

**INSTITUTE OF ENGINEERING AND MANAGEMENT (IEM)**

**KOLKATA**

**PROJECT REPORT**

**ON**

**“Hand Gesture Recognition”**

**SUBMITTED IN FULLFILMENT OF BTECH 1ST YEAR 2022**

BY

Uday Sankar Mukherjee (Section: F2 Roll no.: 17)

Nilanjan Bhattacharya (Section: F2 Roll no.: 18)

Atrij Paul (Section: F2 Roll no.: 25)

Ayan Ghosh (Section: F2 Roll no.: 26)

DATE OF SUBMISSION: 03-05-2022

**ABSTRACT**

Hand gesture recognition is used enormously in the recent years for interact human and machine. There are many types of gestures such as arm, hand, face and many other but hand gestures give more meaningful information than other types of gestures. There are many techniques for hand gesture recognition, such as color marker approach, vision-based approach, glove-based approach and depth-based approach. The main purpose of gesture recognition system is to develop a useful system which can recognize human hand gestures and used them to control electronic devices. This paper reviewed the most common used hand gesture recognition methods, tools and analysis the strength and weakness of these methods, and lists the current challenging problems of hand gesture recognition system.

INDEX

|  |  |  |
| --- | --- | --- |
| SL. NO. | CONTENT | PG. NO. |
| 1 | INTRODUCTION | 4 |
| 2 | DESIGN/  IMPLEMENTATION | 5 |
| 3 | RESULT AND ANALYSIS | 11 |
| 4 | CONCLUSION | 13 |
| 5 | REFERENCES | 14 |

INTRODUCTION

Gesture recognition is an active research field in Human-Computer Interaction technology. It has many applications in virtual environment control and sign language translation, robot control, or music creation. In this machine learning project on Hand Gesture Recognition, we are going to make a real-time Hand Gesture Recognizer using the MediaPipe framework and Tensorflow in OpenCV and Python.

OpenCV is a real-time Computer vision and image-processing framework built on C/C++. But we’ll use it on python via the OpenCV-python package.

The main challenges that need to solve as research issue include:

• Illumination condition is the most sensitivity for hand gesture recognition of vision based system.

• Complex backgrounds, dynamic backgrounds are also main difficult for hand gesture recognition.

• The different size of the user’s hand is another issue.

• The multiple gestures in the same background and different viewpoints.

• The several persons contains in the sense other than the real subjects.

Illumination condition is the most sensitivity for hand gesture recognition of vision based system.

• Complex backgrounds, dynamic backgrounds are also main difficult for hand gesture recognition.

• The different size of the user’s hand is another issue.

• The multiple gestures in the same background and different viewpoints.

• The several persons contains in the sense other than the real subjects.

Illumination condition is the most sensitivity for hand gesture recognition of vision based system.

• Complex backgrounds, dynamic backgrounds are also main difficult for hand gesture recognition.

• The different size of the user’s hand is another issue.

• The multiple gestures in the same background and different viewpoints.

• The several persons contains in the sense other than the real subjects.

* Illumination condition is the most sensitivity for hand gesture recognition of vision-based system.
* Complex backgrounds, dynamic backgrounds are also main difficult for hand gesture recognition.
* The different size of the user’s hand is another issue.
* The multiple gestures in the same background and different viewpoints.
* The several persons contain in the sense other than the real subjects.

DESIGN/IMPLEMENTATION

What is MediaPipe?

MediaPipe is a customizable machine learning solutions framework developed by Google. It is an open-source and cross-platform framework, and it is very lightweight. MediaPipe comes with some pre-trained ML solutions such as face detection, pose estimation, hand recognition, object detection, etc.

Prerequisites for this project:

1. Python – 3.x (we used Python 3.8.8 in this project)  
2. OpenCV – 4.5

* Run “pip install opencv-python” to install OpenCV.

3. MediaPipe – 0.8.5

* Run “pip install mediapipe” to install MediaPipe.

4. Numpy – 1.19.3

Steps to Perform:

#### **Step 1 – Import necessary packages:**

#### **Step 2 – Initialize models:**

#### **Step 3 – Read frames from a webcam:**

#### **Step 4 – Detect hand keypoints**

#### **Step 5 – Recognize hand gestures**

[**Hand Gesture Recognition ML Project Code**](https://techvidvan.s3.amazonaws.com/machine-learning-projects/hand-gesture-recognition-code.zip)

#hand gesture recognition

import cv2

import mediapipe as mp

mp\_drawing = mp.solutions.drawing\_utils

mp\_hands = mp.solutions.hands

cap = cv2.VideoCapture(0)

with mp\_hands.Hands(

    min\_detection\_confidence=0.5,

    min\_tracking\_confidence=0.5) as hands:

  while cap.isOpened():

    success, image = cap.read()

    if not success:

      print("Ignoring empty camera frame.")

      continue

    image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR\_BGR2RGB)

    image.flags.writeable = False

    results = hands.process(image)

    image.flags.writeable = True

    image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

    if results.multi\_hand\_landmarks:

      for hand\_landmarks in results.multi\_hand\_landmarks:

        mp\_drawing.draw\_landmarks(

            image, hand\_landmarks, mp\_hands.HAND\_CONNECTIONS)

    cv2.imshow('MediaPipe Hands', image)

    if cv2.waitKey(5) & 0xFF == 27:

      break

cap.release()

*#volume control*

import cv2

import mediapipe

import numpy as np

import HandTrackingModule as ht

import math

from ctypes import cast, POINTER

from comtypes import CLSCTX\_ALL

from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

cap = cv2.VideoCapture(0 , cv2.CAP\_DSHOW)

cap.set(3 , 640)

cap.set(4 , 480)

detector = ht.HandDetector()

devices = AudioUtilities.GetSpeakers()

interface = devices.Activate(

    IAudioEndpointVolume.\_iid\_, CLSCTX\_ALL, None)

volume = cast(interface, POINTER(IAudioEndpointVolume))

volRange = volume.GetVolumeRange()

'''We see that the minimum range is -65 and the maximum is 0'''

'''volume.GetVolumeRange() will give the value in the form of a tuple with the minimum value , maximum value and another parameter'''

minVol = volRange[0]

maxVol = volRange[1]

while(True):

    \_ , img = cap.read()

    img = cv2.flip(img , 1)

    img = detector.findHands(img)

    idList = detector.findPosition(img , False)

    if(len(idList)!=0):

        x1 , y1 = idList[4][1] , idList[4][2]

        x2 , y2 = idList[8][1] , idList[8][2]

        x3 , y3 = (x1 + x2)//2 , (y1 + y2)//2

        cv2.circle(img , (x1 , y1) , 15 , (255  , 0 , 255) , cv2.FILLED)

        cv2.circle(img , (x2 , y2) , 15 , (255  , 0 , 255) , cv2.FILLED)

        cv2.line(img , (x1 , y1) , (x2 , y2) , (255 , 0 , 255) , 3)

        cv2.circle(img , (x3 , y3) , 15 , (255 , 0 , 255) , cv2.FILLED)

        length = math.hypot(x2 - x1 , y2 - y1)

        print(length)

       '''Here we see the maximum length coming to be almost 400 and the minimum length to be 15'''

        if length<=50:

            cv2.circle(img , (x3 , y3) , 15 , (0 , 255 , 0) , cv2.FILLED)

        '''Converting length to volume range'''

        vol = np.interp(length , [50 , 400] , [minVol , maxVol])

        print(int(length) , vol)

        volume.SetMasterVolumeLevel(vol , None)

        volBar = np.interp(length , [50 , 400] , [400 , 150])

        cv2.rectangle(img , (50 , 150) , (85 , 400) , (0 , 255 , 0) , 3)

        cv2.rectangle(img , (50 , int(volBar)) , (85 , 400) , (0 , 255 , 0) , cv2.FILLED)

        volPercent = np.interp(length , [50 , 400] , [0 , 100])

        cv2.putText(img , f'{int(volPercent)} %' , (40 , 100) , cv2.FONT\_HERSHEY\_SCRIPT\_COMPLEX , 1 , (255 , 0 , 0) , 3)

    cv2.imshow("Result" , img)

    cv2.waitKey(1)

#finger counting

import cv2

import numpy as np

import HandTrackingModule as ht

cap = cv2.VideoCapture(0 , cv2.CAP\_DSHOW)

cap.set(3 , 700)

cap.set(4 , 600)

detector  = ht.HandDetector()

def getNumber(fingers):

    s = ""

    for i in fingers:

        s += str(i)

    if(s == "00000"):

        return "A"

    elif(s == "10001"):

        return "Y"

    elif(s == "00001"):

        return "I"

    elif(s == "11000"):

        return "L"

    elif(s == "01000"):

        return 1

    elif(s == "01100"):

        return 2

    elif(s == "11100"):

        return 3

    elif(s == "01111"):

        return 4

    elif(s == "11111"):

        return 5

    elif(s == "01110"):

        return 6

    elif(s == "01101"):

        return 7

    elif(s == "01011"):

        return 8

    elif(s == "00111"):

        return 9

while(True):

    \_ , img = cap.read()

    img = cv2.flip(img , 1)

    img = detector.findHands(img)

    idList = detector.findPosition(img , False)

    tipId = [4 , 8 , 12 , 16 , 20]

    if(len(idList)!=0):

        fingers = []

        if(idList[tipId[0]][1] < idList[tipId[0]-2][1]):

            fingers.append(1)

        else:

            fingers.append(0)

        for id in range(1 , len(tipId)):

            if(idList[tipId[id]][2] < idList[tipId[id]-2][2]):

                fingers.append(1)

            else:

                fingers.append(0)

        cv2.putText(img , str(getNumber(fingers)) , (45 , 375) , cv2.FONT\_HERSHEY\_PLAIN , 10 , (255 , 0 , 255) , 20)

    cv2.imshow("Result" , img)

    cv2.waitKey(1)

*#sign language*

import cv2

import numpy as np

import HandTrackingModule as ht

cap = cv2.VideoCapture(0 , cv2.CAP\_DSHOW)

cap.set(3 , 800)

cap.set(4 , 700)

detector  = ht.HandDetector()

def getNumber(fingers):

    s = ""

    for i in fingers:

        s += str(i)

    if(s == "00011"):

        return "OKAY"

    elif(s == "01000"):

        return "OUT"

    elif(s == "01100"):

        return "VICTORY"

    elif(s== "10000"):

        return "FINE"

    elif(s== "11111"):

        return "STOP"

    elif(s== "01001"):

        return "ROCK"

while(True):

    \_ , img = cap.read()

    img = cv2.flip(img , 1)

    img = detector.findHands(img)

    idList = detector.findPosition(img , False)

    tipId = [4 , 8 , 12 , 16 , 20]

    if(len(idList)!=0):

        fingers = []

        if(idList[tipId[0]][1] < idList[tipId[0]-2][1]):

            fingers.append(1)

        else:

            fingers.append(0)

        for id in range(1 , len(tipId)):

            if(idList[tipId[id]][2] < idList[tipId[id]-2][2]):

                fingers.append(1)

            else:

                fingers.append(0)

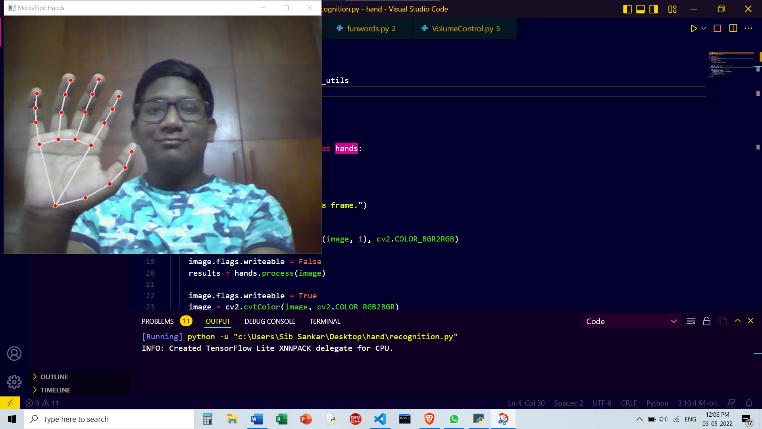
        cv2.putText(img , str(getNumber(fingers)) , (45 , 375) , cv2.FONT\_HERSHEY\_PLAIN , 10 , (255 , 0 , 255) , 20)

    cv2.imshow("Result" , img)

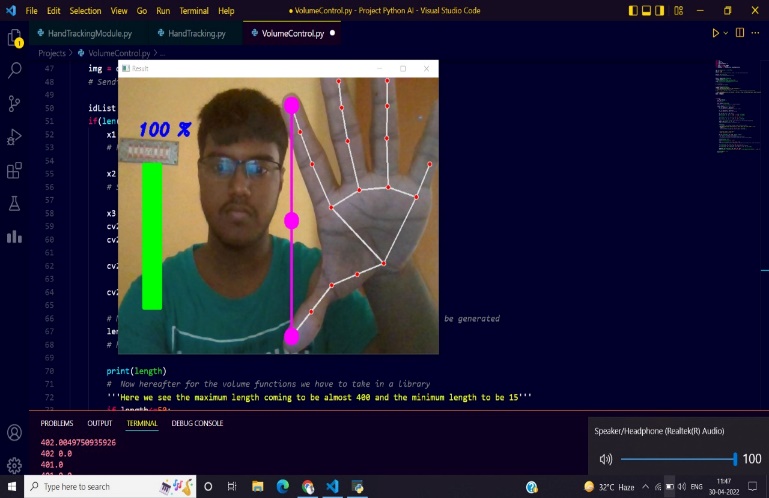
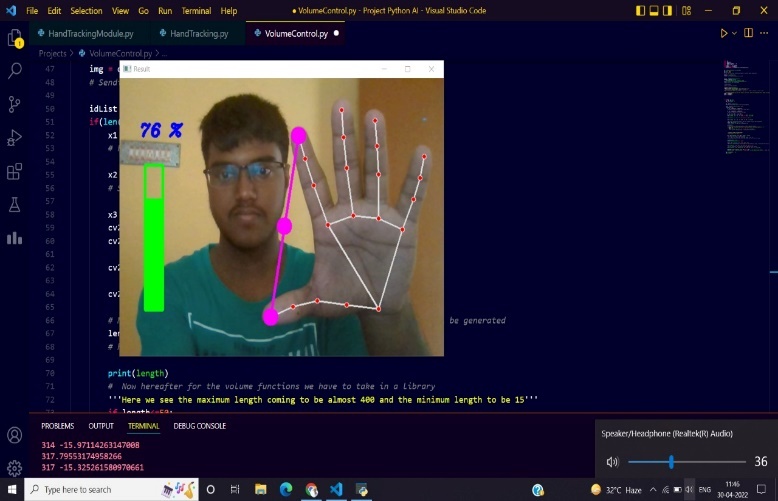
    cv2.waitKey(1)

RESULT AND ANALYSIS

* Hand Recognition



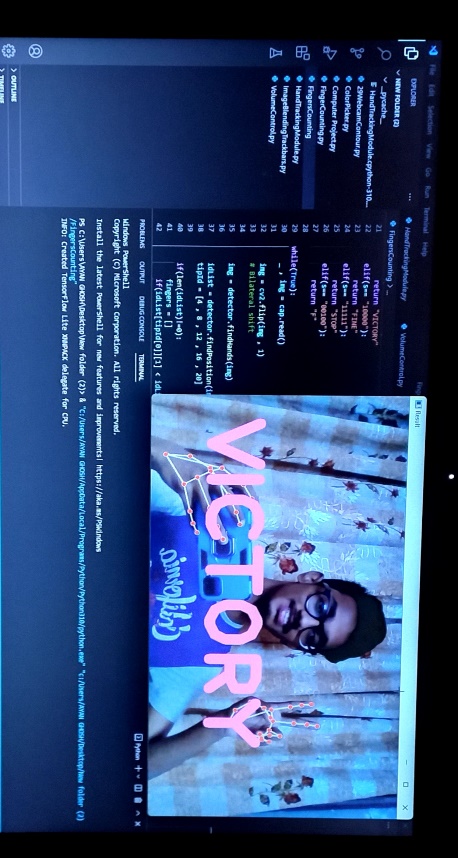
* Volume Control



* Finger Counting



* Sign Language



The explanation of the results will be focused on the simulation part since the different for the hardware

implementation is the source code of the real-time input video. These results are analysed throughout

the project scope. At first, the result of Haar-cascade classifier will be explain in detail to show how the

hand is detected before image processing. Then, the simulation part begins where the area of interest

that has been calculated. The Python file is actually the command file to run the specified program onto

the Python command script. Table 1 shows the details of the generated Haar-cascade classifier.

The explanation of the results will be focused on the simulation part since the different for the hardware

implementation is the source code of the real-time input video. These results are analysed throughout

the project scope. At first, the result of Haar-cascade classifier will be explain in detail to show how the

hand is detected before image processing. Then, the simulation part begins where the area of interest

that has been calculated. The Python file is actually the command file to run the specified program onto

the Python command script. Table 1 shows the details of the generated Haar-cascade classifier.

The proposed system is a real time video processing that is based on a real time application system

In the analysis phase we gathered information regarding various gesture recognition systems existent today and the techniques and algorithms they employ and the success/failure rate of these systems.

CONCLUSIONS

In this paper we have given a comprehensive survey of various hand gesture recognition systems. The principal component of the hand gesture recognition system are includes segmentation and tracking hand from the background and then feature extracted from the segmented hand image using various algorithms and finally, recognized the types of hand gestures by using recognition methods.

In this project we have planned, designed and implemented the system for Hand gesture recognition system for Sign Language recognition, virtual volume controlling and Finger counting.

* This project can be more interactive with the help of tracking real time hand movements and controlling mouse pointer on screen. The shortcoming of requiring a plain background can be overcome with the help of Background Image substraction or Machine Learning Techniques.
* To create a website which operates using hand gestures. JavaScript can be dynamically combined with the gesture recognition logic for the same.
* To use the gesture recognition logic in sensitive areas of work like hospitals and nuclear power plants where sterility between machines and human is vital.
* To create a battery free technology that enables the operation of mobile devices with hand gestures.

REFERENCES

* <https://google.github.io/mediapipe/solutions/hands.html>
* <https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/>
* <https://www.researchgate.net/publication/332109831_Hand_Gesture_Detection_and_Recognition_System_A_Critical_Review>
* <http://203.201.63.46:8080/jspui/bitstream/123456789/6228/1/PR3225%20-%20Recognition%20of%20hand%20Gesture%20of%20Humans%20using%20Machine%20Learning.pdf>