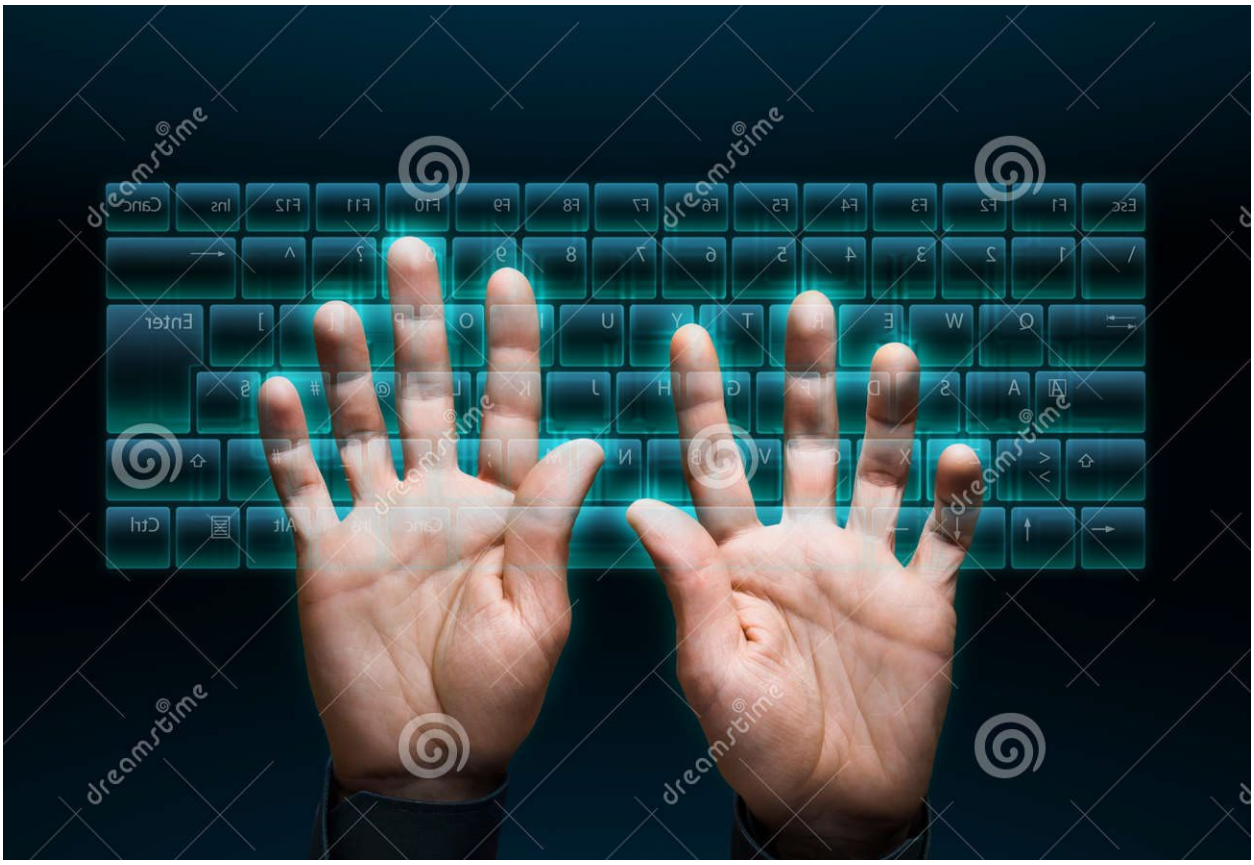


Creating an Artificial Intelligence Virtual Keyboard

Documentation

Kirtan Patel

Hao Loi (Faculty Sponsor)

Department of Computer Science, Quinsigamond Community College

GitHub: https://github.com/KirPatel/AI-Virtual-keyboard

Project basic Needs:

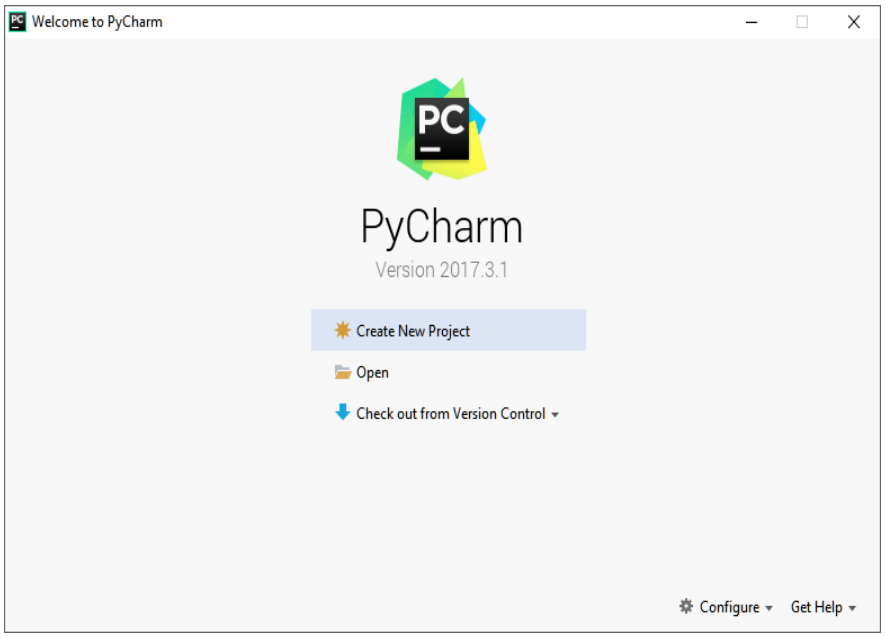
1. Laptop or PC with 8 to 16gb RAM.
2. Built-in Camera or Webcam

Python:

1. To download and install Python, visit the official website of Python https://www.python.org/downloads/ and choose your version. We have chosen Python version 3.7
2. Once the download is completed, run the .exe file to install Python. Now click on Install Now.
3. You can see Python installing at this point.
4. When it finishes, you can see a screen that says the Setup was successful. Now click on “Close”.

PyCharm:

1. To download PyCharm visit the website https://www.jetbrains.com/pycharm/download/ and Click the "DOWNLOAD" link under the Community Section.
2. Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click "Next".
3. On the next screen, Change the installation path if required. Click "Next".
4. On the next screen, you can create a desktop shortcut if you want and click on "Next".
5. Choose the start menu folder. Keep selected JetBrains and click on "Install".
6. Wait for the installation to finish.
7. Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the "Run PyCharm Community Edition" box first and click "Finish".
8. After you click on "Finish," the Following screen will appear.



OpenCV:

1. Go to the terminal option at the bottom of the IDE window.
2. The pip (package manager) can also be used to download and install OpenCV. To install OpenCV, just type the following command:

   pip install opencv-python

3. Now simply import OpenCV in your python program in which you want to use image processing functions.

Mediapipe:

1. Make sure that Bazel and OpenCV are correctly installed and configured for MediaPipe. Please see Installation for how to setup Bazel and OpenCV for MediaPipe on Linux and macOS.

2. Windows: Download the latest protoc win64 zip from the Protobuf GitHub repo, unzip the file, and copy the protoc.exe executable to a preferred location. Please ensure that location is added into the Path environment variable.

3. Activate a Python virtual environment.

   $ python3 -m venv mp env && source m.env.bin/activate

4. In the virtual environment, go to the MediaPipe repo directory.

5. Install the required Python packages.

   (mp.env)mediapipe$ pip3 install -r requitements.txt

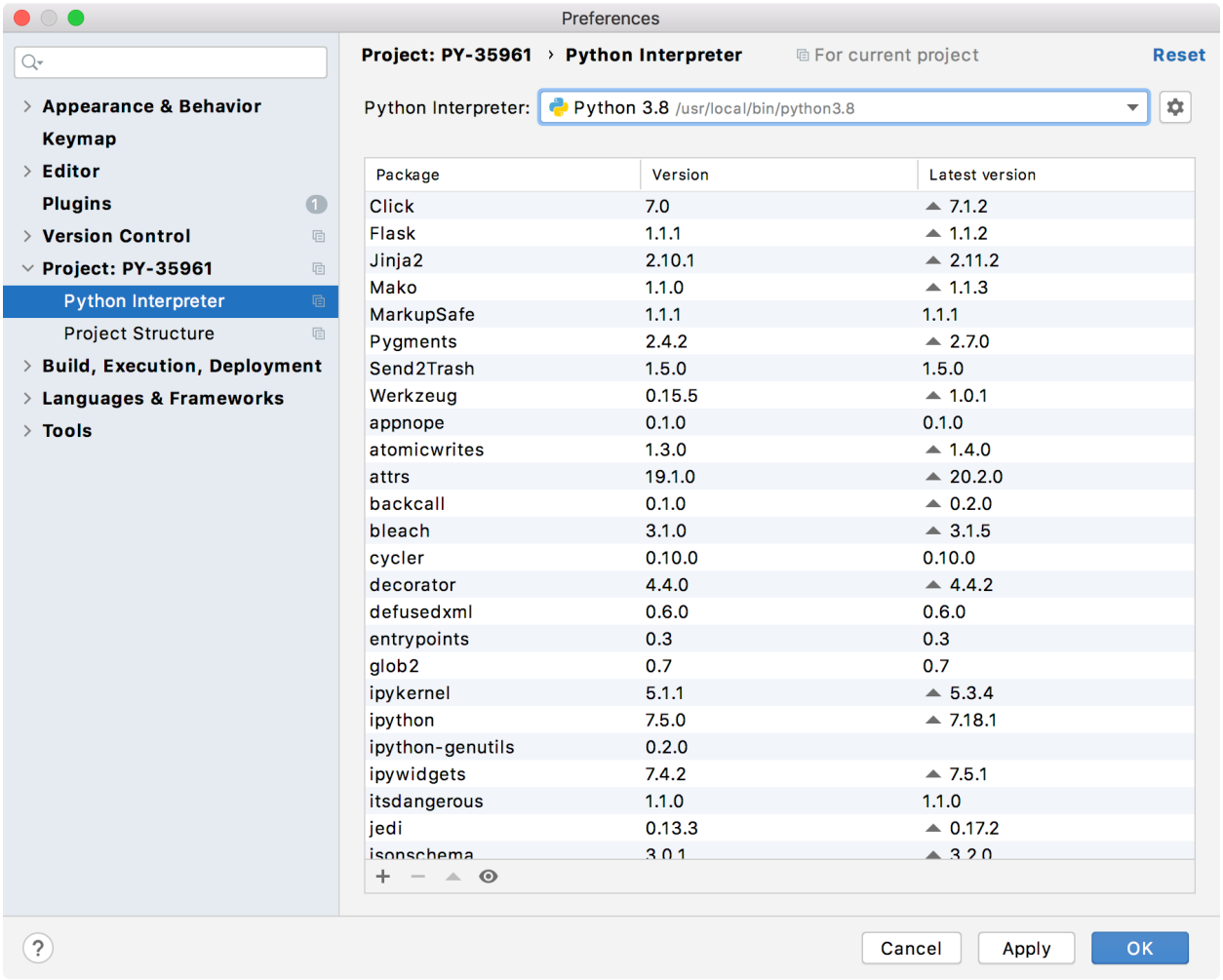
6. Generate and install MediaPipe package.

   (mp.env)mediapipe$ python3 setup.py gen.protos

   (mp.env)mediapipe$ python3 setup.py install –link-opencv

Pynput library:

1. Open File > Settings > Project from the PyCharm menu.
2. Select your current project.
3. Click the Python Interpreter tab within your project tab.
4. Click the small + symbol to add a new library to the project
5. Now type in the library to be installed (Pynput).
6. For Example:



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 mediapipe

---> pip install pynput
```

Import Libraries for Virtual Keyboard Using OpenCV

Now let’s import the required modules

```
import cv2
import mediapipe
from time import sleep
import numpy as np
from pynput.keyboard import Controller
```

Here we are importing the OpenCV and Mediapipe module 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

```
keyLayout = [["1", "2”, "3", "4", "5", "6", "7", "8", "9", "0", "<"]
        ["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"],
        [" "]]
Empty Array = ""
```

We initialize all keys with creating an array of lists according to the layout of our keyboard and define an empty string to store the typed keys.

Defining Function and Class

```
mpHands = mp.solutions.hands
mpDraw = mp.solutions.drawing_utils
Hands = mpHands.Hands()
keyboard = Controller()
mpDraw = mp.solutions.drawing_utils

class settings():
    def __init__(self, pos, size, text):
        self.pos = pos
        self.size = size
        self.text = text
```

```
def keyboardEdit(img, storedVar):
    for button in storedVar:
        x, y = button.pos
        w, h = button.size


        cv.rectangle(img, button.pos, (x + w, y + h), (0, 0, 0), 5)
        cv.putText(img, button.text, (x + 15, y + 35),
cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 4)


    return img
```

Initialize the keyboard controller, and define a function with name settings() and it takes two arguments that is an image and the keyboardedit and return the image. Here Inside the settings() 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 colors.

```
class settings():
    def __init__(self, pos, size, text):
        self.pos = pos
        self.size = size
```

```
        self.text = text
```

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

```
def keyboardEdit(img, storedVar):
    for button in storedVar:
        x, y = button.pos
        w, h = button.size


        cv.rectangle(img, button.pos, (x + w, y + h), (0, 0, 0), 5)
        cv.putText(img, button.text, (x + 15, y + 35),
cv.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 4)


    return img
```

The above loop will loop through the keyboard Button objects where we give position and text as inputs are appended in a list called stored var. 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()
    imgRGB = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    results = Hands.process(imgRGB)


    printer = []
```

```
    if results.multi_hand_landmarks:
        for hand_in_frame in results.multi_hand_landmarks:
            mpDraw.draw_landmarks(img, hand_in_frame,
mpHands.HAND_CONNECTIONS)

        for id, lm in enumerate(results.multi_hand_landmarks[0].landmark):
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            printer.append([cx, cy])
         if printer:
            for button in arr:
                x, y = button.pos
                w, h = button.size

                if x < printer[8][0] < x + w and y < printer[8][1] < y + h:
                    cv.rectangle(img, (x - 5, y - 5), (x + w + 5, y + h + 5), (0, 0, 255),
cv.FILLED)
                    x1, y1 = printer[8][0], printer[8][1]
                    x2, y2 = printer[12][0], printer[12][1]
                    clicked = math.hypot(x2 - x1 - 30, y2 - y1 - 30)
                    print(clicked)
                    if button.text == "<":
                        keyboard.press(Key.backspace)
                        keyboard.release(Key.backspace)

                    if not clicked > 63:
                        keyboard.press(button.text)
```

```
                cv.rectangle(img, (x - 5, y - 5), (x + w + 5, y + h + 5), (0, 255, 0),
cv.FILLED)
                sleep(0.15)


    img = keyboardEdit(img, arr)
```

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 Hands = mpHands.Hands() 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.

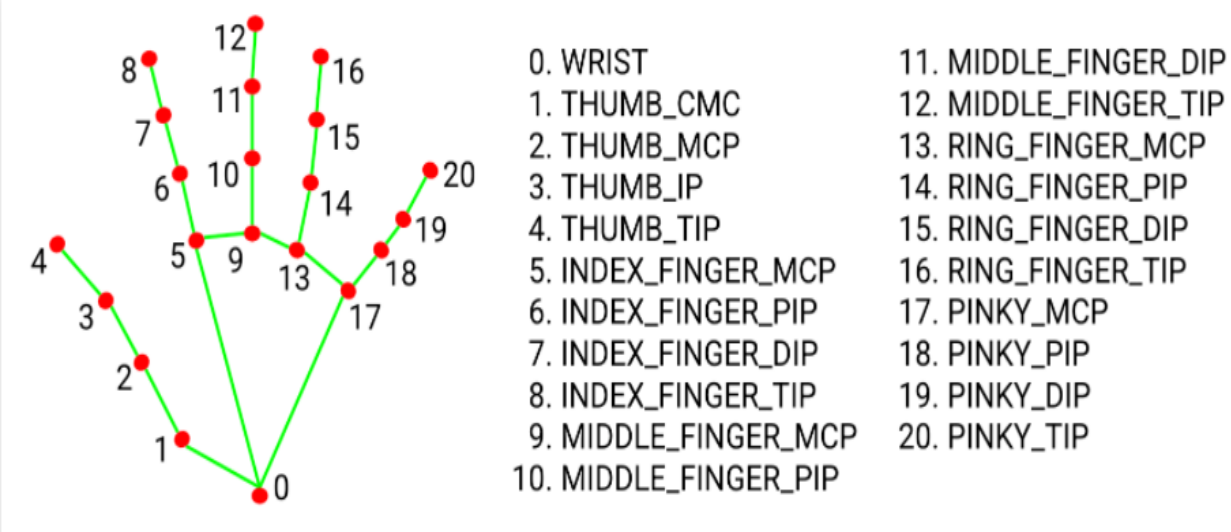
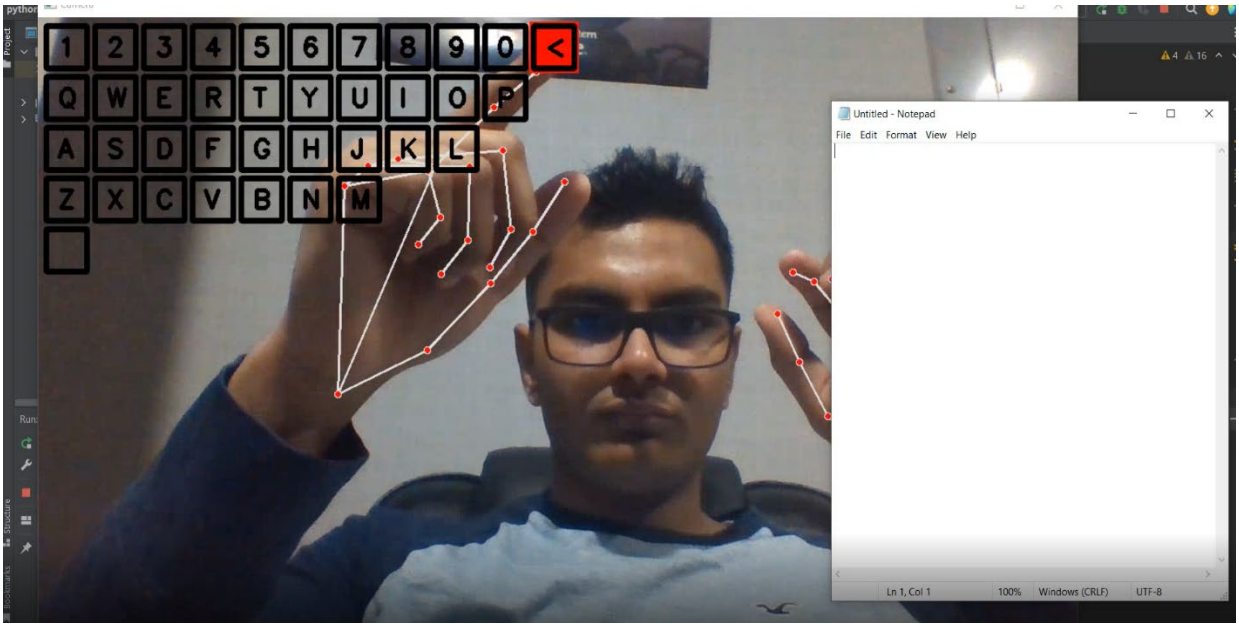


Image 1: Hand Landmark Model

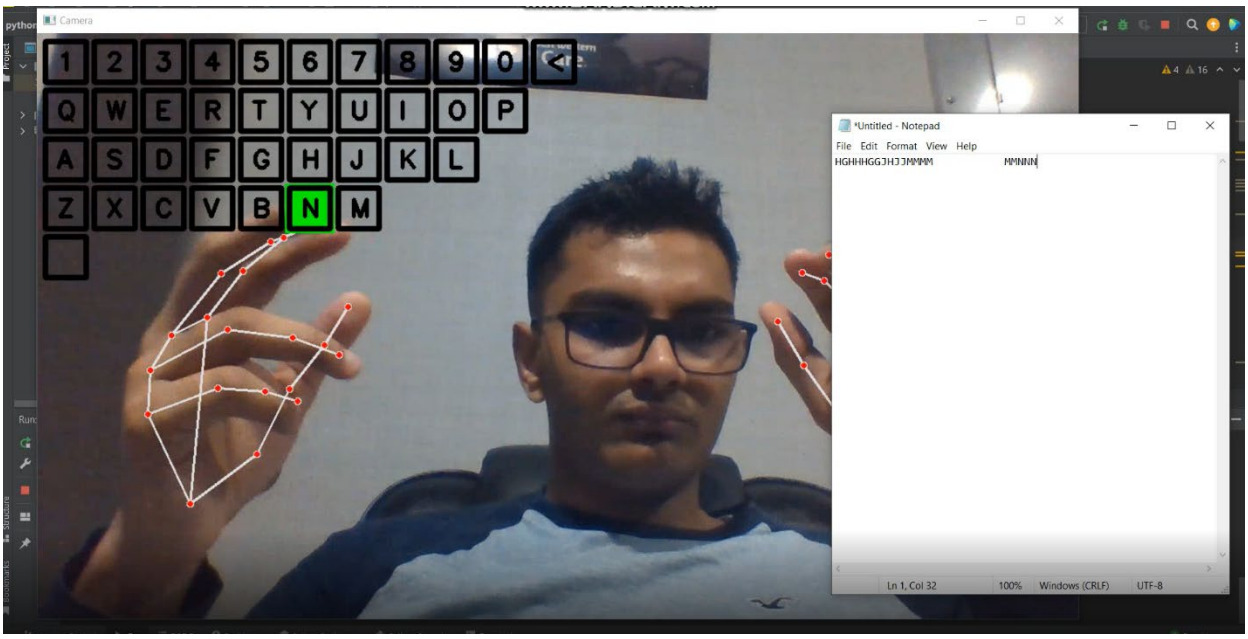
After that, we need to find the distance between the top point of our index finger and middle finger. In the above image, you can see the top points which we require are point 8 and point 12. Hence, we need to pass 8, 12 inside a distance finding function to get the distance between them. In the above code you can see keyboard = Controller() 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 keybaordedit) function to press the keys. In the above code and we are passing button 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 keybaordedit() function with transparent_layout() function.

Work Cited

Ray, A. (2022). How Hand gestures are replacing other computer input systems. *Analytics India Mag*. Retrieved from https://analyticsindiamag.com/how-hand-gestures-are-replacing-other-computer-input-systems/

University of Waterloo. (2022). System recognizes hand gestures to expand computer input on a keyboard. *Science Daily*. Retrieved from https://www.sciencedaily.com/releases/2022/01/220105094430.htm

Chaudhary, P. (2016). User defined custom virtual keyboard. *Arxiv Organization*, pp. 1-11., Retrieved from https://arxiv.org/pdf/1907.13285.pdf

Hernanto, S. (2011). Webcam virtual keyboard. *International Conference on Electrical Engineering and Informatics.* https://www.diva- Retrieved from portal.org/smash/get/diva2:1504621/FULLTEXT01.pdf

Lakshmanamoorthy, R. (2021). Real-time GUI Interactions with OpenCV in Python. *Analytics India Mag*. Retrieved from https://analyticsindiamag.com/real-time-gui-interactions-with-opencv-in-python/