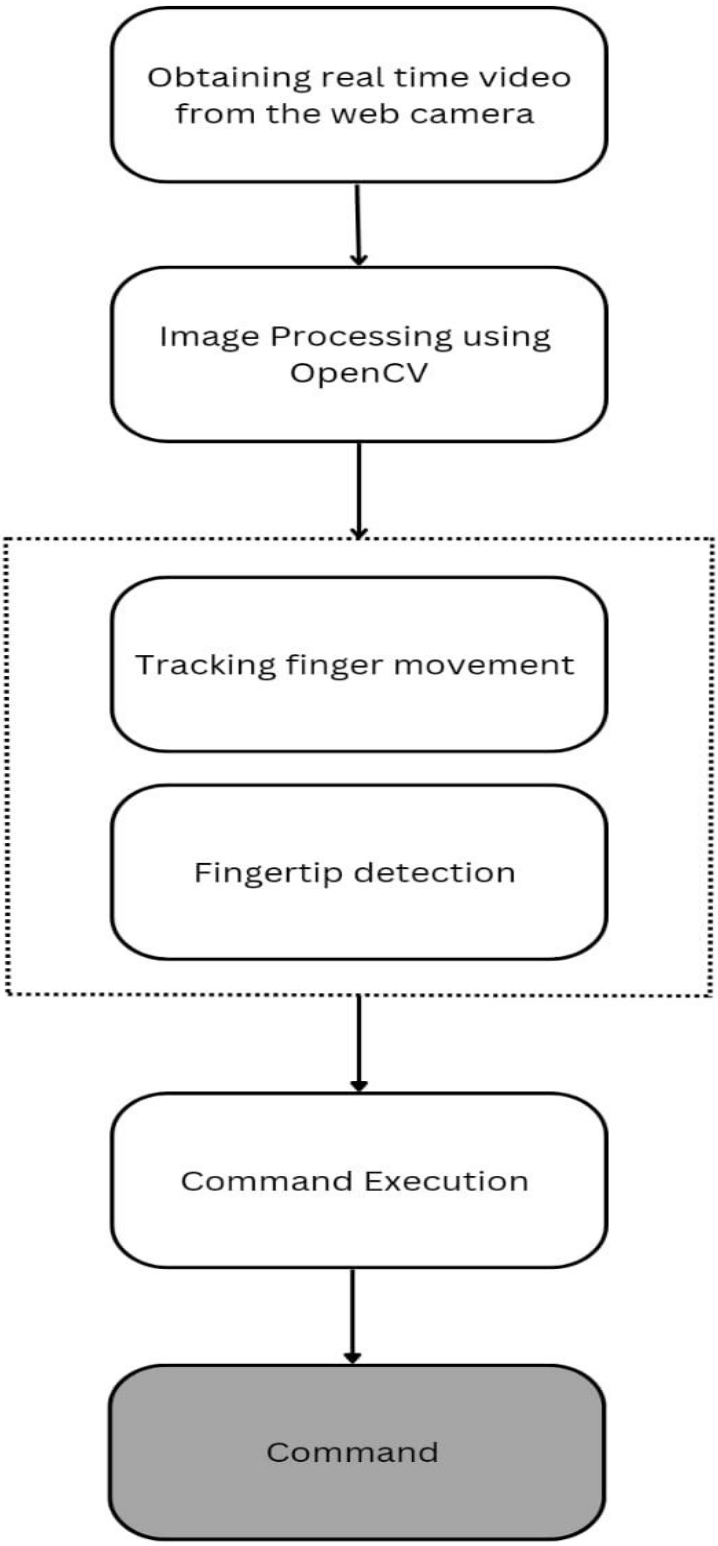
If a letter has been selected for 10 frames, write the letter and clear the buffer. Our project identifies the swiping action that we have enabled in our system and can click numbers, alphabets, and other characters on the keyboard. This is because our virtual keyboard has a long, straight row that can detect fingertips.

I. ARCHITECTURE DIAGRAM

The architecture of the proposed system majorly comprises of the three phases – Camera live feed, Image processing using OpenCV and fingertip detection

In fig 1, the project's operational procedure is shown using a data flow diagram. Initially, the real-time video is captured using the web camera, and then it is preprocessed using OpenCV. Then the finger movement is tracked and the fingertip is detected. Based on the detection, the command is executed as an output.




Obtaining real time video from the web camera
Image Processing using OpenCV
Tracking finger movement
Fingertip detection
Command Execution
Command

II. METHODOLOGY

A) IMAGE PROCESSING USING OPENCV

The manipulation of digital images utilizing a digital computer is the subject of image processing. The creation of a computer system that can process images is the primary focus of image processing. The system accepts a digital image as input, processes it with efficient algorithms, and outputs an image.

We used OpenCV in our research since it is a substantial open-source toolkit for image processing, machine learning, and computer vision. OpenCV supports a wide number of programming languages, including Python, C++, Java, and many more. It can distinguish objects, faces, and even human handwriting by analyzing images and videos.

We've integrated it along the highly optimized library for numerical operations known as NumPy. OpenCV can be used to combine any Numpy-based operations.

While RGB is the standard color code, BGR (Blue Green Red) is the one OpenCV employs.

B) FINGER TIP DETECTION

The fingertip plays a vital role in this project. It is used for typing purposes to obtain the output. The typing is not done by touching the screen directly.

A green-colored insulation tape is stuck on the fingertip of the user’s hand. The user moves his finger above the letter he wants to type. The web camera captures the movement of the finger and types the letters accordingly. The output can be obtained in a notepad, MS Word, or any other platform where typing is involved. Depending on the needs of the user, the keyboard's layout can be modified.

C) DATA PREPROCESSING

Any sort of processing done on raw data to prepare it for a subsequent phase of data processing is referred to as data preprocessing, which is a subset of data preparation.

Data preprocessing changes the data's structure so that data mining, machine learning, and other data science projects can process it more quickly and efficiently. To guarantee accurate results, the techniques are frequently used right at the start of the machine learning and AI development pipeline.

In our project, we dealt with mismatched data types, mixed data values, data outliners, missing data, and other issues using date preprocessing. It makes it easier to get precise results.

III. ADVANTAGES OF THE PROPOSED SYSTEM

Regardless of whether a device has a touch screen or not, our software can be used on any device. Due to the system's exclusive reliance on fingertip detection using a web camera as opposed to a laser, it is less expensive and useful for a variety of users throughout the world. The layout of the keyboard can be changed to suit the user's requirements.

RESULTS AND OUTCOMES

Although the virtual keyboard's design cannot be completely random, some customization is possible. If a user generates an extremely random keyboard, the recognition accuracy will partially decline. The virtual keyboard can also output the recognition result of a customized keyboard to provide users the opportunity to edit their work until it is satisfactory.

SOURCE CODE

CODE:

```
import cv2 as cv
import numpy as np
import imutils
import json
import pyautogui
import time
cam = cv.VideoCapture(0)
arr=[]
nums=["1","2","3","4","5","6","7","8","9","0"]
row1=["Q","W","E","R","T","Y","U","I","O","P"]
row2=["A","S","D","F","G","H","J","K","L"]
row3=["Z","X","C","V","B","N","M"]
row4=["space","enter","backspace","shift"]
row5=["left","up","down","right"]
row6=["volumeup","volumedown","volumemute"]
x=10
y=20
for i in range(0,10):
    data={}
    data["x"]=x
    data["y"]=y
    data["w"]=100
    data["h"]=80
    data["value"]=nums[i]
    arr.append(data)
```

```
    x=x+100
y=100
x=10
for i in range(0,10):
    data={}
    data["x"]=x
    data["y"]=y
    data["w"]=100
    data["h"]=80
    data["value"]=row1[i]
    arr.append(data)
    x=x+100
x=110
y=180
for i in range(0,9):
    data={}
    data["x"]=x
    data["y"]=y
    data["w"]=100
    data["h"]=80
    data["value"]=row2[i]
    arr.append(data)
    x=x+100
x=210
y=260
for i in range(0,7):
    data={}
    data["x"]=x
    data["y"]=y
```

```
    data["w"]=100
    data["h"]=80
    data["value"]=row3[i]
    arr.append(data)
    x=x+100
x=110
y=340
data={}
data["x"]=x
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row4[0]
arr.append(data)
data={}
data["x"]=310
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row4[1]
arr.append(data)
data={}
data["x"]=510
data["y"]=340
data["w"]=250
data["h"]=80
data["value"]=row4[2]
arr.append(data)
data={}
```

```
data["x"]=760
data["y"]=340
data["w"]=200
data["h"]=80
data["value"]=row4[3]
arr.append(data)
x=110
y=420
data={}
data["x"]=x
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row5[0]
arr.append(data)
data={}
data["x"]=310
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row5[1]
arr.append(data)
data={}
data["x"]=510
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row5[2]
arr.append(data)
```

```
data={}
data["x"]=710
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row5[3]
arr.append(data)
x=10
y=500
data={}
data["x"]=x
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row6[0]
arr.append(data)
data={}
data["x"]=210
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row6[1]
arr.append(data)
data={}
data["x"]=410
data["y"]=y
data["w"]=200
data["h"]=80
data["value"]=row6[2]
```

```
arr.append(data)

json_string=json.dumps(arr)
json_data=json.loads(json_string)
#print(json_data)
while(1):
    ret,img = cam.read()
    img = cv.GaussianBlur(img,(5,5),0)
    img=imutils.resize(img,width=1030,height=700)
    height,width=img.shape[:2]
    x=10
    y=20
    for i in range(0,10):
        cv.rectangle(img,(x,y),(x+100,y+80),(0,255,255),2)

cv.putText(img,nums[i],(x+50,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,2
55),2,cv.LINE_AA,False)
        x=x+100
    y=100
    x=10
    for i in range(0,10):
        cv.rectangle(img,(x,y),(x+100,y+80),(0,255,255),2)

cv.putText(img,row1[i],(x+50,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,2
55),2,cv.LINE_AA,False)
        x=x+100
    x=110
    y=180
    for i in range(0,9):
```

```
        cv.rectangle(img,(x,y),(x+100,y+80),(0,255,255),2)

cv.putText(img,row2[i],(x+50,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,2
55),2,cv.LINE_AA,False)
        x=x+100


    x=210
    y=260
    for i in range(0,7):
        cv.rectangle(img,(x,y),(x+100,y+80),(0,255,255),2)

cv.putText(img,row3[i],(x+50,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,2
55),2,cv.LINE_AA,False)
        x=x+100
    x=110
    y=340
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,row4[0],(x+70,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,
255),2,cv.LINE_AA,False)
    x=310
    y=340
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,row4[1],(x+70,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,
255),2,cv.LINE_AA,False)
    x=510
    y=340
    cv.rectangle(img,(x,y),(x+250,y+80),(0,255,255),2)
```

```
cv.putText(img,row4[2],(x+70,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2,cv.LINE_AA,False)
    x=760
    y=340
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,row4[3],(x+70,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2,cv.LINE_AA,False)
    x=110
    y=420
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,row5[0],(x+100,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2,cv.LINE_AA,False)
    x=310
    y=420
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,row5[1],(x+100,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2,cv.LINE_AA,False)
    x=510
    y=420
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,row5[2],(x+100,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2,cv.LINE_AA,False)
    x=710
    y=420
```

```
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,row5[3],(x+100,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0
,255),2,cv.LINE_AA,False)
    x=10
    y=500
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,"V+",(x+60,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,255
),2,cv.LINE_AA,False)
    x=210
    y=500
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)
    cv.putText(img,"V-
",(x+60,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2,cv.LINE_AA,Fa
lse)
    x=410
    y=500
    cv.rectangle(img,(x,y),(x+200,y+80),(0,255,255),2)

cv.putText(img,"Vmute",(x+60,y+40),cv.FONT_HERSHEY_SIMPLEX,1,(0,0,
255),2,cv.LINE_AA,False)

    hsv_img = cv.cvtColor(img,cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv_img,np.array([65,60,60]),np.array([80,255,255]))
    mask_open = cv.morphologyEx(mask,cv.MORPH_OPEN,np.ones((5,5)))
    mask_close =
cv.morphologyEx(mask_open,cv.MORPH_CLOSE,np.ones((20,20)))
    mask_final = mask_close
```

```
    conts,_ =
cv.findContours(mask_final.copy(),cv.RETR_EXTERNAL,cv.CHAIN_APP
RO X_SIMPLE)
    #cv.drawContours(img,conts,-1,(255,0,0),3)
    if(len(conts)==1):
        x,y,w,h =
        cv.boundingRect(conts[0]
        )cx = round(x+w/2)
        cy = round(y+h/2)
        cv.circle(img,(cx,cy),20,(
        0,0,255),2)word=""
        for i in range(len(json_data)):
            if cx>=int(json_data[i]["x"]) and
cx<=int(json_data[i]["x"])+int(json_data[i]["w"])
andcy>=int(json_data[i]["y"]) and
cy<=int(json_data[i]["y"])+int(json_data[i]["h"]):
                word=json_data[
        i]["value"]
        #print(word)
        pyautogui.press(word
        )
        #pyautogui.PAUSE=
        2.5 #time.sleep(1)

    cv.imshow(
    "cam",img)
    cv.waitKey(
    10)
```

CONCLUSION

In order to complete this project, we used a camera and image processing to create a virtual keyboard. By implementing a virtual keyboard system, we were able to obtain a better grasp of the tradeoffs to take into account when deciding on image analysis methods for a bigger system. To determine whether a virtual keyboard could make it easier for users to do their existing typing duties on touch displays, we need to learn what are those tasks.

It is possible to construct a device that is even better and more user-friendly by expanding on the fundamental concept of building a virtual keyboard. The current idea can be modified in the short term in the following ways:

Speed Improvement: By porting the entire image processing code to C or .NET, the device's operating speed can be improved.

Support for multi-touch: Only one touch is supported by our implementation. By using a variety of colors for tags and color image processing, it can be made multi-touch. For better operation, more advanced algorithms for image processing will be required.

Support for Android: By developing a separate app that acts as a device driver, the virtual keyboard can be adapted for Android-based devices. However, as most mobile or tablet processors are slower than those of PCs, additional research is required before this can be implemented. On a device that already lacks resources, image processing could consume valuable resources.

Using the DSP processor as an interface: For image processing, a DSP processor can be incorporated into the device itself. This would solve the issue raised in the previous point, but it would cost more money to buy the gadget.