II-B.Tech-CSE-AI-Semester 4,2020-2024

21ICN211` - Introduction To Communication Networks

END-SEMESTER PROJECT

HAND GESTURE OVER RASPBERRY PI

AMRITA VISHWA VIDYAPEETHAM,ETTIMADAI, COIMBATORE.

Submitted by

G.Krishna Rayalu (CB.EN.U4AIE20016)

Jangala Gouthami (CB.EN.U4AIE20023)

N.Manish Rama Gopal (CB.EN.U4AIE20037)

D.Rohith(CB.EN.U4AIE20060)

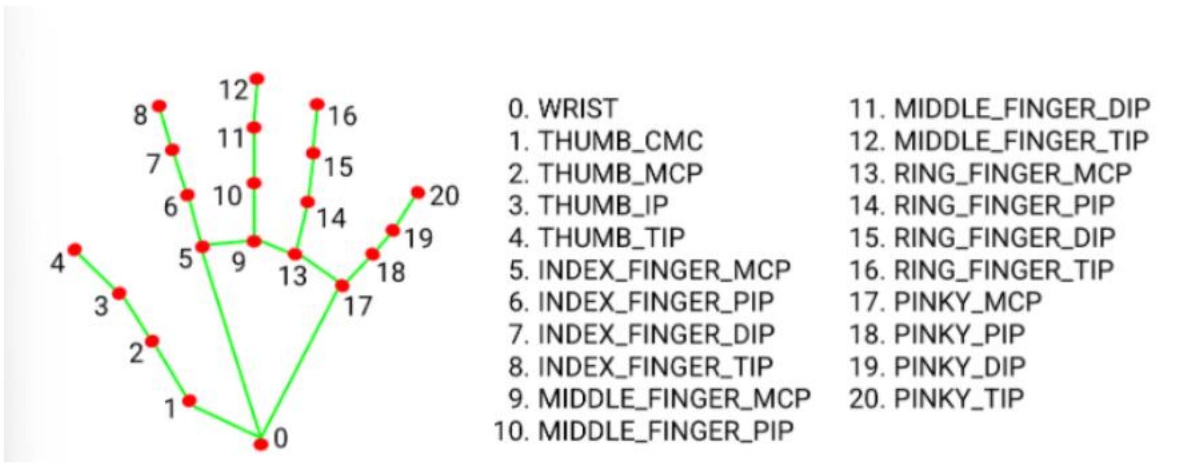
Submitted on : 16-07-2022

Abstract

In this project we have developed a hand gesture recognition device which is used to communicate between a machine and a person. This project is about Real-Time Hand Gesture Recognition and to perform system control operations as designed. This application allows the use of an external camera/webcam to detect the user's hand movements and perform simple operations in response. A clear gesture must be performed by the user for the system to identify. This is captured by the webcam, which recognizes the gesture and performs the action against a set of recognized gestures. This project can be developed in the future to assist deaf people who cannot learn operations by using a computer using this method. The project specifically focuses on gesture recognition using picture signs and perform system control operations as designed.

Introduction

In this project we will be using the hands module of Mediapipe library to perform some basic operations. The hands module creates a 21 points-based localization of our hand. If we supply a hand image/frame to this module, it will return a 21-point vector showing the coordinates of 21 important landmarks present on our hand.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

The points would mean the same irrespective of your input image. This means that point 4 would always be the tip of your thumb, 8 would always be the tip of the index finger. So, once you have the 21-point vector, it’s up to your creativity what kind of project you create.

Pre-Requisites

TCP

The TCP stands for Transmission Control Protocol. If we want the communication between two computers and communication should be good and reliable. For example, we want to view a web page, then we expect that nothing should be missing on the page, or we want to download a file, then we require a complete file, i.e., nothing should be missing either it could be a text or an image. This can only be possible due to the TCP. It is one of the most widely used protocols over the TCP/IP network.

Features of TCP:

- Data delivery
- Connection oriented protocol

UDP

UDP stands for User Datagram Protocol. It is a message-oriented communication protocol, used for communication channels and data paths. UDP uses a connectionless communication model with a minimum of initializing protocol mechanisms.

The protocol is implemented on top of the IP (Internet Protocol), and is collectively called the UDP/IP Network Stack. UDP disowns certain features offered by other similar protocols, such as Handshaking (TLS/SSL) for initializing a connection, no acknowledgment for transmission progress, Data restructuring, etc. in order to offer a fast communication even on low bandwidth networks. Due to these characteristics of UDP it is implemented in VoIP, Web Streaming, Broadcasting, Gaming, etc.

Characteristics

- Connection: Is connectionless protocol, therefore no Handshake is required to establish the connection
- No Acknowledgement: Data is sent unilaterally by the sender, and therefore no interaction of the recipient system is required (ex. sending acknowledgment about the received packets isn’t required).
- Less Overhead: Since no connection is required to establish the protocol, there are no ongoing connections to maintain. Which leads to less overhead.
- Orderless: The data may not arrive in order.
- Speed: Faster than other protocols of the same nature and requires fewer resources to function efficiently.

Difference between TCP and UDP

	TCP	UDP
Full form	It stands for Transmission Control Protocol.	It stands for User Datagram Protocol.
Type of connection	It is a connection-oriented protocol, which means that the connection needs to be established before the data is transmitted over the network.	It is a connectionless protocol, which means that it sends the data without checking whether the system is ready to receive or not.
Reliable	TCP is a reliable protocol as it provides assurance for the delivery of data packets.	UDP is an unreliable protocol as it does not take the guarantee for the delivery of packets.
Speed	TCP is slower than UDP	UDP is faster than TCP
Header size	The size of TCP is 20 bytes.	The size of the UDP is 8 bytes.
Acknowledgment	TCP uses the three-way-handshake concept. In this concept, if the sender receives the ACK, then the sender will send the data. TCP also has the ability to resend the lost data.	UDP does not wait for any acknowledgment; it just sends the data.
Flow control mechanism	It follows the flow control mechanism in which too many packets cannot be sent to the receiver at the same time.	This protocol follows no such mechanism.
Error checking	TCP performs error checking by using a checksum. When the data is corrected, then the data is retransmitted to the receiver.	It does not perform any error checking, and also does not resend the lost data packets.
Applications	This protocol is mainly used where a secure and reliable communication process is required, like military services, web browsing, and e-mail.	This protocol is used where fast communication is required and does not care about the reliability like VoIP, game streaming, video and music streaming, etc.

Procedure

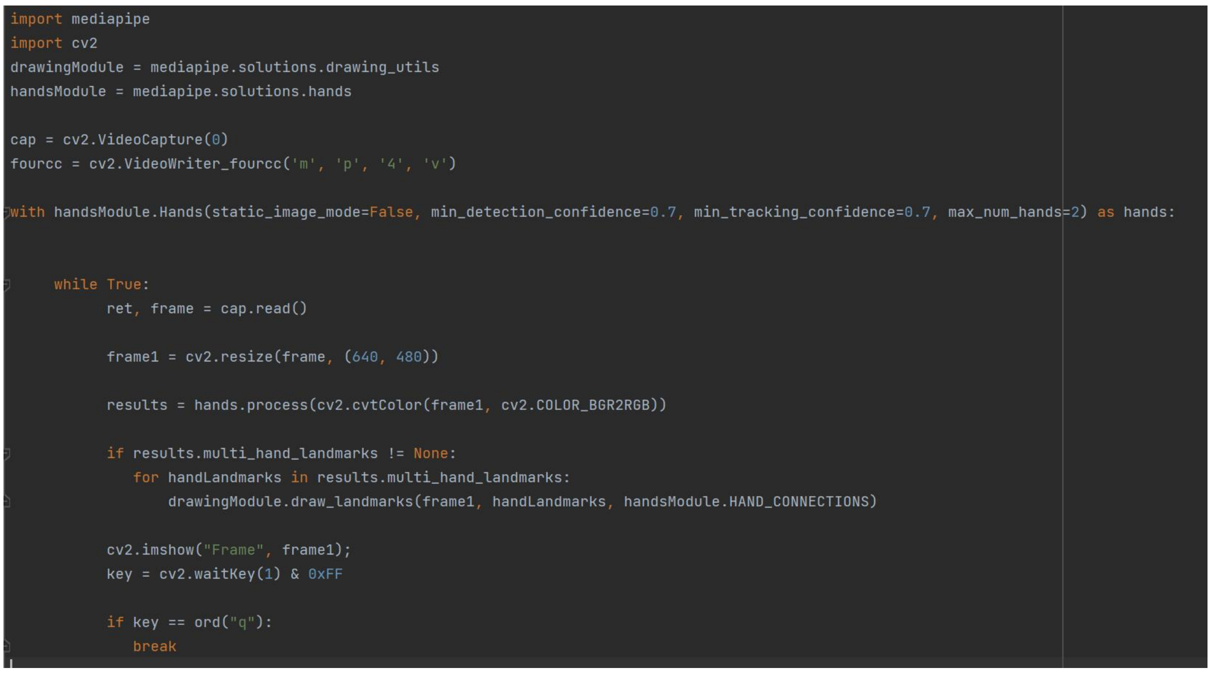
1. Initially we will be installing the various libraries that are required to make the functionality compatible with python environment that is present in the raspberry pi .
2. After completing all the installations, we will be developing our codes that is:
   - The Hand Tracking code
   - The Finger count and Finger up-down recogniser code
   - The computer gesture control code

The Hand Tracking code

This code will help us identify any hands seen in front of it through computer vision and then use machine learning to draw a hand framework over the top of any hands identified.

Working:

- We use Mediapipe to draw the hand framework over the top of hands it identifies in Real-Time.
- CV2 Functionality is then used to create a Video stream and add some values.
- We then add confidence values and extra settings to MediaPipe hand tracking. As we are using a live video stream that is not a static.
- Image mode then comes into picture where confidence values in regards to overall detection and tracking are added and we will only let two hands to be tracked at the same time.
- More hands can be tracked at the same time if desired but will slow down the system
- An infinity loop is then created to produce the live feed to our desktop that will search for hands.
- A frame size of 640 x 480 is used which offers a nice balance between speed and accurate identification.
- We then produce the hand framework overplay on top of the hand.
- Then few line of code are added that are responsible to find the location of index finger.
- The current frame is then updated to the desktop and a key is associated to stop the execution of code.

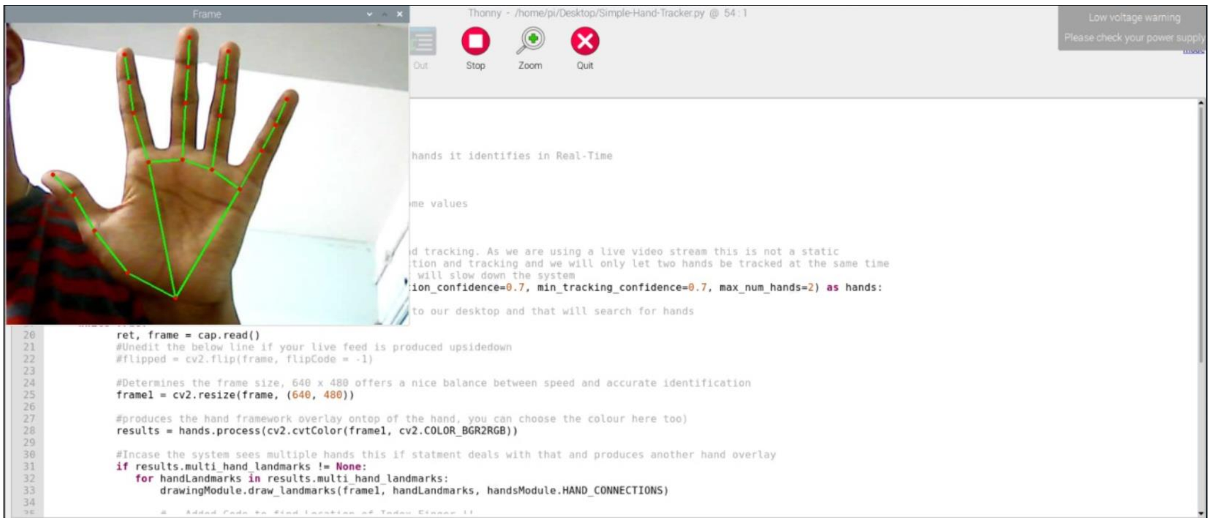


```
import mediapipe
import cv2
drawingModule = mediapipe.solutions.drawing_utils
handsModule = mediapipe.solutions.hands

cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')

with handsModule.Hands(static_image_mode=False, min_detection_confidence=0.7, min_tracking_confidence=0.7, max_num_hands=2) as hands:


     while True:
           ret, frame = cap.read()

           frame1 = cv2.resize(frame, (640, 480))

           results = hands.process(cv2.cvtColor(frame1, cv2.COLOR_BGR2RGB))

           if results.multi_hand_landmarks != None:
              for handLandmarks in results.multi_hand_landmarks:
                  drawingModule.draw_landmarks(frame1, handLandmarks, handsModule.HAND_CONNECTIONS)

           cv2.imshow("Frame", frame1);
           key = cv2.waitKey(1) & 0xFF

           if key == ord("q"):
              break
```

So as soon as we run it;

it will open up a new window and initiate hand identification.

Output


Frame
Thonny - /home/pi/Desktop/Simple-Hand-Tracker.py @ 54 : 1
Out
Stop
Zoom
Quit
Low voltage warning
Please check your power supply

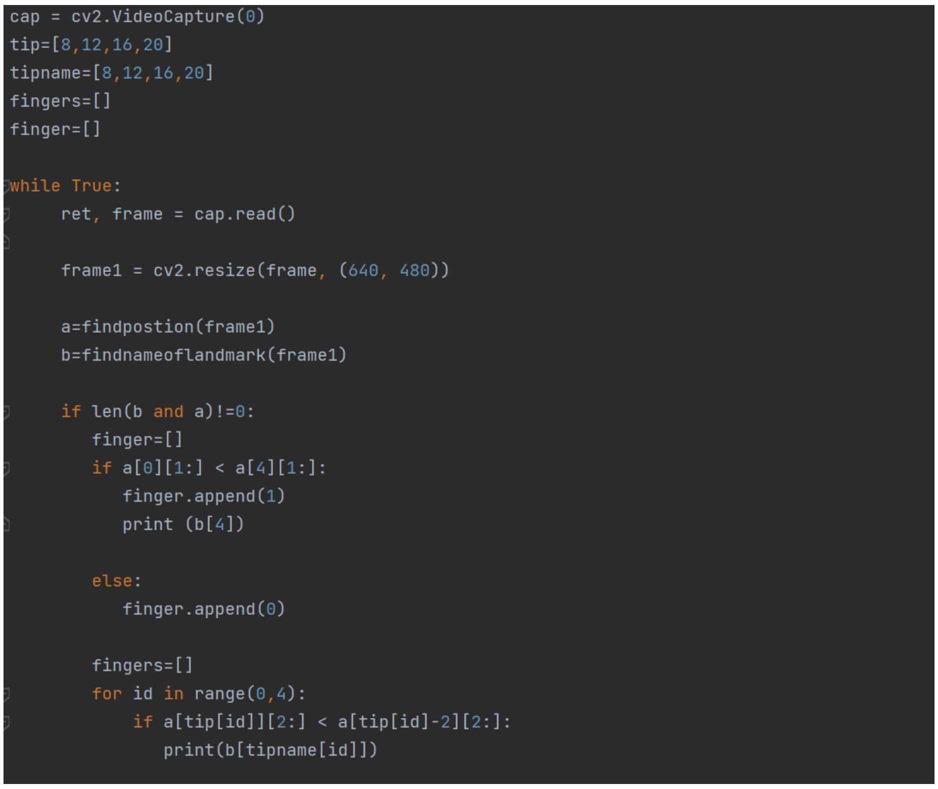
```
hands it identifies in Real-Time

me values

d tracking. As we are using a live video stream this is not a static
tion and tracking and we will only let two hands be tracked at the same time
 will slow down the system
ion_confidence=0.7, min_tracking_confidence=0.7, max_num_hands=2) as hands:

to our desktop and that will search for hands

        ret, frame = cap.read()
        #Unedit the below line if your live feed is produced upsidedown
        #flipped = cv2.flip(frame, flipCode = -1)

        #Determines the frame size, 640 x 480 offers a nice balance between speed and accurate identification
        frame1 = cv2.resize(frame, (640, 480))

        #produces the hand framework overlay ontop of the hand, you can choose the colour here too)
        results = hands.process(cv2.cvtColor(frame1, cv2.COLOR_BGR2RGB))

        #Incase the system sees multiple hands this if statment deals with that and produces another hand overlay
        if results.multi_hand_landmarks != None:
           for handLandmarks in results.multi_hand_landmarks:
               drawingModule.draw_landmarks(frame1, handLandmarks, handsModule.HAND_CONNECTIONS)
```

The Finger count and Finger up-down recogniser code

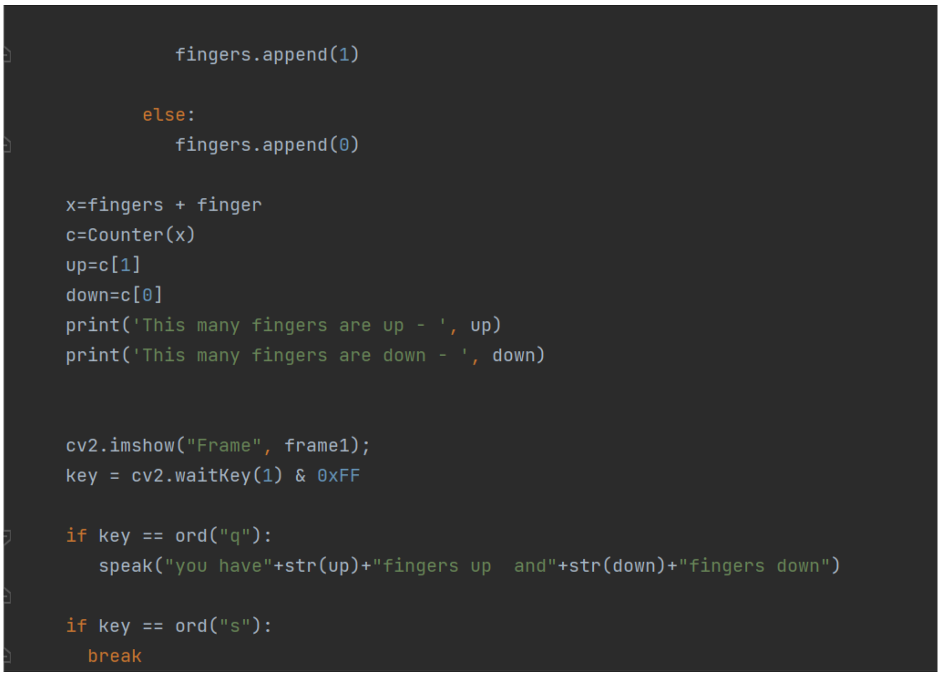
In this step we add code to the above script so that it can know when we have a finger up/down and to give a total for how many fingers are up/down.

Working:

- It is similar to above code but some more functionalities are added.
- After producing the feed and determining the location of joints f fingers a series of ‘If’ statements are added.
- These ‘If’ statements will determine if a finger is up or down and then print the details to the console.
- A function is then added that will print to the terminal the number of fingers that are up or down.
- Current frame is then updated and a key ‘s’ is associated such that when it is pressed the number fingers up/down are announced out loud.
- Also, a key with to stop the execution of file is declared in the code.

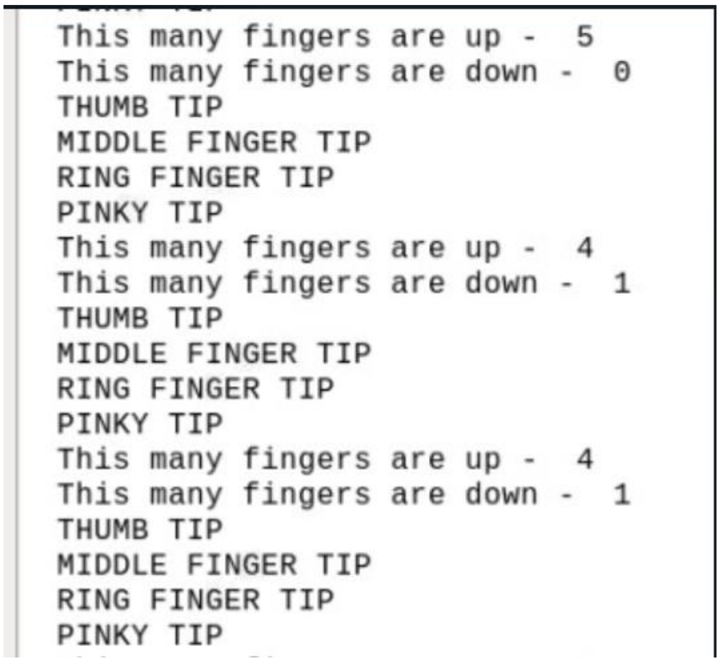


```
cap = cv2.VideoCapture(0)
tip=[8,12,16,20]
tipname=[8,12,16,20]
fingers=[]
finger=[]

while True:
     ret, frame = cap.read()

     frame1 = cv2.resize(frame, (640, 480))

     a=findpostion(frame1)
     b=findnameoflandmark(frame1)

     if len(b and a)!=0:
        finger=[]
        if a[0][1:] < a[4][1:]:
           finger.append(1)
           print (b[4])

        else:
           finger.append(0)

        fingers=[]
        for id in range(0,4):
            if a[tip[id]][2:] < a[tip[id]-2][2:]:
               print(b[tipname[id]])
```



```
                fingers.append(1)

            else:
                fingers.append(0)

     x=fingers + finger
     c=Counter(x)
     up=c[1]
     down=c[0]
     print('This many fingers are up - ', up)
     print('This many fingers are down - ', down)


     cv2.imshow("Frame", frame1);
     key = cv2.waitKey(1) & 0xFF

     if key == ord("q"):
        speak("you have"+str(up)+"fingers up  and"+str(down)+"fingers down")

     if key == ord("s"):
       break
```

Output



```
This many fingers are up -  5
This many fingers are down -  0
THUMB TIP
MIDDLE FINGER TIP
RING FINGER TIP
PINKY TIP
This many fingers are up -  4
This many fingers are down -  1
THUMB TIP
MIDDLE FINGER TIP
RING FINGER TIP
PINKY TIP
This many fingers are up -  4
This many fingers are down -  1
THUMB TIP
MIDDLE FINGER TIP
RING FINGER TIP
PINKY TIP
```

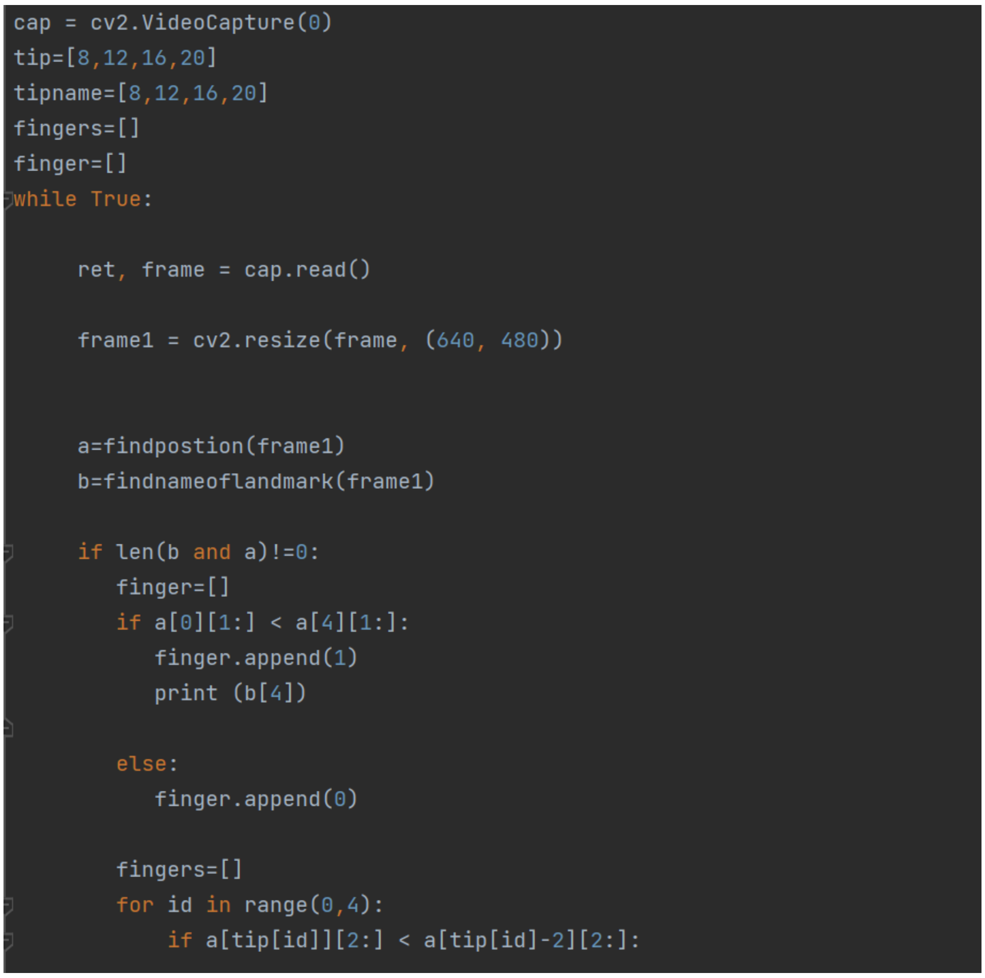
The computer gesture control code

In this step we add code to the above script so that we can have some based on the number of fingers that are up/down.

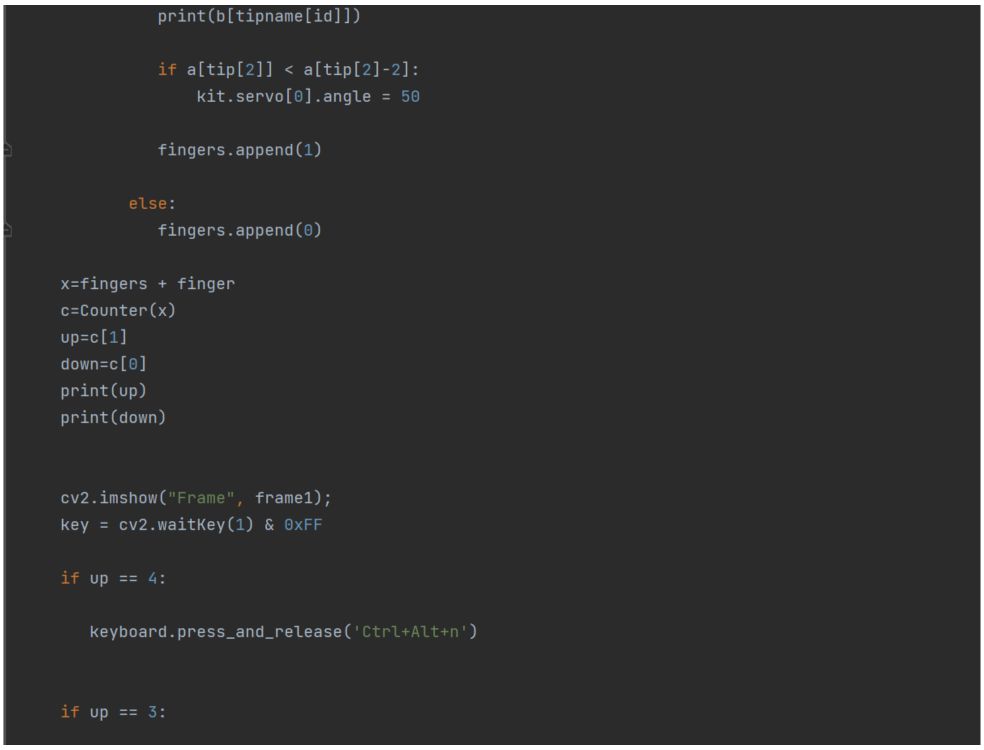
Working:

- Here the working of code is similar to the above steps but different conditions are added for different scenarios
- Conditions:
  1. 5 fingers up – No change
  2. 4 fingers up – Video Play
  3. 3 fingers up – Video Stop
  4. 2 fingers up – Full volume
  5. 1 finger up – Zero Volume

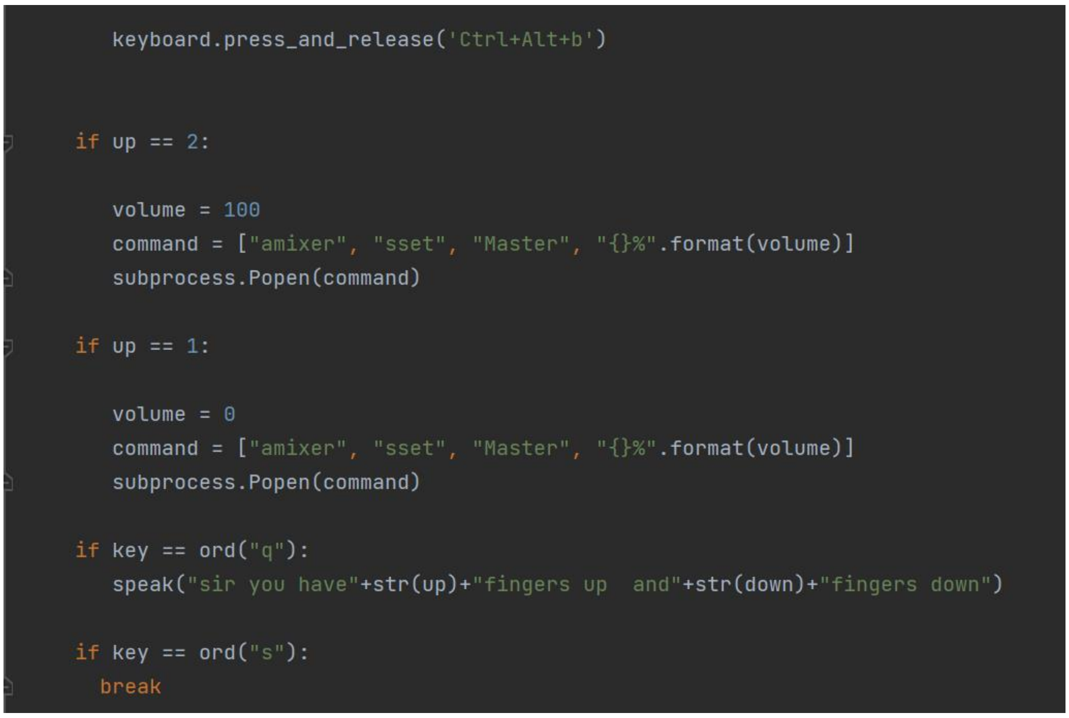
Detailed output is shown in live.



```python
cap = cv2.VideoCapture(0)
tip=[8,12,16,20]
tipname=[8,12,16,20]
fingers=[]
finger=[]
while True:

    ret, frame = cap.read()

    frame1 = cv2.resize(frame, (640, 480))


    a=findpostion(frame1)
    b=findnameoflandmark(frame1)

    if len(b and a)!=0:
        finger=[]
        if a[0][1:] < a[4][1:]:
            finger.append(1)
            print (b[4])

        else:
            finger.append(0)

        fingers=[]
        for id in range(0,4):
            if a[tip[id]][2:] < a[tip[id]-2][2:]:
```



```python
                print(b[tipname[id]])

                if a[tip[2]] < a[tip[2]-2]:
                    kit.servo[0].angle = 50

                fingers.append(1)

            else:
                fingers.append(0)

    x=fingers + finger
    c=Counter(x)
    up=c[1]
    down=c[0]
    print(up)
    print(down)


    cv2.imshow("Frame", frame1);
    key = cv2.waitKey(1) & 0xFF

    if up == 4:

       keyboard.press_and_release('Ctrl+Alt+n')


    if up == 3:
```



```
        keyboard.press_and_release('Ctrl+Alt+b')


    if up == 2:

        volume = 100
        command = ["amixer", "sset", "Master", "{}%".format(volume)]
        subprocess.Popen(command)

    if up == 1:

        volume = 0
        command = ["amixer", "sset", "Master", "{}%".format(volume)]
        subprocess.Popen(command)

    if key == ord("q"):
        speak("sir you have"+str(up)+"fingers up  and"+str(down)+"fingers down")

    if key == ord("s"):
       break
```

Socket Programming

Sockets allow communication between two different processes on the same or different machines. Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

Socket Types

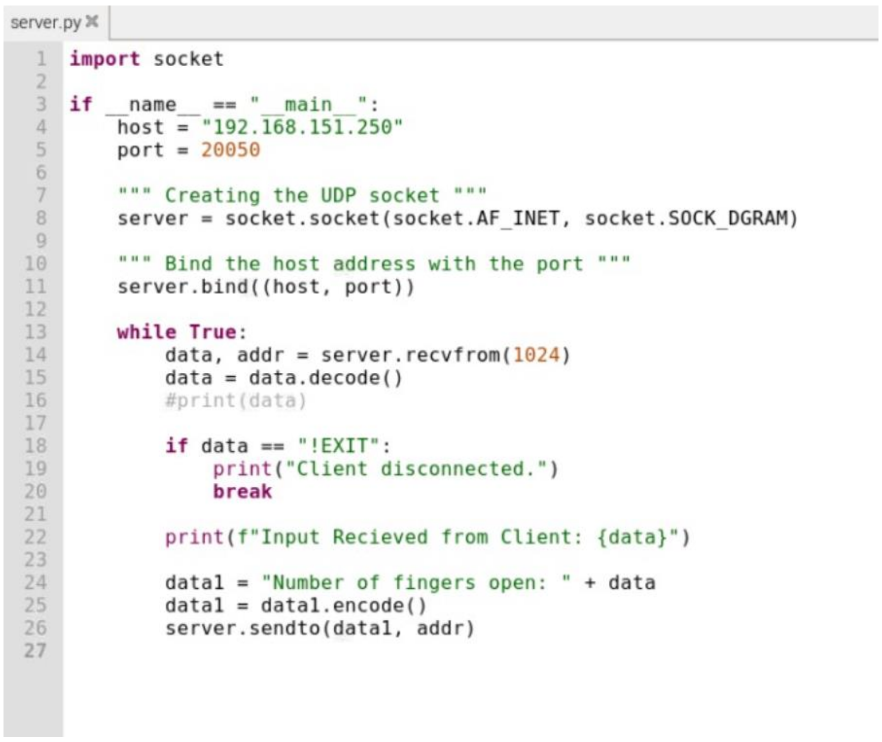
- **Stream Sockets** − Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.
- **Datagram Sockets** − Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).
- **Raw Sockets** − These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.
- **Sequenced Packet Sockets** − They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the

Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets, either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to be used with all outgoing data, and allows the user to receive the headers on incoming packets.

The Various System calls that are used in order to communicate are:

- Socket()→Initialise the nodes so that the data generated from the application layer can be sent to the other endpoint/node.
- Bind()→A socket address will be associated which is a combination of IP address and the port number.
- Listen()→Sets the server model to passive mode that is ready for a client
- Accept()→Now the requests are accepted by the server.
- Send()→writable function
- Recv()→readable function
- Close()→terminating the communication

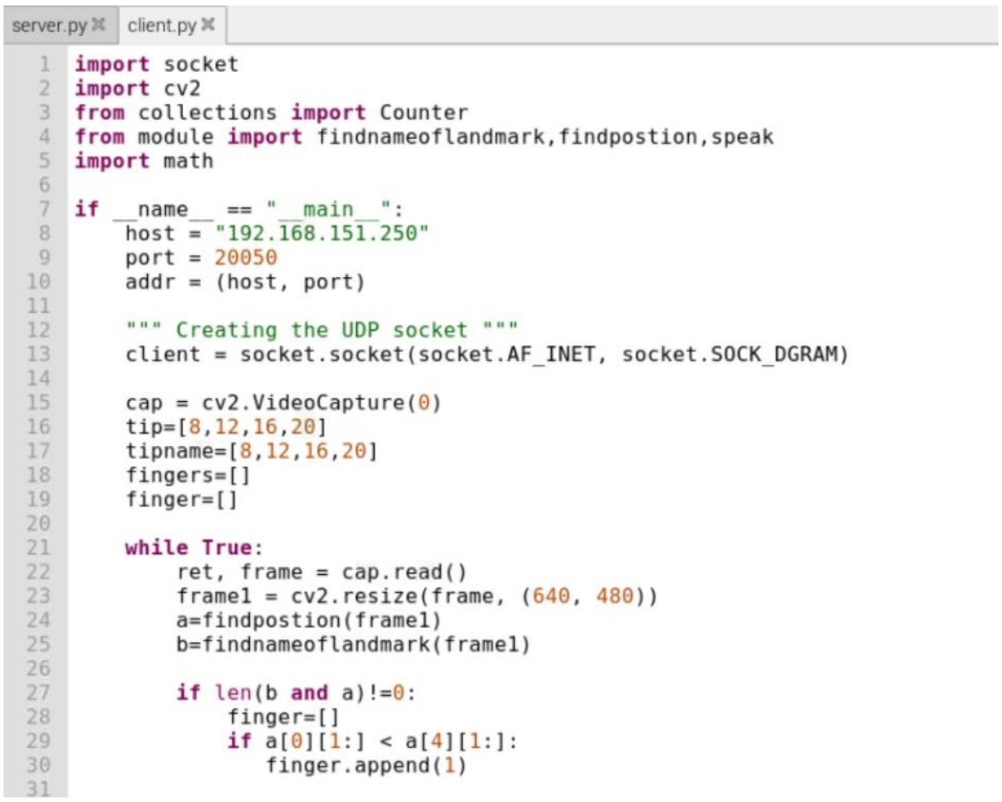
Implementation of Server-Client Model



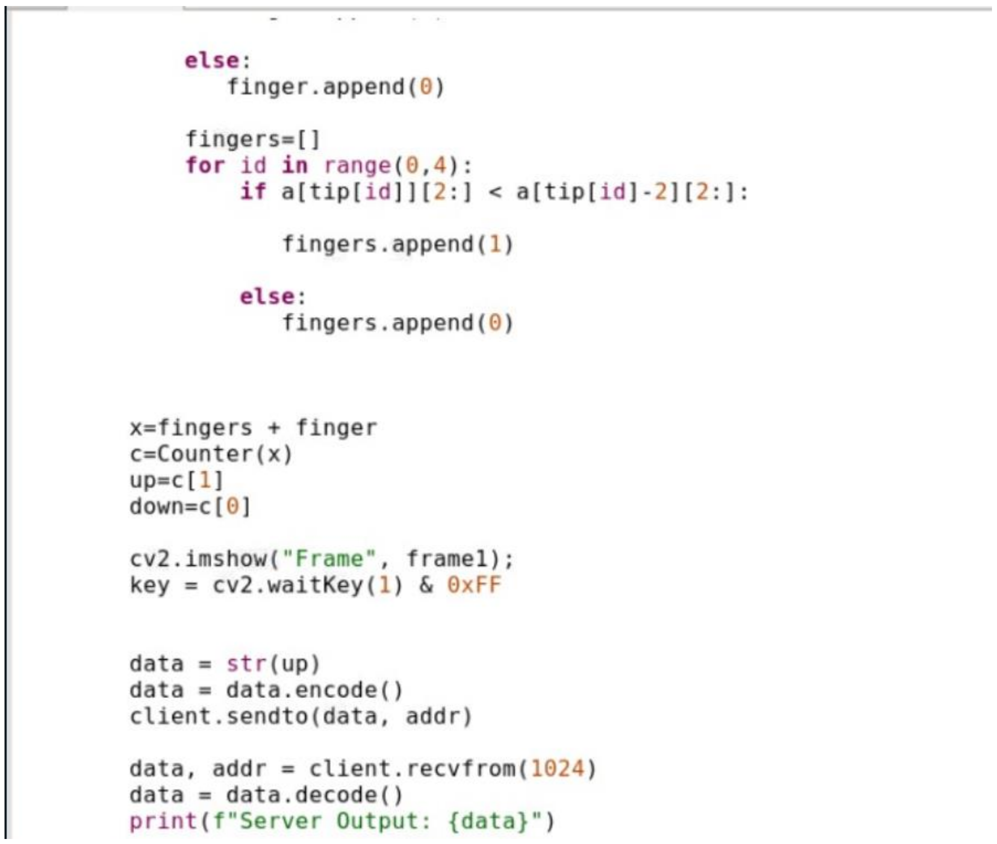
```python
import socket

if __name__ == "__main__":
    host = "192.168.151.250"
    port = 20050

    """ Creating the UDP socket """
    server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    """ Bind the host address with the port """
    server.bind((host, port))

    while True:
        data, addr = server.recvfrom(1024)
        data = data.decode()
        #print(data)

        if data == "!EXIT":
            print("Client disconnected.")
            break

        print(f"Input Recieved from Client: {data}")

        data1 = "Number of fingers open: " + data
        data1 = data1.encode()
        server.sendto(data1, addr)
```

Working

- In this code we will be ensuring that we will be creating a UDP socket which is a combination of IP address and port number.
- Here we are establishing a client-server model using UDP socket to enable continuous data flow between client and server.
- We set the code such that the encoded message is received from the client and then decode it which is then read and published in the output.
- We also set an exiting parameter key.



```python
import socket
import cv2
from collections import Counter
from module import findnameoflandmark,findpostion,speak
import math

if __name__ == "__main__":
    host = "192.168.151.250"
    port = 20050
    addr = (host, port)

    """ Creating the UDP socket """
    client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    cap = cv2.VideoCapture(0)
    tip=[8,12,16,20]
    tipname=[8,12,16,20]
    fingers=[]
    finger=[]

    while True:
        ret, frame = cap.read()
        frame1 = cv2.resize(frame, (640, 480))
        a=findpostion(frame1)
        b=findnameoflandmark(frame1)

        if len(b and a)!=0:
            finger=[]
            if a[0][1:] < a[4][1:]:
               finger.append(1)

```

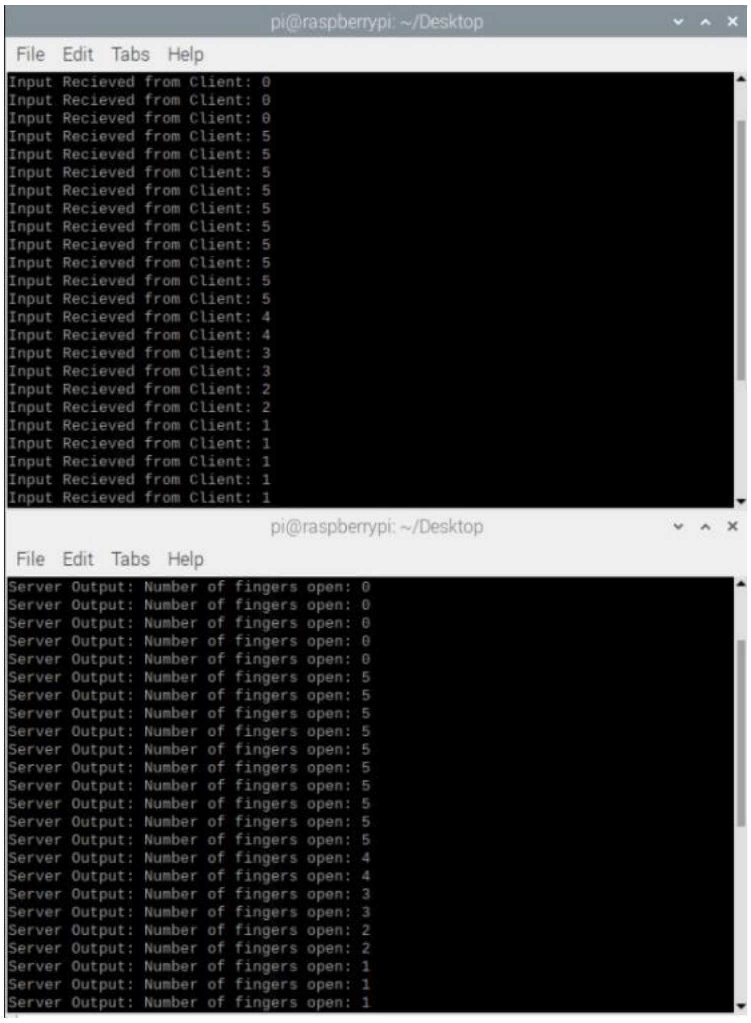


```
            else:
                finger.append(0)

            fingers=[]
            for id in range(0,4):
                if a[tip[id]][2:] < a[tip[id]-2][2:]:

                    fingers.append(1)

                else:
                    fingers.append(0)


        x=fingers + finger
        c=Counter(x)
        up=c[1]
        down=c[0]

        cv2.imshow("Frame", frame1);
        key = cv2.waitKey(1) & 0xFF


        data = str(up)
        data = data.encode()
        client.sendto(data, addr)

        data, addr = client.recvfrom(1024)
        data = data.decode()
        print(f"Server Output: {data}")
```

Working

- The main component of code is a finger-up/down recogniser.
- A UDP socket is created.
- The data is then encoded and sent to the server
- The data is then duplicated by the server and the received data is then decoded and gets printed in the output.

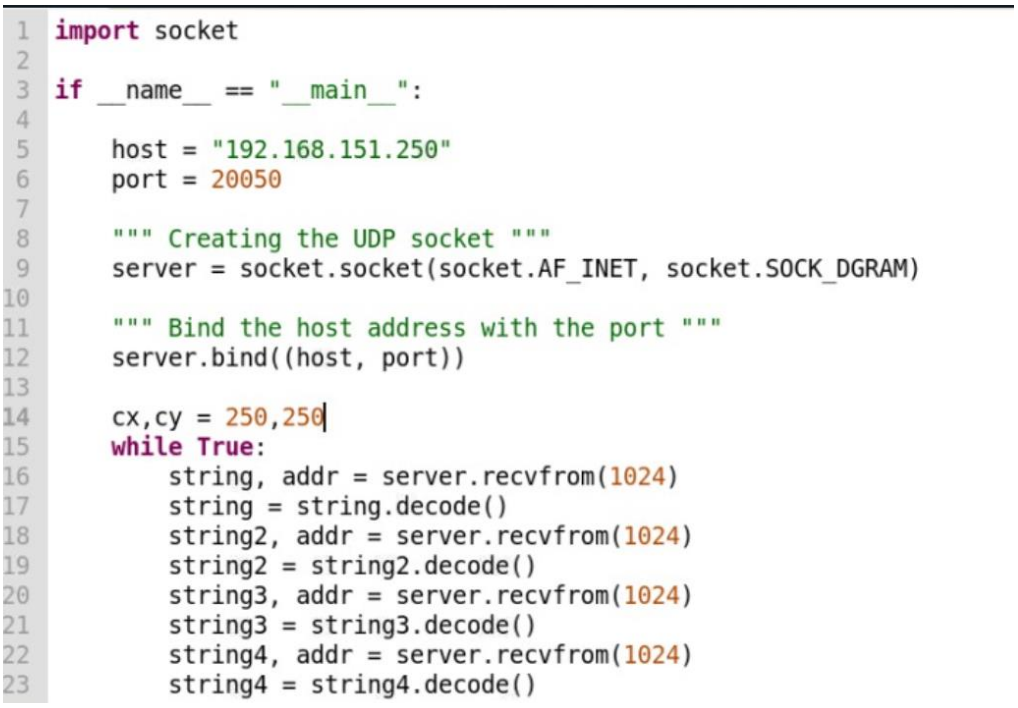
Output



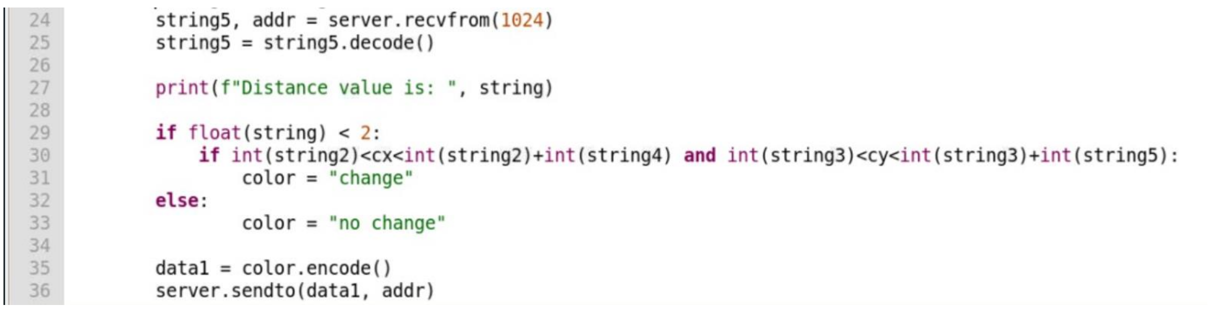
```
Input Recieved from Client: 0
Input Recieved from Client: 0
Input Recieved from Client: 0
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 5
Input Recieved from Client: 4
Input Recieved from Client: 4
Input Recieved from Client: 3
Input Recieved from Client: 3
Input Recieved from Client: 2
Input Recieved from Client: 2
Input Recieved from Client: 1
Input Recieved from Client: 1
Input Recieved from Client: 1
Input Recieved from Client: 1
Input Recieved from Client: 1
```

```
Server Output: Number of fingers open: 0
Server Output: Number of fingers open: 0
Server Output: Number of fingers open: 0
Server Output: Number of fingers open: 0
Server Output: Number of fingers open: 0
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 5
Server Output: Number of fingers open: 4
Server Output: Number of fingers open: 4
Server Output: Number of fingers open: 3
Server Output: Number of fingers open: 3
Server Output: Number of fingers open: 2
Server Output: Number of fingers open: 2
Server Output: Number of fingers open: 1
Server Output: Number of fingers open: 1
Server Output: Number of fingers open: 1
```

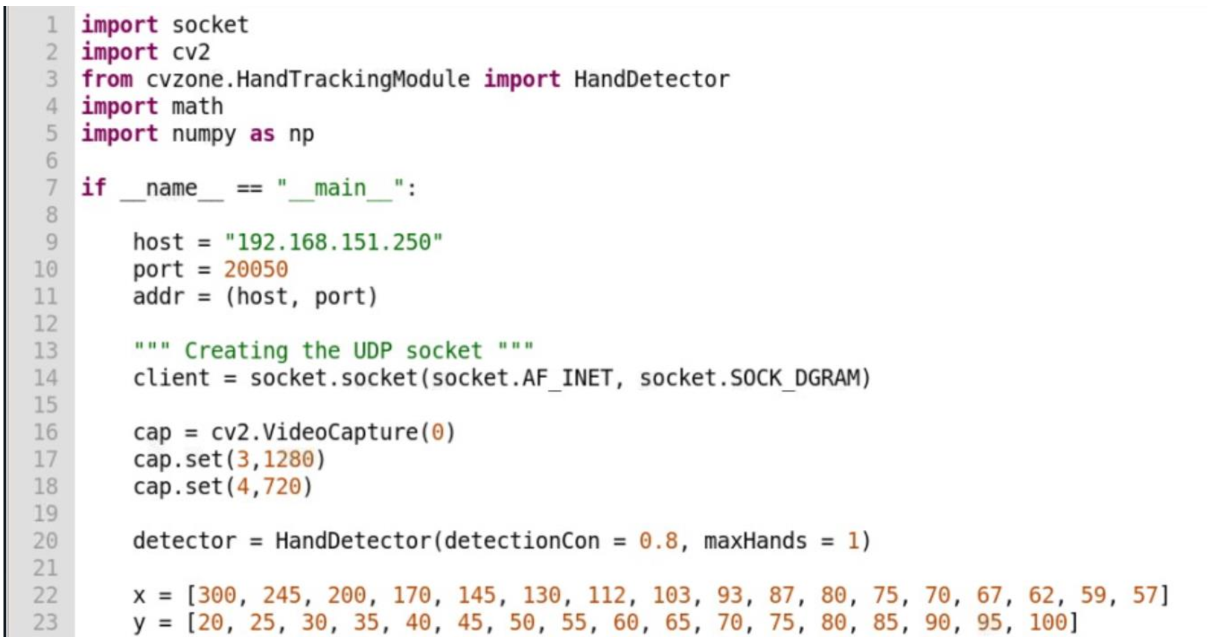
UDP model for distance prediction



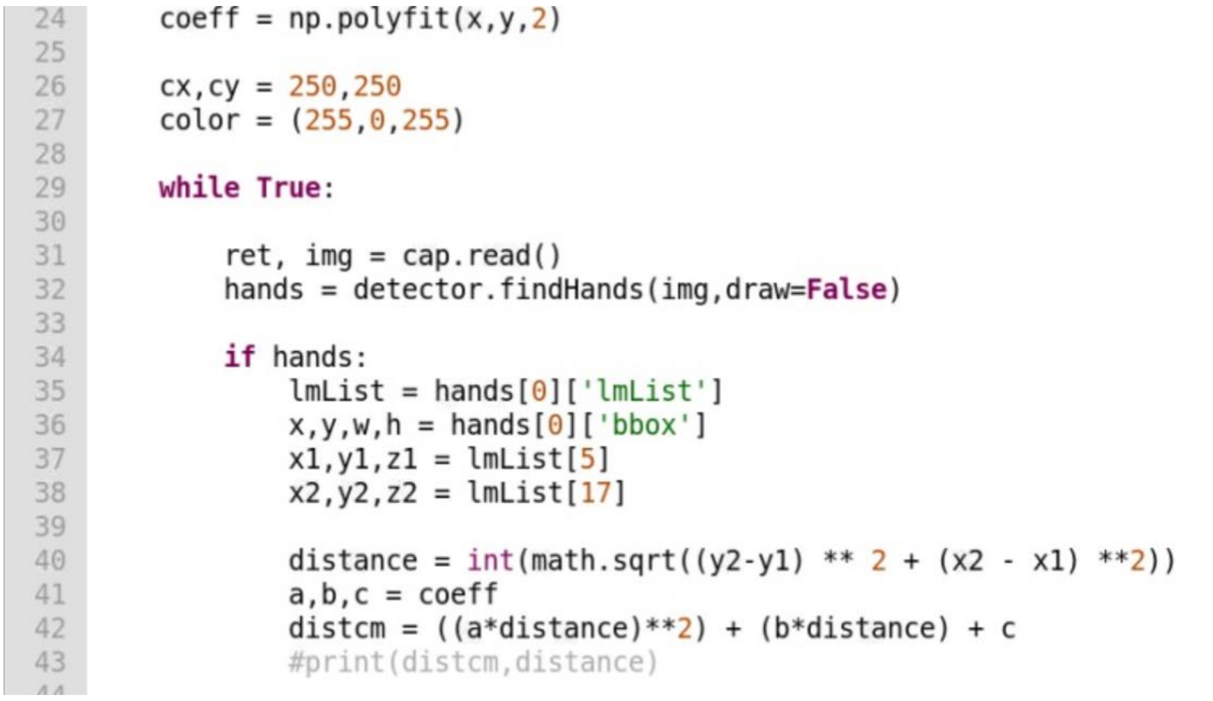
```python
import socket

if __name__ == "__main__":

    host = "192.168.151.250"
    port = 20050

    """ Creating the UDP socket """
    server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    """ Bind the host address with the port """
    server.bind((host, port))

    cx,cy = 250,250
    while True:
        string, addr = server.recvfrom(1024)
        string = string.decode()
        string2, addr = server.recvfrom(1024)
        string2 = string2.decode()
        string3, addr = server.recvfrom(1024)
        string3 = string3.decode()
        string4, addr = server.recvfrom(1024)
        string4 = string4.decode()
```



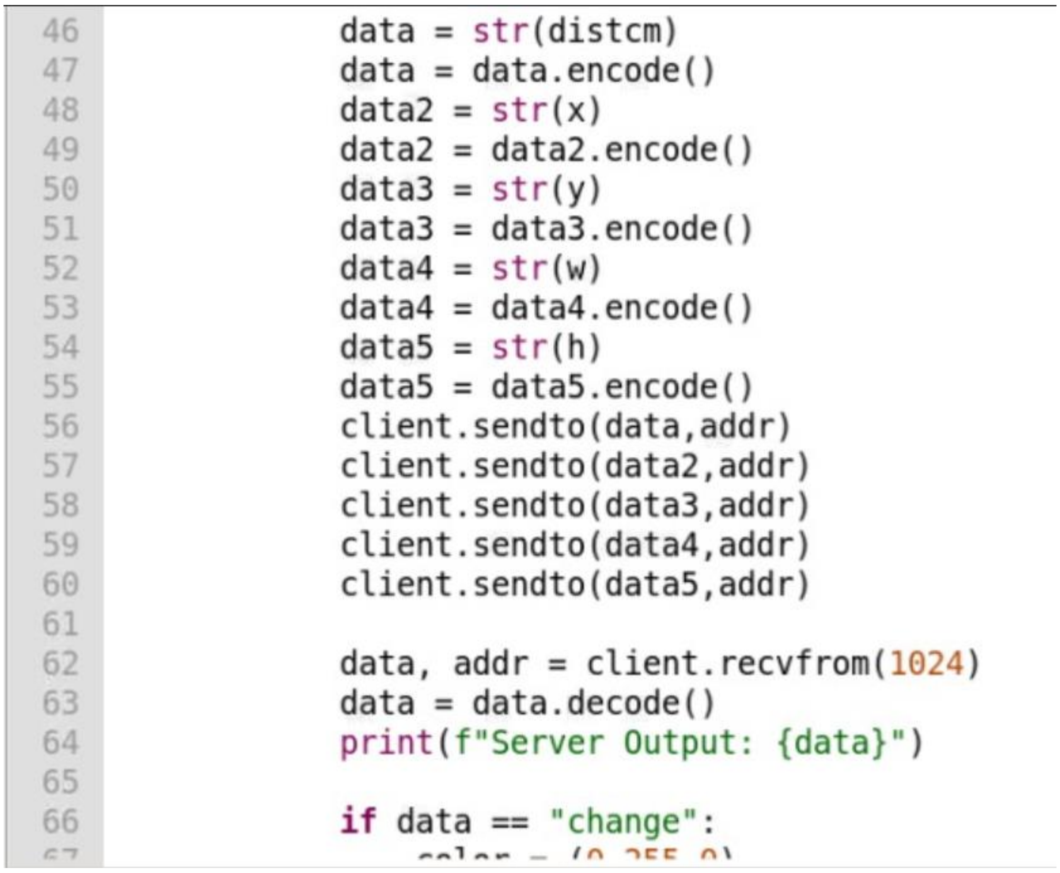
```python
        string5, addr = server.recvfrom(1024)
        string5 = string5.decode()

        print(f"Distance value is: ", string)

        if float(string) < 2:
            if int(string2)<cx<int(string2)+int(string4) and int(string3)<cy<int(string3)+int(string5):
                color = "change"
        else:
                color = "no change"

        data1 = color.encode()
        server.sendto(data1, addr)
```



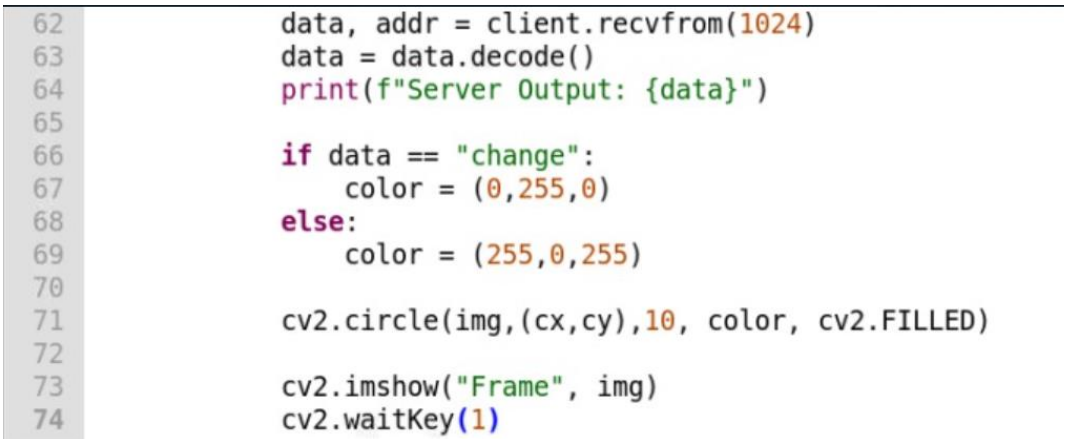
```python
import socket
import cv2
from cvzone.HandTrackingModule import HandDetector
import math
import numpy as np

if __name__ == "__main__":

    host = "192.168.151.250"
    port = 20050
    addr = (host, port)

    """ Creating the UDP socket """
    client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

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

    detector = HandDetector(detectionCon = 0.8, maxHands = 1)

    x = [300, 245, 200, 170, 145, 130, 112, 103, 93, 87, 80, 75, 70, 67, 62, 59, 57]
    y = [20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
```



```python
    coeff = np.polyfit(x,y,2)

    cx,cy = 250,250
    color = (255,0,255)

    while True:

        ret, img = cap.read()
        hands = detector.findHands(img,draw=False)

        if hands:
            lmList = hands[0]['lmList']
            x,y,w,h = hands[0]['bbox']
            x1,y1,z1 = lmList[5]
            x2,y2,z2 = lmList[17]

            distance = int(math.sqrt((y2-y1) ** 2 + (x2 - x1) **2))
            a,b,c = coeff
            distcm = ((a*distance)**2) + (b*distance) + c
            #print(distcm,distance)
```



```python
data = str(distcm)
data = data.encode()
data2 = str(x)
data2 = data2.encode()
data3 = str(y)
data3 = data3.encode()
data4 = str(w)
data4 = data4.encode()
data5 = str(h)
data5 = data5.encode()
client.sendto(data,addr)
client.sendto(data2,addr)
client.sendto(data3,addr)
client.sendto(data4,addr)
client.sendto(data5,addr)

data, addr = client.recvfrom(1024)
data = data.decode()
print(f"Server Output: {data}")

if data == "change":
```



```python
data, addr = client.recvfrom(1024)
data = data.decode()
print(f"Server Output: {data}")

if data == "change":
    color = (0,255,0)
else:
    color = (255,0,255)

cv2.circle(img,(cx,cy),10, color, cv2.FILLED)

cv2.imshow("Frame", img)
cv2.waitKey(1)
```

Detailed output is shown in live.

Working

- The main objective is to find whether the hand which has been detected using the web camera is reaching a certain point on the screen to achieve this we are establishing a client-server model which uses UDP sockets for communication.
- The client captures the data using the webcam and sends certain distance value which has been calculated to the server, the server analyses this value and sends a message to the client(Change/no-change).
- The final output is given by the client based on the message received from the server(changes the colour of dot for certain distance value).

Conclusion

This project can be further developed where the coordinates generated or the data generated can be used to move the actuators and perform various tasks.