

How to Develop a Virtual Keyboard Using OpenCV

[ADVANCED](#)[COMPUTER VISION](#)[PROJECT](#)[TECHNIQUE](#)

This article was published as a part of the [Data Science Blogathon](#)

Introduction

[OpenCV](#) is the most popular library for the task of computer vision, it is a cross-platform open-source library for machine learning, image processing, etc. using which real-time computer vision applications are developed.

CVzone is a computer vision package, where it uses OpenCV and MediaPipe libraries as its core that makes us easy to run like hand tracking, face detection, facial landmark detection, pose estimation, etc., and also image processing and other computer vision-related applications. Check [here](#) for more information.

Implementation of Virtual Keyboard Using OpenCV

Let us create a virtual Keyboard.

First, let us install the required modules.

```
---> pip install numpy ---> pip install opencv-python ---> pip install cvzone ---> pip install pynput
```

Import Libraries for Virtual Keyboard Using OpenCV

Now let's import the required modules

```
import cv2 import cvzone from cvzone.HandTrackingModule import HandDetector from time import sleep import numpy as np from pynput.keyboard import Controller
```

Here we are importing the HandDetector module from cvzone.HandTrackingModule and then in order to make the virtual keyboard work we need to import Controller from pynput.keyboard.

```
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW) cap.set(3, 1280) cap.set(4, 720)
```

Now let's take real-time input from cv2.VideoCapture

```
detector = HandDetector(detectionCon=0.8) keyboard_keys = [["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"], ["A", "S", "D", "F", "G", "H", "J", "K", "L", ";"], ["Z", "X", "C", "V", "B", "N", "M", ",", ".", "/"]]
```

```
final_text = ""
```

We initialize HandDetector with detection confidence of 0.8 and assign it to the detector. Then we create an array of lists according to the layout of our keyboard and define an empty string to store the typed keys.

Defining Draw Function

```
keyboard = Controller()
```

```
def draw(img, buttonList): for button in buttonList: x, y = button.pos w, h = button.size
cvzone.cornerRect(img, (button.pos[0], button.pos[1], button.size[0],button.size[0]), 20 ,rt=0)
cv2.rectangle(img, button.pos, (int(x + w), int(y + h)), (255, 144, 30), cv2.FILLED) cv2.putText(img,
button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4) return img
```

Initialize the keyboard controller, and define a function with name *draw()* and it takes two arguments that is an image and the buttonList and return the image. Here Inside the *draw()* function, we are using cvzone's *cornerRect* function to draw rectangle edges at the corner of each keys. It is in order to make our keyboard layout look better. It will look something like the below images.



You can also try changing different colours.

```
class Button(): def __init__(self, pos, text, size=[85, 85]): self.pos = pos self.size = size self.text =
text
```

Then we define a class called Button() and we give position, text and size as the inputs so that we can arrange the keyboard keys in a well-defined order.

```
buttonList = [] # mybutton = Button([100, 100], "Q") for k in range(len(keyboard_keys)): for x, key in
enumerate(keyboard_keys[k]): buttonList.append(Button([100 * x + 25, 100 * k + 50], key))
```

The above loop will loop through the keyboard keys and Button objects where we give position and text as inputs are appended in a list called button list. Later we can pass this list to draw function to draw on top of our real-time frame.

Main Program for Virtual Keyboard Using OpenCV

Here comes the important part.

```
while True: success, img = cap.read() img = detector.findHands(img) lmList, bboxInfo =
detector.findPosition(img) img = draw(img, buttonList) # change the draw funtion to transparent_layout for
transparent keys if lmList: for button in buttonList: x, y = button.pos w, h = button.size if x < lmList[8]
[0]<x+w and y < lmList[8][1] < y+h: cv2.rectangle(img, button.pos, (x + w, y + h), (0, 255, 255), cv2.FILLED)
cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4) l, _, _ =
detector.findDistance(8,12, img, draw=False) print(l) if l < 25: keyboard.press(button.text)
cv2.rectangle(img, button.pos, (x + w, y + h), (0, 255, 0), cv2.FILLED) cv2.putText(img, button.text, (x +
```

```

20, y + 65), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4) final_text += button.text sleep(0.20)
cv2.rectangle(img, (25,350), (700, 450), (255, 255, 255), cv2.FILLED) cv2.putText(img, final_text, (60, 425),
cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4) # cv2.rectangle(img, (100,100), (200,200), # (100, 255, 0),
cv2.FILLED) # cv2.putText(img, 'Q', (120,180), cv2.FONT_HERSHEY_PLAIN, 5, # (0, 0, 0), 5) # img =
mybutton.draw(img) cv2.imshow("output", img) cv2.waitKey(1)

```

Inside the while loop the main function takes place, first we read the real-time input frames and store it in a variable called *img*. Then we pass that image to the *detector.findHands()* in order to find the hand in the frame. Then in that image, we need to find the position and bounding box information of that detected hand.

Here we can find the distance between the top point of our index finger and middle finger, if the distance between the two is less than a certain threshold, then we can type the letter on which we are indicating. Once we get the position then we loop through the entire position list. From that list, we find button position and button size and then we plot it on the frame according to a well-defined manner.

Image 1: Hand Landmark Model

After that, we need to find the distance between the top point of our index finger and middle finger. In the above image, you can see the top points which we require are point 8 and point 12. Hence we need to pass 8, 12 inside a distance finding function in order to get the distance between them. In the above code you can see *detector.findDistance()* and there we passed 8, 12, and image in order to find the distance and we set the draw flag to false so that we do not need any line between the two points.

If the distance between the points is very less we will use *press()* function to press the keys. In the above code *keyboard.press()* and we are passing *button.text* in order to display that pressed key. And finally, we draw a small white rectangular box just below our keyboard layout in order to display the pressed key.

Once you execute the whole code it looks something like this.

After you bring the index finger and middle finger close to each other on top of a particular letter, you can type that letter.

If you need the keyboard layout to be more customized, we can make the keyboard layout transparent. We just need to add a transparent layout function and replace the *draw()* function with *transparent_layout()* function.

Let us define the *transparent_layout()* function. Below is the function, it takes the same input as that of the *draw()* function. Here we assign a numpy's *zero_like()* function to a variable called *imgNew* and perform the desired operation on that, like having the corner rectangle, creating the rectangle box for each key, and putting the text inside the box. After that, we copy that image to a new variable and create a mask of *imgNew* and we use OpenCV's *addWeighted()* function to place the mask on top of the actual image. Hence this makes the keyboard layout to be transparent.

Customizing the Keyboard

```
def transparent_layout(img, buttonList): imgNew = np.zeros_like(img, np.uint8) for button in buttonList: x, y = button.pos cvzone.cornerRect(imgNew, (button.pos[0], button.pos[1], button.size[0],button.size[0]), 20,rt=0) cv2.rectangle(imgNew, button.pos, (x + button.size[0], y + button.size[1]), (255, 144, 30), cv2.FILLED) cv2.putText(imgNew, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4)
```

```
out = img.copy()
alpaha = 0.5
mask = imgNew.astype(bool)
```

```
print(mask.shape)
```

```
out[mask] = cv2.addWeighted(img, alpaha, imgNew, 1-alpaha, 0)[mask] return out
```

Once you replace the *draw()* function inside while loop with *transparent_layout()* function it will look like this. (below image)

Entire Code for Virtual Keyboard Using OpenCV

Below is the entire code

```
import cv2
import cvzone
from cvzone.HandTrackingModule import HandDetector
from time import sleep
import numpy as np
from pynput.keyboard import Controller
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
cap.set(3, 1280)
cap.set(4, 720)
detector = HandDetector(detectionCon=0.8)
keyboard_keys = ["Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P"], ["A", "S", "D", "F", "G", "H", "J", "K", "L", ";"], ["Z", "X", "C", "V", "B", "N", "M", ",", ".", "/"]
final_text = ""
keyboard = Controller()

def draw(img, buttonList):
    for button in buttonList:
        x, y = button.pos
        w, h = button.size
        cvzone.cornerRect(img, (button.pos[0], button.pos[1], button.size[0], button.size[0]), 20, rt=0)
        cv2.rectangle(img, button.pos, (int(x + w), int(y + h)), (255, 144, 30), cv2.FILLED)
        cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4)
    return img

def transparent_layout(img, buttonList):
    imgNew = np.zeros_like(img, np.uint8)
    for button in buttonList:
        x, y = button.pos
        cvzone.cornerRect(imgNew, (button.pos[0], button.pos[1], button.size[0], button.size[0]), 20, rt=0)
        cv2.rectangle(imgNew, button.pos, (x + button.size[0], y +
```

```

button.size[1]), (255, 144, 30), cv2.FILLED) cv2.putText(imgNew, button.text, (x + 20, y + 65),
cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4) out = img.copy() alpaha = 0.5 mask = imgNew.astype(bool)
print(mask.shape) out[mask] = cv2.addWeighted(img, alpaha, imgNew, 1-alpaha, 0)[mask] return out
class Button():
    def __init__(self, pos, text, size=[85, 85]):
        self.pos = pos
        self.size = size
        self.text = text
buttonList = []
# mybutton = Button([100, 100], "Q")
for k in range(len(keyboard_keys)):
    for x, key in enumerate(keyboard_keys[k]):
        buttonList.append(Button([100 * x + 25, 100 * k + 50], key))
while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bboxInfo = detector.findPosition(img)
    img = draw(img, buttonList)
    # change the draw funtion to transparent_layout for transparent keys
    if lmList:
        for button in buttonList:
            x, y = button.pos
            w, h = button.size
            if x < lmList[8][0] < x+w and y < lmList[8][1] < y+h:
                cv2.rectangle(img, button.pos, (x + w, y + h), (0, 255, 255), cv2.FILLED)
                cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4)
            l, _, _ = detector.findDistance(8,12, img, draw=False)
            print(l)
            if l < 25:
                keyboard.press(button.text)
                cv2.rectangle(img, button.pos, (x + w, y + h), (0, 255, 0), cv2.FILLED)
                cv2.putText(img, button.text, (x + 20, y + 65), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4)
            final_text += button.text
            sleep(0.20)
            cv2.rectangle(img, (25,350), (700, 450), (255, 255, 255), cv2.FILLED)
            cv2.putText(img, final_text, (60, 425), cv2.FONT_HERSHEY_PLAIN, 4, (0, 0, 0), 4)
            # cv2.rectangle(img, (100,100), (200,200), # (100, 255, 0), cv2.FILLED)
            # cv2.putText(img, 'Q', (120,180), cv2.FONT_HERSHEY_PLAIN, 5, # (0, 0, 0), 5)
            # img = mybutton.draw(img)
cv2.imshow("output", img)
cv2.waitKey(1)

```

Conclusion

This is the implementation of the virtual keyboard, if you want to take it to the next step you can also all the keypress sounds and then we can also make the keyboard layout move within the frames.

Hope you enjoyed it.

[Reference](#)

Image 1: <https://google.github.io/mediapipe/solutions/hands.html>

My [LinkedIn](#)

Thank You

The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.

Article Url - <https://www.analyticsvidhya.com/blog/2021/09/develop-a-virtual-keyboard-using-opencv/>



[Syed Abdul Gaffar Shakhadri](#)