HUMAN COMPUTER INTERACTION USING GESTURES

A PROJECT REPORT

Submitted By

Aditya Kumar Pandey	**(19011445002)**
Siyaram Sharan Prasad	**(19011445025)**
Apeksha Kumari	**(19011445004)**
Kajal Kumari	**(19011445010)**
Ashutosh Munda	**(19011445005)**
Babita Kumari	**(19011445006)**

Under the Guidance of

Prof. Bias Bhadra

(Head of Department, Department of Computer Science and Engineering)

in partial fulfilment for the

award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

CHAIBASA ENGINEERING COLLEGE

(Estd. By Government of Jharkhand & run by Techno India Group under PPP)

Bistumpur, Jhinkpani, West Singhbhum, Jharkhand, 833215

2023

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

………………………………………….

Signature of the Students

CHAIBASA ENGINEERING COLLEGE

CERTIFICATE

Certified that this project report titled "**HUMAN COMPUTER INTERACTION USING GESTURES"** is the Bonafide work of ADITYA KUMAR PANDEY[19011445002], SIYARAM SHARAN PRASAD[19011445025], APEKSHA KUMARI [19011445004], KAJAL KUMARI[19011445010], ASHUTOSH MUNDA[19011445005], BABITA KUMARI[19011445006], who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

………………………..	………………………..
SIGNATURE	SIGNATURE
Prof. Bias Bhadra	**Mrs. Bias Bhadra**
ASSISTANT PROFESSOR	**HEAD OF DEPARTMENT**
DEPT. OF COMPUTER SCIENCE	**DEPT. OF COMPUTER SCIENCE**
& ENGINEERING	**& ENGINEERING**

ABSTRACT

Device control using hand gestures refers to the ability of a device to interpret and respond to movements of a person's hand as a means of input. This technology typically uses computer vision and machine learning algorithms to recognize specific gestures and translate them into control commands for the device. Hand gesture control can be applied to various types of devices such as smartphones, laptops, gaming consoles, and smart home appliances, making interaction more intuitive and convenient. This technology has the potential to revolutionize the way people interact with technology, making it more accessible and natural.

Our project will work in three different phases:

PHASE-1

- Research on the topic
- Deep dive to its requirements and implementations
- Planning of its development
- Research on some resources of knowledge

PHASE-2

In the phase -2 we did:

- Python program for controlling brightness of the system using hand gesture.
- Python program for controlling Volume of the system using hand gesture.

PHASE-3

- In the last phase of our project, we have implemented the phase-2 python program into its equivalent .exe

- Furthermore, we have added more features like virtual mouse, virtual keyboard, and we controlled some command line features of windows like shut-down, restart, screenshot, lock screen likes features through our hand gestures.

- And then we build the equivalent .exe of these features successfully.

- For the users to use these features, we build a responsive webpage from which user can download the respective .exe as per their need and use the features.

- Webpage URL: https://adityapandey1111.github.io/Human-Computer-Interaction-using-Hand-Gesture/

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to our guide, Mrs. BIAS BHADRA her valuable guidance, consistent encouragement, personal caring, timely help and providing us with an excellent atmosphere for doing the project. All through the work, in spite of her busy schedule, she has extended cheerful and cordial support to us for completing this project work.

....

Signature of the Students

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the advancement of information technology in our culture, we can anticipate a greater integration of computer systems into our surroundings. These circumstances will necessitate new sorts of human-computer interaction, with natural and simple interfaces. As an outcome, human-computer interaction has become an essential part of our daily lives. As a human being, we can no longer ignore the effect of engagement with computing equipment, and it has become our primary concern, as direct use of hands is indeed a natural way for humans to interact with each other and, more recently, with new tech in intelligent environment.

The foundation for putting such a system in place is gesture recognition. For years, gesture recognition was a hot topic. In today's world, there are essentially two approaches for recognizing gestures. Use of gestures as a human-computer interaction communication interface is an appealing option to these cumbersome interface devices. Gesture recognition research aims to develop a system that can recognize certain human movements and utilize them to convey information or control device s. complicated surroundings or real-life settings in which the user wants to utilize the programmed on the go after the system has recognized the picture regions, the image regions can be processed to determine the hand position.

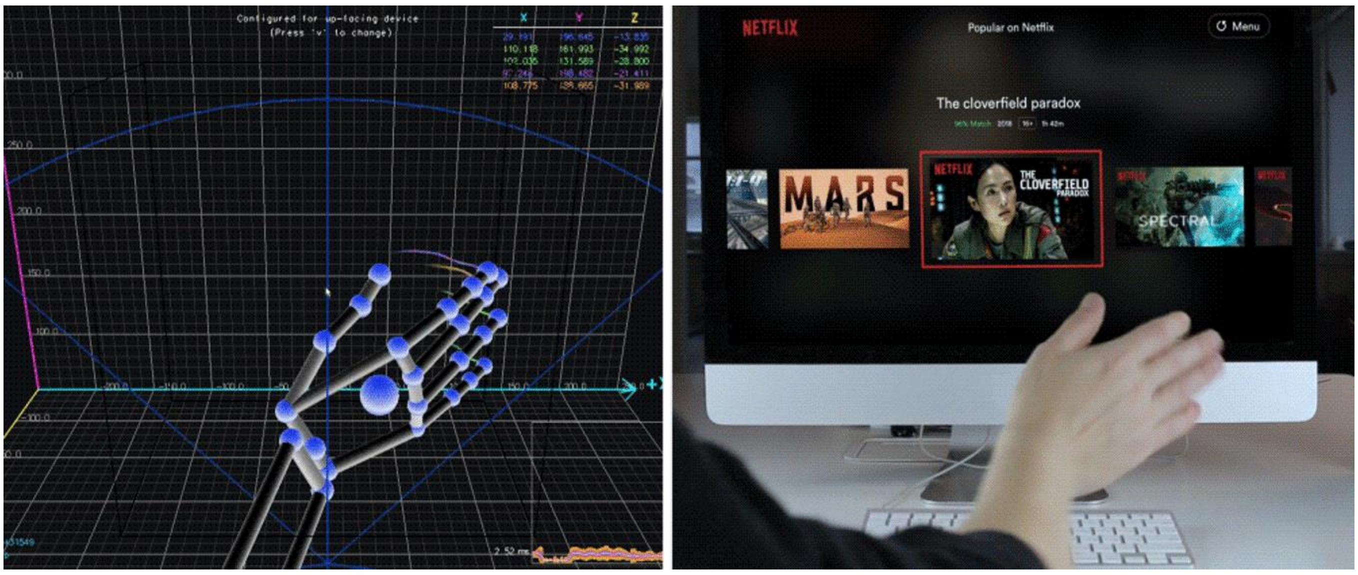


Figure 1.1. Example of gesture controlling device

Despite the fact that the academics and engineers who invented the mouse and keyboards made significant progress, there are still some scenarios when using a keyboard and mouse to connect with a computer is insufficient. A great goal in human computer interaction has been to migrate "natural" ways in human interaction. For a decade, human speech recognition has been the focus of research.

We plan to investigate the usage of hand gestures for interaction in the project, using a computer vision approach. Later, we'll go over the application in greater depth in this paper.

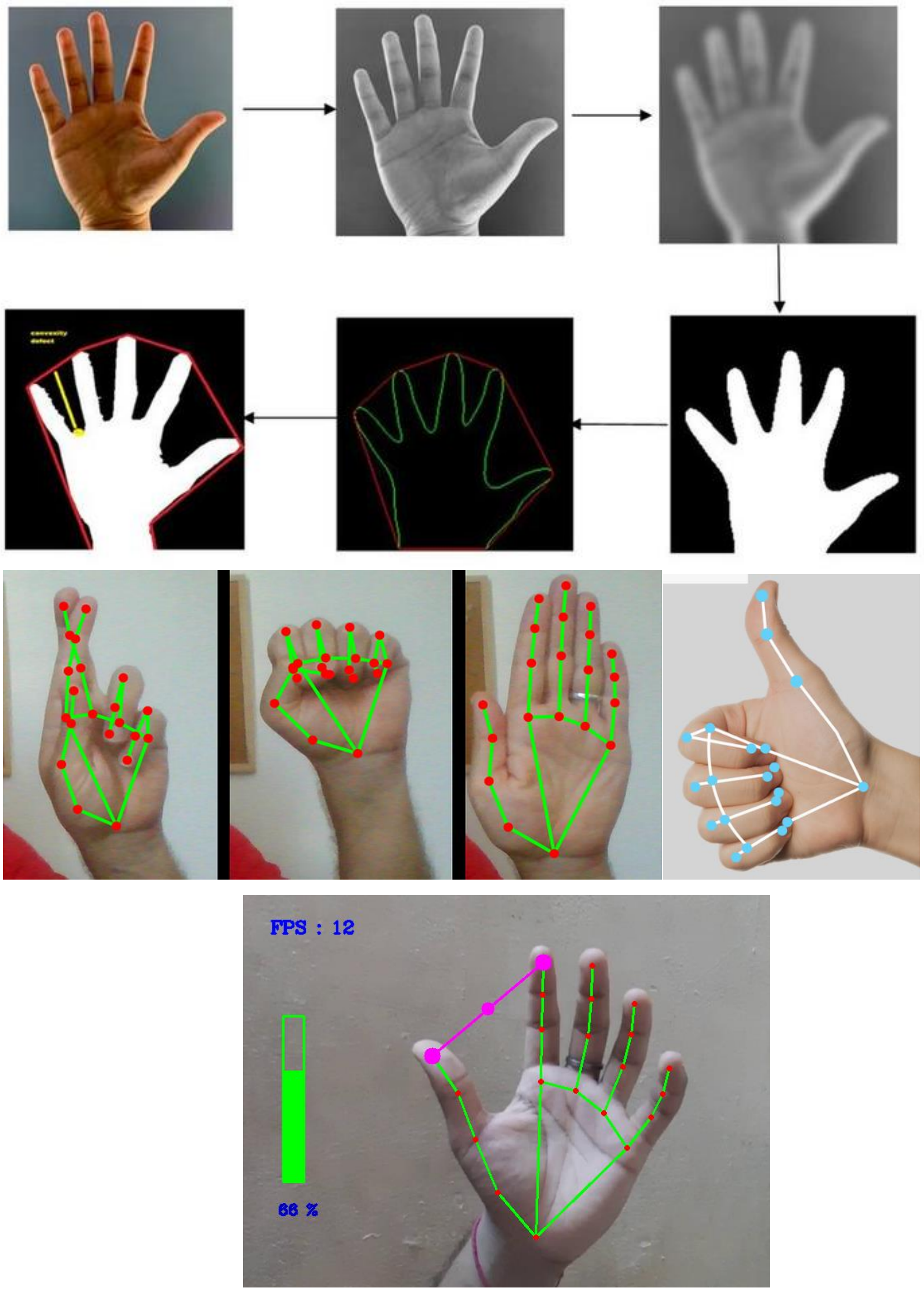


Figure 1.2. Example of hand gesture detection

CHAPTER 2

LITERATURE SURVEY

1. **Interaction with computer using hand gesture with the help of Arduino and Sensors:** The starting approach of interaction with computer using hand gesture was first projected by Myron W. Krueger in 1970. The aim of the perspective was achieved and also the mouse cursor control was performed with the help of an Arduino and sensors, that would interpret hand gestures and then turned the recognized gestures into OS commands that controlled the mouse actions on the display screen of the computer device. Detection and recognition of a particular human body gesture and carry information to the computer is the main objective of gesture recognition. Overall objective of this system was to create the human gestures which can be admit by computer device to control a good sort of devices that are at distant using different hand poses.

   Research Details: Myron W. Krueger in 1970

2. **Hand Gesture Recognition Using PCA in:** In this paper authors presented a scheme using a database driven hand gesture recognition based upon skin color model approach and thresholding approach along with an effective template matching with can be effectively used for human robotics applications and similar other applications. Initially, hand region is segmented by applying skin color model in YCbCr color space. In the next stage thresholding is applied to separate foreground and background. Finally, template based matching technique is developed using Principal Component Analysis (PCA) for recognition.

   Research Details: Mandeep Kaur Ahuja, Amardeep Singh, "Hand Gesture Recognition Using PCA", International Journal of Computer Science Engineering and Technology (IJCSET), Volume 5, Issue 7, pp. 267-27, July 2015.

3. **Hand Gesture Recognition for Sign Language Recognition:** A Review in Authors presented various method of hand gesture and sign language recognition proposed in the past by various researchers. For deaf and dumb people, Sign language is the only way of communication. With the help of sign language, these physical impaired people express their emotions and thoughts to another person.

   Research Details: Pratibha Pandey, Vinay Jain, "Hand Gesture Recognition for Sign Language Recognition: A Review", International Journal of Science, Engineering and Technology Research (IJSETR), Volume 4, Issue 3, March 2015.

The above paper deals with the different algorithm and techniques used for recognizing the hand gesture. Hand gesture recognition system is considered as a way for more intuitive and proficient human computer interaction tool. The range of applications includes virtual prototyping and sign language analysis. From the above consideration it is clear that the vision-based hand gesture recognition has made remarkable progress in the field of hand gesture recognition. The aim of the perspective was achieved and also the mouse cursor control was performed with the help of an Arduino and sensors.

4. **Interaction with computer using hand gesture with the help of Arduino and Sensors**
   Author : Myron W. Krueger
   Date of Publish : Mid of 1970's

5. **Hand Gesture Recognition Using PCA**
   Author : Mandeep Kaur Ahuja, Amardeep Singh
   Date of Publish : Mid of 1970's

6. **Hand Gesture Recognition for Sign Language Recognition**
   Author : Pratibha Pandey, Vinay Jain
   Date of Publish : Mid of 1970's

7. **Hand Gestures Recognition from International Journal of Artificial Intelligence & Applications (IJAIA)**
   Author : Rafiqul Zaman Khan, Noor Adnan Ibraheem
   Date of Publish : Mid of 1970's

8. **Hand Gesture Recognition System Using Camera**
   Author : Viraj Shinde, Tushar Bacchav, Jitendra Pawar, Mangesh Sanap
   Date of Publish : January 2014

CHAPTER 3

METHODOLOGY

3.1. BASIC METHODOLOGY OF HAND GESTURE DETECTION

We conceived and designed a Basic system for vision-based hand gesture identification that included several stages that we explained using an algorithm. Figure 3.1 depicts the functioning flowchart of the gesture recognition system.

This experiment was carried out in a controlled environment with good lighting and a plain background devoid of any other skin-like things. A command can be given to any multimedia application running on the computer based on the identified gesture to control its numerous activities.

Below is the flow chart of Hand Gesture Detection using OpenCV and Gaussian Mixture-based Background/Foreground Segmentation Algorithm to subtract background.

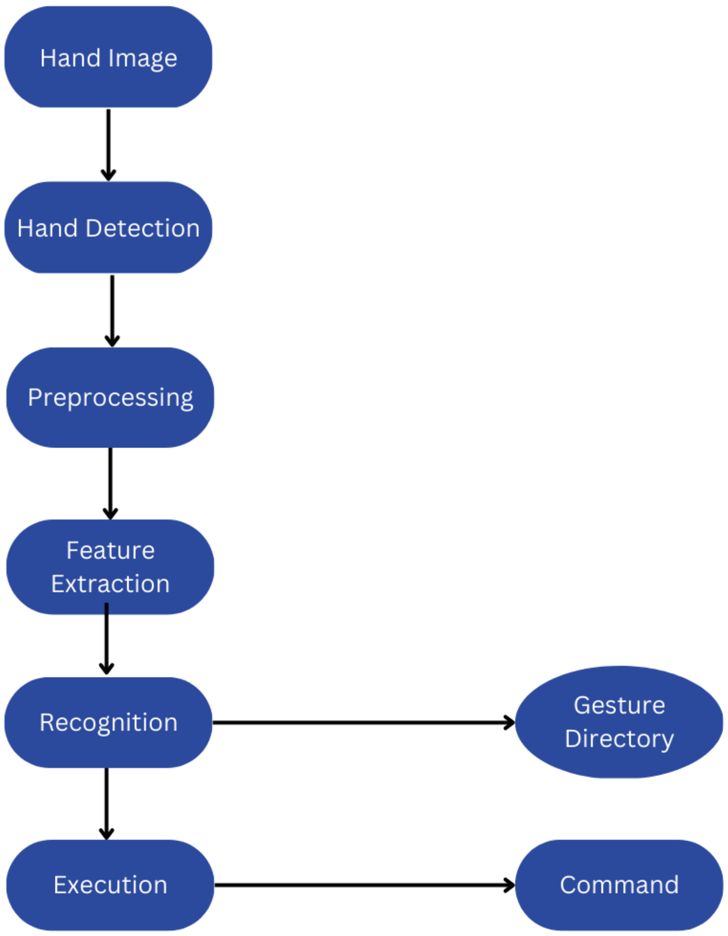


Figure. 3.1. Flowchart of Gesture Detection (hand gesture)

3.2. PROPOSED METHODOLOGY OF OUR APPROACH

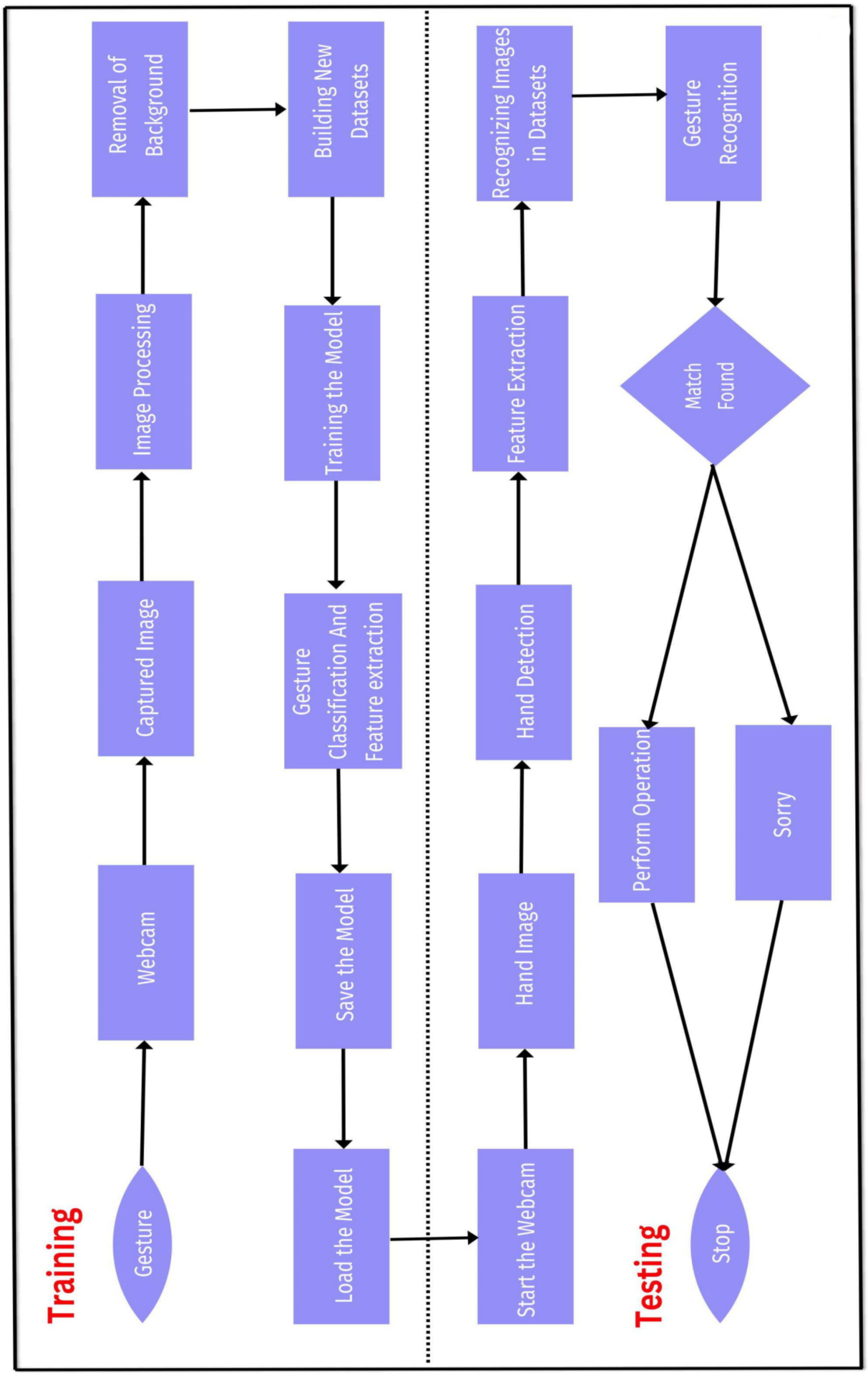


Figure 3.2. Proposed Methodology of Gesture Detection (hand gesture)

Here is a step-wise methodology for device control using hand gestures explaining the flowchart figure 3.2.

This flow chart is divided into two phases:

- Training Phase
- Testing Phase

A general explanation of overall methodology is mentioned below as per figure 3.2.

1. **Image Acquisition:**
   - Capture a live video feed of the hand using a camera or depth sensor.

2. **Image Pre-processing:**
   - Convert the captured image to grayscale or other Color spaces as needed.
   - Apply filters to reduce noise and enhance the edges of the hand.
   - Perform background subtraction to separate the hand from the background.

3. **Hand Detection:**
   - Use computer vision techniques such as blob detection, edge detection, or skin Color detection to detect the hand region in the image.

4. **Hand Segmentation:**
   - Isolate the hand region from the background by segmenting the hand region from the rest of the image.

5. **Hand Feature Extraction:**
   - Extract relevant features from the hand region, such as size, shape, orientation, and movement.
   - Use techniques such as contour detection, convex hull detection, and principal component analysis (PCA) to extract hand features.

6. **Hand Gesture Recognition:**
   - Classify the hand gesture based on the extracted features using techniques such as machine learning, deep learning, or rule-based algorithms.
   - Train a model using labelled data to recognize hand gestures accurately.

7. **Device Control:**
   - Send commands or perform actions on the device based on the recognized hand gesture.
   - Map different hand gestures to specific device actions or commands.

8. **Performance Evaluation:**
   - Evaluate the performance of the system by measuring accuracy and response time of the gesture recognition system.
   - Refine and improve the system by modifying the algorithm and retraining the model if necessary.

3.3. ALGORITHM

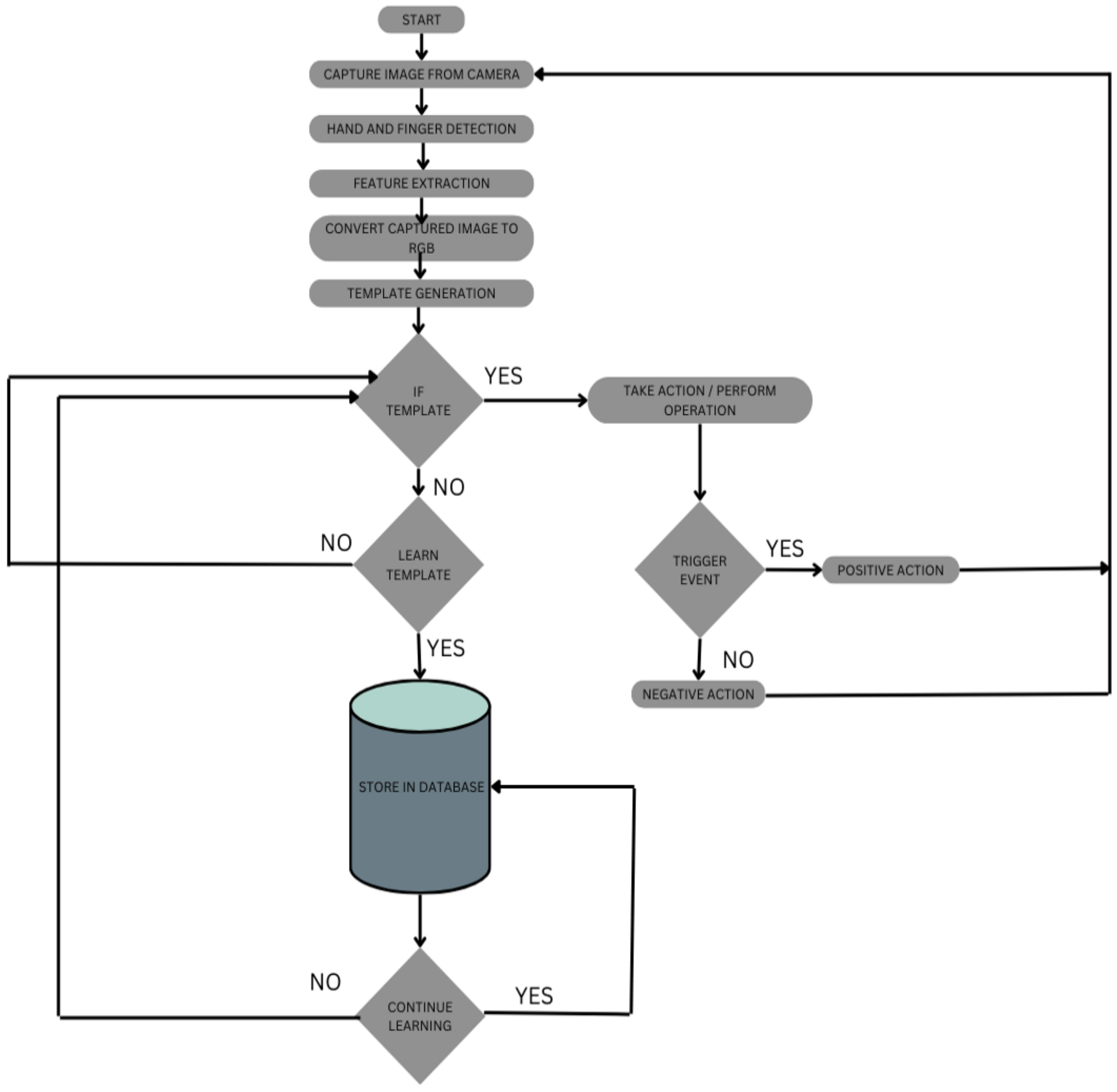


Figure 3.3. Algorithms of HCI

Algorithms we followed in our implementation:

- Import necessary libraries like OpenCV, Mediapipe, Numpy, and screen_brightness_control.
- Create a VideoCapture object to check the camera.

- Initialize the hands module of Mediapipe.
- Create a named window and start a thread to manage the window.
- Start an infinite loop.
- Read a frame from the camera.
- Convert the frame from BGR to RGB color space.
- Process the RGB frame with hands module of Mediapipe and get the landmarks of hands detected in the frame.
- If landmarks are detected, draw them on the frame.
- If fingertips are detected, calculate the distance between them.
- Convert the distance value from the hand range (30-350) to respective features range.
- Display the frame in the named window.
- If the spacebar is pressed, break the loop.
- Release the camera and close the window.

3.4. DETAILED PROCESS AND IMPLEMENTATION

A. Environment and Operating System

OS: Windows

Platform: Python 3

Libraries: OpenCV 3, mediapipe==0.8.3, tensorflow==

Platforms required: Visual Studio Code

B. Requirements from Python and OpenCV module

import cv2

import numpy as np

import copy

import math

import pyautogui

import mediapipe as mp

import time

C. How to run it?

Step 1: Open any Web browser and Type our "https://adityapandey1111.github.io/Human-Computer-Interaction-using-Hand-Gesture/" in your browser search box, OR, Click the URL given below.

Website Link: https://adityapandey1111.github.io/Human-Computer-Interaction-using-Hand-Gesture/

Step 2: Navigate through the webpage and find the download button below the python .exe file in which you are interested.

Step 3: Click Download, in few seconds it will automatically start downloading.

Step 4: Now open the downloaded .exe file and run it.

Step 5: Make sure that your webcam is in working condition.

Anyways if it is not working properly an error message will be prompted to you.

Step 6: Show your hand gesture to the webcam as it is mentioned in the documentation of the project. The program will respond accordingly and you will see related output.

Step 7: To close the program either you terminate directly from the "Task Manager" or simply click the "Quit"(button mentioned in the program for closing the app) button.

Step 8: Program terminated.

3.5. DETAILED PROGRAM AND CODE

Here is the complete code for each featured .exe is mentioned:

<u>File-1. Brightness_Control.py</u>

```
import cv2
import mediapipe as mp
import numpy as np
import screen_brightness_control as sbc
from math import hypot
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

cap = cv2.VideoCapture(0)

mpHands = mp.solutions.hands
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils

brightness = 0
brightbar = 400
brightper = 0

cv2.namedWindow("Image", cv2.WINDOW_AUTOSIZE)
cv2.startWindowThread()
cv2.moveWindow("Image", -10000, -10000)

while True:
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    results = hands.process(imgRGB)

    lmList = []
    if results.multi_hand_landmarks:
        for handlandmark in results.multi_hand_landmarks:
            for id, lm in enumerate(handlandmark.landmark):
                h, w, _ = img.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                lmList.append([id, cx, cy])
            mpDraw.draw_landmarks(img, handlandmark,
mpHands.HAND_CONNECTIONS)

    if lmList != []:
        x1, y1 = lmList[4][1], lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cv2.circle(img, (x1, y1), 13, (255, 0, 0), cv2.FILLED)
        cv2.circle(img, (x2, y2), 13, (255, 0, 0), cv2.FILLED
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 2

        length = hypot(x2 - x1 - 30, y2 - y1 - 30)

        brightness = int(np.interp(length, [30, 350], [0, 255]))
        brightbar = int(np.interp(length, [30, 350], [400, 150]))
        brightper = int(np.interp(length, [12, 350], [0, 100]))
        sbc.set_brightness(int(brightper))

        print(brightness, int(length))
        cv2.rectangle(img, (50, 150), (85, 400), (0, 0, 255),4
        cv2.rectangle(img, (50, int(brightbar)), (85, 400), (0, 0,
255), cv2.FILLED)
        cv2.putText(img, f"{int(brightper)}%", (10, 40),
cv2.FONT_ITALIC, 1, (0, 255, 98), 3)

    cv2.imshow('Image', img)
    if cv2.waitKey(1) & 0xff == ord(' '
        break

cap.release()
cv2.destroyAllWindows
```

File-2. Volume_Control.py

```
import cv2
import mediapipe as mp
from math import hypot
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
import numpy as np

cap = cv2.VideoCapture(0)

mpHands = mp.solutions.hands
hands = mpHands.Hands()
```

```
mpDraw = mp.solutions.drawing_utils

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL,
None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
volbar = 400
volper = 0

volMin, volMax = volume.GetVolumeRange()[:2]

cv2.namedWindow("Image", cv2.WINDOW_AUTOSIZE)
cv2.startWindowThread()
cv2.moveWindow("Image", -10000, -10000)

while True:
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    results = hands.process(imgRGB)

    lmList = []
    if results.multi_hand_landmarks:
        for handlandmark in results.multi_hand_landmarks:
            for id, lm in enumerate(handlandmark.landmark):
                h, w, _ = img.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                lmList.append([id, cx, cy])
            mpDraw.draw_landmarks(img, handlandmark,
mpHands.HAND_CONNECTIONS)

    if lmList != []:
        x1, y1 = lmList[4][1], lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cv2.circle(img, (x1, y1), 13, (255, 0, 0), cv2.FILLED)
        cv2.circle(img, (x2, y2), 13, (255, 0, 0), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3

        length = hypot(x2 - x1, y2 - y1
        vol = np.interp(length, [30, 350], [volMin, volMax])
        volbar = np.interp(length, [30, 350], [400, 150])
        volper = np.interp(length, [30, 350], [0, 100])

        print(vol, int(length))
        volume.SetMasterVolumeLevel(vol, None)

        cv2.rectangle(img, (50, 150), (85, 400), (0, 0, 255),4
        cv2.rectangle(img, (50, int(volbar)), (85, 400), (0, 0, 255),
cv2.FILLED)
        cv2.putText(img, f"{int(volper)}%", (10, 40),
cv2.FONT_ITALIC, 1, (0, 255, 98), 3)

    cv2.imshow('Image', img)
    if cv2.waitKey(1) & 0xff == ord(' '):
        break

cap.release()
cv2.destroyAllWindows()
```

File-3. Virtual_Mouse.py

```
import cv2
import time
import mediapipe as mp
import autopy
import numpy as np
import math

width_cam = 640
height_cam = 480
frame_reduction = 100
smoothen = 6

p_time = 0
p_loca_x, p_loca_y = 0, 0

cap = cv2.VideoCapture(0)
cap.set(3, height_cam)
cap.set(4, width_cam)

mp_hand = mp.solutions.hands
hand = mp_hand.Hands(max_num_hands=1)
mp_draw = mp.solutions.drawing_utils

width_screen, height_screen = autopy.screen.size()

cv2.namedWindow("Image", cv2.WINDOW_AUTOSIZE)
cv2.startWindowThread()
```

```
cv2.moveWindow("Image", -10000, -10000)

while True:
    success, img = cap.read()
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    result = hand.process(img_rgb)
    if result.multi_hand_landmarks:
        lm_list = []
        x_list = []
        y_list = []
        for hand_lm in result.multi_hand_landmarks:
            for id, lm in enumerate(hand_lm.landmark):
                height, width, channel = img.shape
                x, y = int(lm.x * width), int(lm.y * height)
                x_list.append(x)
                y_list.append(y)
                lm_list.append([id, x, y])
                mp_draw.draw_landmarks(img, hand_lm,
mp_hand.HAND_CONNECTIONS)

                x_min, x_max = min(x_list), max(x_list)
                y_min, y_max = min(y_list), max(y_list)
            cv2.rectangle(
                img, (x_min - 20, y_min - 20),
                (x_max + 20, y_max + 20),
                (0, 255, 0), 2
            )
        if len(lm_list) > 15:
            x1, y1 = lm_list[8][1:]
            x2, y2 = lm_list[12][1:]

            cv2.rectangle(img, (frame_reduction, frame_reduction),
                          (width_cam - frame_reduction, height_cam -
frame_reduction),
                          (255, 255, 0), 3)

            if y1 < lm_list[6][2] and y2 > lm_list[10][2]:
                x3 = np.interp(x1, (frame_reduction, width_cam -
frame_reduction), (0, width_screen))
                y3 = np.interp(y1, (frame_reduction, height_cam -
frame_reduction), (0, height_screen))

                c_loca_x = p_loca_x + (x3 - p_loca_x) / smoothen
                c_loca_y = p_loca_y + (y3 - p_loca_y) / smoothen
                autopy.mouse.move(width_screen - c_loca_x, c_loca_y)
                cv2.circle(
                    img, (x1, y1), 10,
                    (255, 0, 255), cv2.FILLED
                )
                p_loca_x, p_loca_y = c_loca_x, c_loca_y

            elif y1 < lm_list[6][2] and y2 < lm_list[10][2]:
                length = math.hypot(x2 - x1, y2 - y1)
                if length < 40:
                    cv2.circle(
                        img, (x1, y1), 10,
                        (0, 255, 0), cv2.FILLED
                    )
                    autopy.mouse.click()

    c_time = time.time()
    fps = 1 / (c_time - p_time)
    p_time = c_time
    cv2.putText(
        img, f'FPS: {int(fps)}', (10, 50),
        cv2.FONT_HERSHEY_PLAIN, 2,
        (255, 0, 0), 2
    )
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

<u>File-4. System_OS.py</u>

```
import cv2
import numpy as np
import mediapipe as mp
import tensorflow as tf
from tensorflow.keras.models import load_model
import os
import pyautogui

mpHands = mp.solutions.hands
hands = mpHands.Hands(max_num_hands=1, min_detection_confidence=0.7)
mpDraw = mp.solutions.drawing_utils
model = load_model('SystemOS\mp_hand_gesture')
f = open('SystemOS\gesture.names', 'r')
classNames = f.read().split('\n')
f.close()
```

```
print(classNames)

cap = cv2.VideoCapture(0)

cv2.namedWindow("Image", cv2.WINDOW_AUTOSIZE)
cv2.startWindowThread()
cv2.moveWindow("Image", -10000, -10000)

while True:
    _, frame = cap.read()
    x, y, c = frame.shape

    frame = cv2.flip(frame, 1)
    framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    result = hands.process(framergb)

    className = ''

    if result.multi_hand_landmarks:
        landmarks = []
        for handslms in result.multi_hand_landmarks:
            for lm in handslms.landmark:
                lmx = int(lm.x * x)
                lmy = int(lm.y * y)

                landmarks.append([lmx, lmy])

            mpDraw.draw_landmarks(frame, handslms,
mpHands.HAND_CONNECTIONS)

            prediction = model.predict([landmarks])
            classID = np.argmax(prediction)
            className = classNames[classID]

            if className == 'rock':
                os.system("rundll32.exe user32.dll,LockWorkStation")
                break

            elif className == 'thumbs up':
                os.system("shutdown /s /t 1")
                break

            elif className == 'stop':
                os.system("shutdown /r /t 1")
                break

            elif className == 'fist':
                pyautogui.hotkey('win','PrtScr')
                break

    cv2.putText(frame, className, (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 
1, (0,0,255), 2, cv2.LINE_AA)


    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

File-5. Virtual_Keyboard.py

```
from tkinter import *
import tkinter.font as font
from tkinter import messagebox
from sys import exit as end
from os import system

# if user has the keyboard module installed
has_keyboard = True

try:
    import keyboard
except (ModuleNotFoundError, ImportError):
    # user doesn't have keyboard module installed
    dummy = Tk()
    dummy.withdraw()
    messagebox.showwarning('Missing Module: keyboard', 'Your system 
is missing the module "keyboard" for this program to work 
correctly.\n\nPlease click OK to install the "keyboard" module 
automatically.\nIn case this fails, the keyboard will still open in a 
non functional state')
    kbmodulestatus = system('python -m pip install keyboard')
    if kbmodulestatus != 0:
        messagebox.showerror('Error', 'Couldn\'t install "keyboard" 
module automatically. Please try again manually in command prompt 
using command:\n\npip install keyboard')
        dummy.destroy()
        has_keyboard = False
    else:
        import keyboard
        dummy.destroy()
        has_keyboard = True

class VirtualKeyboard:

    def __init__(self, master=Tk()):
        # Main Window
        self.master = master

        # prevent from crash if photo isn't found
        try:
            vkblogo = PhotoImage(file="vkblogo.png")
            self.master.iconphoto(True, vkblogo)
        except TclError:
            # logo not found locally
            pass

        # Colors
        self.darkgray = "#242424"
        self.gray = "#383838"
        self.darkred = "#591717"
        self.red = "#822626"
        self.darkpurple = "#7151c4"
        self.purple = "#9369ff"
        self.darkblue = "#386cba"
        self.blue = "#488bf0"
        self.darkyellow = "#bfb967"
        self.yellow = "#ebe481"

        self.master.configure(bg=self.gray)
        self.unmap_bind = self.master.bind("<Unmap>", lambda e:
[self.rel_win(), self.rel_alts(), self.rel_shifts(),
self.rel_ctrls()])

        # makes sure shift/ctrl/alt/win keys aren't pressed down
after keyboard closed
        self.master.protocol("WM_DELETE_WINDOW", lambda:
[self.master.destroy(), end()])
        self.master.title("Virtual Keyboard (NON FUNCTIONAL)")

        self.user_scr_width = int(self.master.winfo_screenwidth())
        self.user_scr_height = int(self.master.winfo_screenheight())
        self.trans_value = 0.7
        self.master.attributes('-alpha', self.trans_value)
        self.master.attributes('-topmost', True)

        # avoid keyboard size conflicts for screens with lower
resolution
        self.size_value_map = [
            (int(0.63 * self.user_scr_width), int(0.37 *
self.user_scr_height)),
            (int(0.70 * self.user_scr_width), int(0.42 *
self.user_scr_height)),
            (int(0.78 * self.user_scr_width), int(0.46 *
self.user_scr_height)),
            (int(0.86 * self.user_scr_width), int(0.51 *
self.user_scr_height)),
            (int(0.94 * self.user_scr_width), int(0.56 *
self.user_scr_height))
        ]
        self.size_value_names = ["Very Small", "Small", "Medium",
"Large", "Very Large"]

        # index for both size value map and size value names
        self.size_current = 2

        # open keyboard in medium size by default (not resizable)

self.master.geometry(f"{self.size_value_map[self.size_current][0]}x{s
elf.size_value_map[self.size_current][1]}")
        self.master.resizable(False, False)

        # keys in every row
        self.row1keys = ["esc", "f1", "f2", "f3", "f4", "f5", "f6",
"f7", "f8", "f9", "f10",
                         "f11", "f12", "print_screen", "scroll_lock",
"numlock"]

        self.row2keys = ["`", "1", "2", "3", "4", "5", "6", "7",
                         "8", "9", "0", "-", "=", "backspace",
"page_up"]

        self.row3keys = ["tab", "q", "w", "e", "r", "t", "y", 'u',
                         'i', 'o', 'p', '[', ']', 'enter',
'page_down']
```

```
        self.row4keys = ["capslock", 'a', 's', 'd', 'f', 'g', 'h', 
'j',
                         'k', 'l', ';', "'", '\\', 'delete', 'home', 
'end']

        self.row5keys = ["left shift", 'z', 'x', 'c', 'v', 'b', 'n', 
'm',
                         ',', '.', '/', 'right shift', 'up', 
'insert']

        self.row6keys = ["left ctrl", 'win', 'alt', 'spacebar', 'alt 
gr',
                         'right ctrl', ':)', 'left', 'down', 'right']

        # buttons for each row
        self.row1buttons = []
        self.row2buttons = []
        self.row3buttons = []
        self.row4buttons = []
        self.row5buttons = []
        self.row6buttons = []

        # efficiency i guess?
        appendrow1 = self.row1buttons.append
        appendrow2 = self.row2buttons.append
        appendrow3 = self.row3buttons.append
        appendrow4 = self.row4buttons.append
        appendrow5 = self.row5buttons.append
        appendrow6 = self.row6buttons.append

        # prevents frames having inconsistent relative dimensions
        self.master.columnconfigure(0, weight=1)
        for i in range(7):
            self.master.rowconfigure(i, weight=1)

        # Create fonts acc to resolution
        if self.user_scr_width < 1600:
            self.keyfont = font.Font(family="Calibri", size=10, 
weight='bold')
            self.bottomfont = font.Font(family='Calibri', size=11, 
weight='bold')
            self.neetfont = font.Font(family='Lucida Handwriting', 
size=8, weight='normal')
```

```
        else:
            self.keyfont = font.Font(family="Calibri", size=13,
weight='bold')
            self.bottomfont = font.Font(family='Calibri', size=13,
weight='bold')
            self.neetfont = font.Font(family='Lucida Handwriting',
size=10, weight='normal')

        # spl_key_pressed is True if ALT, CTRL, SHIFT or WIN are held
down using right click
        # if it is False, the 4 mentioned keys get released on
clicking any other key
        self.spl_key_pressed = False

        #   ROW 1

        # create a frame for row1buttons
        keyframe1 = Frame(self.master, height=1)
        keyframe1.rowconfigure(0, weight=1)

        # create row1buttons
        for key in self.row1keys:
            ind = self.row1keys.index(key)
            keyframe1.columnconfigure(ind, weight=1)
            appendrow1(Button(
                keyframe1,
                font=self.keyfont,
                border=7,
                bg=self.gray,
                activebackground=self.darkgray,
                activeforeground="#bababa",
                fg="white",
                width=1,
                relief=RAISED
            ))
            if key == "print_screen":
                self.row1buttons[ind].config(text="PrtScr", width=3,
height=2)
            elif key == "scroll_lock":
                self.row1buttons[ind].config(text="ScrLck", width=3)
            elif key == "numlock":
                self.row1buttons[ind].config(text="NumLck", width=3)
            else:
                self.row1buttons[ind].config(text=key.title())
            self.row1buttons[ind].grid(row=0, column=ind,
sticky="NSEW")

        #   ROW 2   #

        # create a frame for row2buttons
        keyframe2 = Frame(self.master, height=1)
        keyframe2.rowconfigure(0, weight=1)

        # create row2buttons
        for key in self.row2keys:
            ind = self.row2keys.index(key)
            if ind == 13:
                keyframe2.columnconfigure(ind, weight=2)
            else:
                keyframe2.columnconfigure(ind, weight=1)
            appendrow2(Button(
                keyframe2,
                font=self.keyfont,
                border=7,
                bg=self.gray,
                activebackground=self.darkgray,
                activeforeground="#bababa",
                fg="white",
                width=1,
                relief=RAISED
            ))
            if key == "page_up":
                self.row2buttons[ind].config(text="Pg Up", width=2)
            elif key == "backspace":
                self.row2buttons[ind].config(text=key.title(),
width=4)

            self.row2buttons[ind].grid(row=0, column=ind,
sticky="NSEW")

        self.row2buttons[0].config(text="~\n`")
        self.row2buttons[1].config(text="!\n1")
        self.row2buttons[2].config(text="@\n2")
        self.row2buttons[3].config(text="#\n3")
        self.row2buttons[4].config(text="$\n4")
        self.row2buttons[5].config(text="%\n5")
        self.row2buttons[6].config(text="^\n6")
        self.row2buttons[7].config(text="&\n7")
        self.row2buttons[8].config(text="*\n8")
        self.row2buttons[9].config(text="(\n9")
        self.row2buttons[10].config(text=")\n0")
        self.row2buttons[11].config(text="_\n-")
        self.row2buttons[12].config(text="+\n=")

        #   ROW 3   #

        # create a frame for row3buttons
        keyframe3 = Frame(self.master, width=1)
        keyframe3.rowconfigure(0, weight=1)

        # create row3buttons
        for key in self.row3keys:
            ind = self.row3keys.index(key)
            if ind == 13:
                keyframe3.columnconfigure(ind, weight=2)
            else:
                keyframe3.columnconfigure(ind, weight=1)
            appendrow3(Button(
                keyframe3,
                font=self.keyfont,
                border=7,
                bg=self.gray,
                activebackground=self.darkgray,
                activeforeground="#bababa",
                fg="white",
                width=1,
                relief=RAISED
            ))
            if key == "page_down":
                self.row3buttons[ind].config(text="Pg Dn", width=2)
            elif key == "[":
                self.row3buttons[ind].config(text="{\n[", width=1)
            elif key == "]":
                self.row3buttons[ind].config(text="}\n]", width=1)
            elif key == "tab":
                self.row3buttons[ind].config(text="Tab", width=3)
            elif key == "enter":
                self.row3buttons[ind].config(text="Enter", width=3)
            else:
                self.row3buttons[ind].config(text=key.title())
```

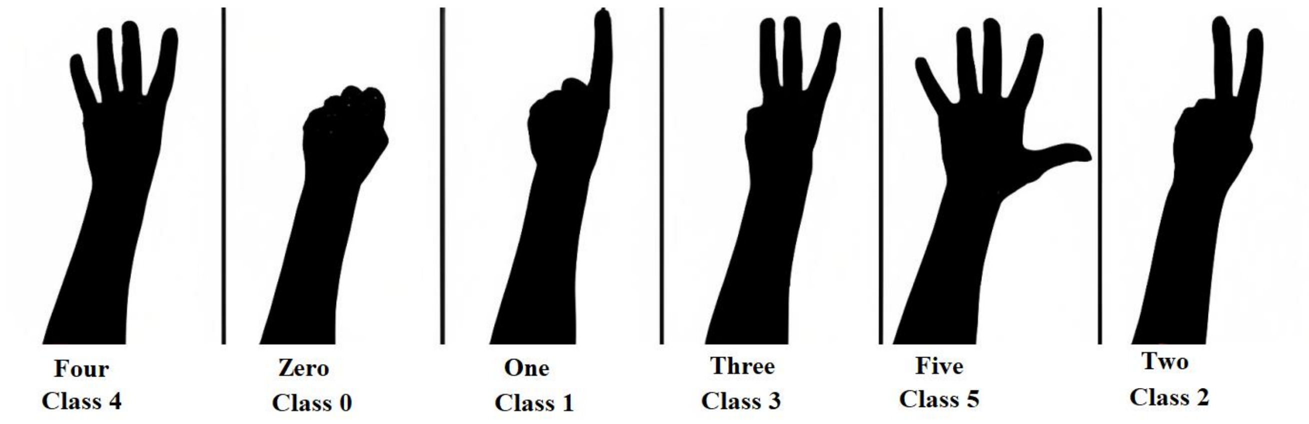
```
            self.row3buttons[ind].grid(row=0, column=ind,
sticky="NSEW")

        #   ROW 4   #

        # create a frame for row4buttons
        keyframe4 = Frame(self.master, height=1)
        keyframe4.rowconfigure(0, weight=1)

        # create row4buttons
        for key in self.row4keys:
            ind = self.row4keys.index(key)
            keyframe4.columnconfigure(ind, weight=1)
            appendrow4(Button(
                keyframe4,
                font=self.keyfont,
                border=7,
                bg=self.gray,
                activebackground=self.darkgray,
                activeforeground="#bababa",
                fg="white",
                width=2,
                relief=RAISED
            ))
            if key == ";":
                self.row4buttons[ind].config(text=":\n;")
            elif key == "'":
                self.row4buttons[ind].config(text='"\n\'')
            elif key == "\\":
                self.row4buttons[ind].config(text="|\n\\")
            elif key == "capslock":
                self.row4buttons[ind].config(text="CapsLck", width=5)
            else:
                self.row4buttons[ind].config(text=key.title())

            self.row4buttons[ind].grid(row=0, column=ind,
sticky="NSEW")

        #   ROW 5   #

        # create a frame for row5buttons
        keyframe5 = Frame(self.master, height=1)
        keyframe5.rowconfigure(0, weight=1)
        # create row5buttons
        for key in self.row5keys:
            ind = self.row5keys.index(key)
            if ind == 0 or ind == 11:
                keyframe5.columnconfigure(ind, weight=3)
            else:
                keyframe5.columnconfigure(ind, weight=1)
            appendrow5(Button(
                keyframe5,
                font=self.keyfont,
                border=7,
                bg=self.gray,
                activebackground=self.darkgray,
                activeforeground="#bababa",
                fg="white",
                width=1,
                relief=RAISED
            ))
            if key == ",":
                self.row5buttons[ind].config(text="<\n,")
            elif key == ".":
                self.row5buttons[ind].config(text=">\n.")
            elif key == "/":
                self.row5buttons[ind].config(text="?\n/")
            elif key == "up":
                self.row5buttons[ind].config(text="↑")
            elif key == "insert":
                self.row5buttons[ind].config(text="Insert", width=1)
            elif key == "left shift":
                self.row5buttons[ind].config(text="Shift", width=6)
            elif key == "right shift":
                self.row5buttons[ind].config(text="Shift", width=6)
            else:
                self.row5buttons[ind].config(text=key.title())

            self.row5buttons[ind].grid(row=0, column=ind,
sticky="NSEW")

        #   ROW 6   #

        # create a frame for row6buttons
        keyframe6 = Frame(self.master, height=1)
        keyframe6.rowconfigure(0, weight=1)
        # create row6buttons
        for key in self.row6keys:
            ind = self.row6keys.index(key)
            if ind == 3:
                keyframe6.columnconfigure(ind, weight=12)
            else:
                keyframe6.columnconfigure(ind, weight=1)
            appendrow6(Button(
                keyframe6,
                font=self.keyfont,
                border=7,
                bg=self.gray,
                activebackground=self.darkgray,
                activeforeground="#bababa",
                fg="white",
                width=1,
                relief=RAISED
            ))

            if key == "left":
                self.row6buttons[ind].config(text="←")
            elif key == "down":
                self.row6buttons[ind].config(text="↓")
            elif key == "right":
                self.row6buttons[ind].config(text="→")
            elif key == "spacebar":
                self.row6buttons[ind].config(text="\n")
            elif key == "win":
                self.row6buttons[ind].config(text="Win")
            elif key == "left ctrl":
                self.row6buttons[ind].config(text="Ctrl")
            elif key == "right ctrl":
                self.row6buttons[ind].config(text="Ctrl")
            elif key == "alt":
                self.row6buttons[ind].config(text="Alt")
            elif key == "alt gr":
                self.row6buttons[ind].config(text="Alt")
            elif key == ":)":
                self.row6buttons[ind].config(text=key, width=4,
bg=self.red, activebackground=self.darkred, command=self.donothing)
            else:
                self.row6buttons[ind].config(text=key.title())
            self.row6buttons[ind].grid(row=0, column=ind,
sticky="NSEW")

        # create final frame 7 for custom keys
        infoframe7 = Frame(self.master, height=1, bg=self.gray)
        infoframe7.rowconfigure(0, weight=1)

        # empty space
        infoframe7.columnconfigure(0, weight=1)
        self.tips_space = Button(infoframe7, text="Enjoy the buttons
:)", bg=self.gray, relief=FLAT, disabledforeground="white",
font=self.bottomfont, state=DISABLED, height=1)
        self.tips_space.grid(row=0, column=0, sticky="NSEW")

        # copy button
        infoframe7.columnconfigure(2, weight=1)
        self.copy_button = Button(
            infoframe7,
            font=self.bottomfont,
            border=5,
            bg=self.purple,
            text="COPY",
            activebackground=self.darkpurple,
            activeforeground="black",
            fg="black",
            relief=RAISED
        )
        self.copy_button.grid(row=0, column=2, padx=2, sticky="NSEW")

        # cut button
        infoframe7.columnconfigure(3, weight=1)
        self.cut_button = Button(
            infoframe7,
            font=self.bottomfont,
            border=5,
            bg=self.purple,
            text="CUT",
            activebackground=self.darkpurple,
            activeforeground="black",
            fg="black",
            relief=RAISED
        )
        self.cut_button.grid(row=0, column=3, padx=2, sticky="NSEW")
        # paste button
        infoframe7.columnconfigure(4, weight=1)
        self.paste_button = Button(
            infoframe7,
            font=self.bottomfont,
            border=5,
            bg=self.purple,
            text="PASTE",
            activebackground=self.darkpurple,
            activeforeground="black",
            fg="black",
            relief=RAISED
        )
        self.paste_button.grid(row=0, column=4, padx=2,
sticky="NSEW")

        # select all button
        infoframe7.columnconfigure(5, weight=1)
        self.selall_button = Button(
            infoframe7,
            font=self.bottomfont,
            border=5,
            bg=self.purple,
            text="SELECT ALL",
            activebackground=self.darkpurple,
            activeforeground="black",
            fg="black",
            relief=RAISED
        )
        self.selall_button.grid(row=0, column=5, padx=2,
sticky="NSEW")

        # task manager button
        infoframe7.columnconfigure(7, weight=1)
        self.taskmnger_button = Button(
            infoframe7,
            font=self.bottomfont,
            border=5,
            bg=self.blue,
            text="Task Manager",
            activebackground=self.darkblue,
            activeforeground="black",
            fg="black",
            relief=RAISED
        )
        self.taskmnger_button.grid(row=0, column=7, padx=2,
sticky="NSEW")

        # pin keyboard button
        infoframe7.columnconfigure(8, weight=1)
        self.pinkb_button = Button(
            infoframe7,
            font=self.bottomfont,
            border=5,
            bg=self.darkblue,
            text="Unpin Keyboard 📌",
            activebackground=self.blue,
            activeforeground="black",
            fg="black",
            width=15,
            relief=SUNKEN,
            command=self.keyboard_top)
        self.pinkb_button.grid(row=0, column=8, padx=2,
sticky="NSEW")

        # settings button
        infoframe7.columnconfigure(11, weight=1)
        self.settings_button = Button(
            infoframe7,
            font=self.bottomfont,
            border=5,
            bg=self.yellow,
            text="Keyboard Settings",
            activebackground=self.darkyellow,
            activeforeground="black",
            fg="black",
            relief=RAISED,
            command=self.kb_settings
        )
        self.settings_button.grid(row=0, column=11, padx=2,
sticky="NSEW")

        # decor
        infoframe7.columnconfigure(10, weight=1)
        Label(infoframe7, text="Made with Love...\nand python",
bg=self.gray, fg="white", font=self.neetfont).grid(row=0, column=10,
sticky="NSEW")
```

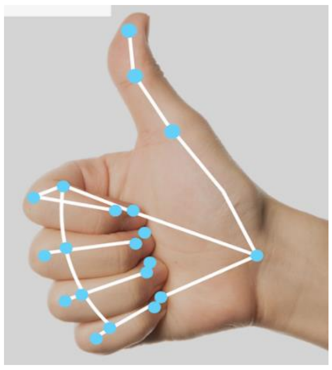
```
        # add the frames to the main window
        keyframe1.grid(row=0, sticky="NSEW", padx=9, pady=6)
        keyframe2.grid(row=1, sticky="NSEW", padx=9)
        keyframe3.grid(row=2, sticky="NSEW", padx=9)
        keyframe4.grid(row=3, sticky="NSEW", padx=9)
        keyframe5.grid(row=4, sticky="NSEW", padx=9)
        keyframe6.grid(row=5, padx=9, sticky="NSEW")
        infoframe7.grid(row=6, padx=9, pady=5, sticky="NSEW")

    # nothing
    def donothing(self):
        """
            This function is empty for now...
            maybe a new feature for the [ :) ] button
            """
        pass

    # an exception to get the symbols ? and _ from the keyboard
module's virtual hotkeys
    # for some reason "SHIFT+-" or "SHIFT+/" don't work :/
    def quest_press(self, x):
        if self.row5buttons[0].cget('relief') == SUNKEN:
            if x == "-":
                self.vpresskey("shift+_")
            elif x == "/":
                self.vpresskey("shift+?")
        else:
            self.vpresskey(x)

        if self.spl_key_pressed:
            keyboard.press('shift')

    # release shift keys
    def rel_shifts(self):
        keyboard.release('shift')

        self.row5buttons[0].config(
            relief=RAISED,
            bg=self.gray,
            activebackground=self.darkgray,
            activeforeground="#bababa",
            fg="white"
        )
        self.row5buttons[11].config(
            relief=RAISED,
            bg=self.gray,
            activebackground=self.darkgray,
            activeforeground="#bababa",
            fg="white"
        )

    # press shift keys
    def prs_shifts(self):
        keyboard.press('shift')

        self.row5buttons[0].config(
            relief=SUNKEN,
            activebackground=self.gray,
            bg=self.darkgray,
            fg="#bababa",
            activeforeground="white")
        self.row5buttons[11].config(
            relief=SUNKEN,
            activebackground=self.gray,
            bg=self.darkgray,
            fg="#bababa",
            activeforeground="white")

    # release ctrl keys
    def rel_ctrls(self):
        keyboard.release('ctrl')

        self.row6buttons[0].config(
            relief=RAISED,
            bg=self.gray,
            activebackground=self.darkgray,
            activeforeground="#bababa",
            fg="white"
        )
        self.row6buttons[5].config(
            relief=RAISED,
            bg=self.gray,
            activebackground=self.darkgray,
            activeforeground="#bababa",
            fg="white"
        )

    # press ctrl keys
    def prs_ctrls(self):
        keyboard.press('ctrl')

        self.row6buttons[0].config(
            relief=SUNKEN,
            activebackground=self.gray,
            bg=self.darkgray,
            fg="#bababa",
            activeforeground="white")
        self.row6buttons[5].config(
            relief=SUNKEN,
            activebackground=self.gray,
            bg=self.darkgray,
            fg="#bababa",
            activeforeground="white")

    # release alt keys
    def rel_alts(self):
        keyboard.release('alt')

        self.row6buttons[2].config(
            relief=RAISED,
            bg=self.gray,
            activebackground=self.darkgray,
            activeforeground="#bababa",
            fg="white"
        )
        self.row6buttons[4].config(
            relief=RAISED,
            bg=self.gray,
            activebackground=self.darkgray,
            activeforeground="#bababa",
            fg="white"
        )

    # press alt keys
    def prs_alts(self):
        keyboard.press('alt')

        self.row6buttons[2].config(
            relief=SUNKEN,
            activebackground=self.gray,
            bg=self.darkgray,
            fg="#bababa",
            activeforeground="white")
        self.row6buttons[4].config(
            relief=SUNKEN,
            activebackground=self.gray,
            bg=self.darkgray,
            fg="#bababa",
            activeforeground="white")

    # release win key
    def rel_win(self):
        keyboard.release('win')

        self.row6buttons[1].config(
            relief=RAISED,
            bg=self.gray,
            activebackground=self.darkgray,
            activeforeground="#bababa",
            fg="white"
        )

    # press win key
    def prs_win(self):
        keyboard.press('win')

        self.row6buttons[1].config(
            relief=SUNKEN,
            activebackground=self.gray,
            bg=self.darkgray,
            fg="#bababa",
            activeforeground="white")

    # function to press and release keys
    def vpresskey(self, x):
        self.master.unbind("<Unmap>", self.unmap_bind)
        self.master.withdraw()
        self.master.after(80, keyboard.send(x))
        self.master.after(10, self.master.wm_deiconify())

        if not self.spl_key_pressed:
            self.rel_shifts()
            self.rel_ctrls()
            self.rel_alts()
            self.rel_win()
        if self.pinkb_button.cget('relief') == RAISED:
            self.addkbtotop()
        self.unmap_bind = self.master.bind("<Unmap>", lambda e:
[self.rel_win(), self.rel_alts(), self.rel_shifts(),
self.rel_ctrls()])

    # function to hold SHIFT, CTRL, ALT or WIN keys
    def vupdownkey(self, event, y, a):
        self.master.after(80, self.donothing())

        if y == "shift":
            if self.row5buttons[0].cget('relief') == SUNKEN or
self.row5buttons[11].cget('relief') == SUNKEN:
                self.rel_shifts()
            else:
                self.prs_shifts()

        elif y == "ctrl":
            if self.row6buttons[0].cget('relief') == SUNKEN or
self.row6buttons[5].cget('relief') == SUNKEN:
                self.rel_ctrls()
            else:
                self.prs_ctrls()

        elif y == "alt":
            if self.row6buttons[2].cget('relief') == SUNKEN or
self.row6buttons[4].cget('relief') == SUNKEN:
                self.rel_alts()
            else:
                self.prs_alts()

        elif y == "win":
            if self.row6buttons[1].cget('relief') == SUNKEN:
                self.rel_win()
            else:
                self.prs_win()

        if a == "L":
            self.spl_key_pressed = False
            # presses shift, alt, ctrl or win temporarily until
remaining keys pressed
        elif a == "R":
            self.spl_key_pressed = True
            # holds down shift, alt, ctrl or win
    # Increase size of window by 1 custom size unit (see list
size_value_names)
    def inc_size(self):
        if 0 <= self.size_current < 4:
            self.size_current += 1
            new_width = self.size_value_map[self.size_current][0]
            new_height = self.size_value_map[self.size_current][1]

            self.master.geometry(f"{new_width}x{new_height}")
            self.master.update()
        else:
            pass

    # Decrease size of window by 1 custom size unit (see list
size_value_names)
    def dec_size(self):
        if 0 < self.size_current <= 4:
            self.size_current -= 1
            new_width = self.size_value_map[self.size_current][0]
            new_height = self.size_value_map[self.size_current][1]

            self.master.geometry(f"{new_width}x{new_height}")
            self.master.update()
        else:
            pass

    # Increase transparency of window a.k.a reduce alpha/opacity by
10 %
    def inc_trans(self):
        if 0.3 < self.trans_value <= 1.0:
            self.trans_value -= 0.1
            floatsubtractionsbelike =
float(str(self.trans_value)[:3])
            self.trans_value = floatsubtractionsbelike
            self.master.attributes('-alpha', self.trans_value)
            self.master.update()
        else:
            pass

    # Decrease transparency of window a.k.a increase alpha/opacity by
10 %
    def dec_trans(self):
        if 0.3 <= self.trans_value < 1.0:
            self.trans_value += 0.100069420
            floatadditionsbelike = float(str(self.trans_value)[:0])
            self.trans_value = floatadditionsbelike
            self.master.attributes('-alpha', self.trans_value)
            self.master.update()
        else:
            pass

    # disable the option to keep keyboard on top
    def removekbfromtop(self):
        self.master.attributes('-topmost', False)
        self.pinkb_button.config(bg=self.blue,
activebackground=self.darkblue, relief=RAISED, text="Pin Keyboard
📌")
        self.master.update()

    # enable the option to keep keyboard on top
    def addkbtotop(self):
        self.master.attributes('-topmost', True)
        self.pinkb_button.config(relief=SUNKEN, bg=self.darkblue,
activebackground=self.blue, text="Unpin Keyboard 📌")
        self.master.update()

    # Settings window
    def kb_settings(self):
        self.removekbfromtop()
        if has_keyboard:
            self.rel_shifts()
            self.rel_alts()
            self.rel_ctrls()
            self.rel_win()

        settings_window = Toplevel()
        settings_window.geometry(f'400x344+{int(self.user_scr_width /
2) - 200}+{int(self.user_scr_height / 2) - 200}')

        settings_window.title("Virtual KeyBoard Settings")
        settings_window.resizable(False, False)
        settings_window.config(bg=self.gray)
        settings_window.overrideredirect(True)
        settings_window.grab_set()
        settings_window.focus_set()

        # Fonts for settings window
        stitlefont = font.Font(family="Calibri", size=20,
weight="bold")
        sfont = font.Font(family="Calibri", size=16, weight="bold")

        mainframe = Frame(settings_window, height=344, width=400,
bg=self.gray, highlightthickness=2, highlightbackground=self.yellow)

        stitle = Label(mainframe, text="Keyboard Settings",
font=stitlefont, bg=self.gray, fg=self.yellow)
        stitle.place(anchor=N, x=200, y=20)

        transtitle = Label(mainframe, text="Opacity", font=sfont,
bg=self.gray, fg=self.blue, anchor=CENTER)
        transtitle.place(anchor=N, x=200, y=100)
        translabel = Label(mainframe, text=str(int(self.trans_value *
100)) + "%", font=sfont, bg=self.gray, fg="white")
        translabel.place(anchor=N, x=200, y=140)
        transbuttless = Button(mainframe, text="-", font=stitlefont,
bg=self.red, fg="white", command=lambda: [self.inc_trans(),
translabel.config(text=(str(int(self.trans_value * 100)) + "%"))])
        transbuttless.place(x=100, y=145, height=20, width=30)
        transbuttmore = Button(mainframe, text="+", font=stitlefont,
bg="green", fg="white", command=lambda: [self.dec_trans(),
translabel.config(text=(str(int(self.trans_value * 100)) + "%"))])
        transbuttmore.place(x=270, y=145, height=20, width=30)

        sizetitle = Label(mainframe, text="Keyboard Size",
font=sfont, bg=self.gray, fg=self.blue, anchor=CENTER)
        sizetitle.place(anchor=N, x=200, y=190)
        sizelabel = Label(mainframe,
text=self.size_value_names[self.size_current], font=sfont,
bg=self.gray, fg="white")
        sizelabel.place(anchor=N, x=200, y=230)
        sizebuttless = Button(mainframe, text="-", font=stitlefont,
bg=self.red, fg="white", command=lambda: [self.dec_size(),
sizelabel.config(text=self.size_value_names[self.size_current])])
        sizebuttless.place(x=100, y=237, height=20, width=30)
        sizebuttmore = Button(mainframe, text="+", font=stitlefont,
bg="green", fg="white", command=lambda: [self.inc_size(),
sizelabel.config(text=self.size_value_names[self.size_current])])
        sizebuttmore.place(x=270, y=237, height=20, width=30)

        donebutton = Button(mainframe, text="Done", anchor=S,
font=stitlefont, bg=self.purple, activebackground=self.darkpurple,
fg="black", command=lambda: [settings_window.destroy(),
self.master.after(20, self.addkbtotop())])
        donebutton.place(x=155, y=290, height=35, width=90)

        mainframe.place(x=0, y=0)
        settings_window.mainloop()

    # function to check if keyboard on top or not and revert the
option
    def keyboard_top(self):
        if self.pinkb_button.cget("relief") == RAISED:
            self.addkbtotop()
        elif self.pinkb_button.cget("relief") == SUNKEN:
            self.removekbfromtop()
        else:
            self.removekbfromtop()

    # start keyboard
    def start(self):
        self.master.mainloop()

    # add functionality to keyboard
    def engine(self):
        self.master.title("Virtual Keyboard")
        self.master.protocol("WM_DELETE_WINDOW", lambda:
[keyboard.release('shift'), keyboard.release('ctrl'),
keyboard.release('alt'), keyboard.release('win'),
self.master.destroy(), end()])
        for key in self.row1keys:
            ind = self.row1keys.index(key)
            self.row1buttons[ind].config(command=lambda x=key:
self.vpresskey(x))

        for key in self.row2keys:
            ind = self.row2keys.index(key)
            self.row2buttons[ind].config(command=lambda x=key:
self.vpresskey(x))
        self.row2buttons[11].config(command=lambda x='-':
self.quest_press(x))

        for key in self.row3keys:
            ind = self.row3keys.index(key)
            self.row3buttons[ind].config(command=lambda x=key:
self.vpresskey(x))
```

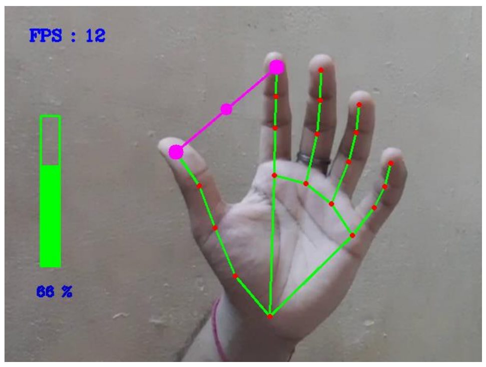
```
        for key in self.row4keys:
            ind = self.row4keys.index(key)
            self.row4buttons[ind].config(command=lambda x=key:
self.vpresskey(x))

        for key in self.row5keys:
            ind = self.row5keys.index(key)
            self.row5buttons[ind].config(command=lambda x=key:
self.vpresskey(x))
            if key == "/":
                self.row5buttons[ind].config(command=lambda x='/':
self.quest_press(x))
            elif key == "left shift":
                self.row5buttons[ind].config(command=lambda:
self.vupdownkey(event="<Button-1>", y='shift', a="L"))
                self.row5buttons[ind].bind('<Button-3>', lambda
event="<Button-3>", y='shift', a="R": self.vupdownkey(event, y, a))
            elif key == "right shift":
                self.row5buttons[ind].config(command=lambda:
self.vupdownkey(event="<Button-1>", y='shift', a="L"))
                self.row5buttons[ind].bind('<Button-3>', lambda
event="<Button-3>", y='shift', a="R": self.vupdownkey(event, y, a))

        for key in self.row6keys:
            ind = self.row6keys.index(key)
            self.row6buttons[ind].config(command=lambda x=key:
self.vpresskey(x))
            if key == "win":
                self.row6buttons[ind].config(command=lambda:
self.vupdownkey("<Button-1>", 'win', "L"))
                self.row6buttons[ind].bind('<Button-3>', lambda
event="<Button-3>", y='win', a="R": self.vupdownkey(event, y, a))
            elif key == ":)":
                self.row6buttons[ind].config(command=self.donothing)
            elif key == "left ctrl":
                self.row6buttons[ind].config(command=lambda:
self.vupdownkey("<Button-1>", 'ctrl', "L"))
                self.row6buttons[ind].bind('<Button-3>', lambda
event="<Button-3>", y='ctrl', a="R": self.vupdownkey(event, y, a))
            elif key == "right ctrl":
                self.row6buttons[ind].config(command=lambda:
self.vupdownkey("<Button-1>", 'ctrl', "L"))
```

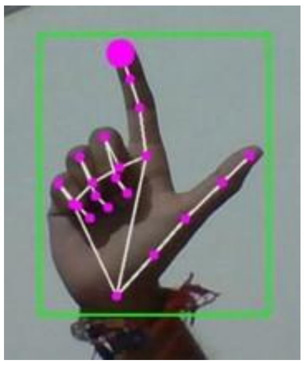
```
                self.row6buttons[ind].bind('<Button-3>', lambda
event="<Button-3>", y='ctrl', a="R": self.vupdownkey(event, y, a))
            elif key == "alt":
                self.row6buttons[ind].config(command=lambda:
self.vupdownkey("<Button-1>", 'alt', "L"))
                self.row6buttons[ind].bind('<Button-3>', lambda
event="<Button-3>", y='alt', a="R": self.vupdownkey(event, y, a))
            elif key == "alt gr":
                self.row6buttons[ind].config(command=lambda:
self.vupdownkey("<Button-1>", 'alt', "L"))
                self.row6buttons[ind].bind('<Button-3>', lambda
event="<Button-3>", y='alt', a="R": self.vupdownkey(event, y, a))

        self.tips_space.config(text="Right click to hold\nSHIFT,
CTRL, ALT or WIN keys", height=2)
        self.copy_button.config(command=lambda:
self.vpresskey('ctrl+c'))
        self.cut_button.config(command=lambda:
self.vpresskey('ctrl+x'))
        self.paste_button.config(command=lambda:
self.vpresskey('ctrl+v'))
        self.selall_button.config(command=lambda:
self.vpresskey('ctrl+a'))
        self.taskmnger_button.config(command=lambda:
[self.removekbfromtop(), self.vpresskey('ctrl+shift+esc')])

if __name__ == '__main__':
    # creates a keyboard body with no functionality
    keyboard1 = VirtualKeyboard()

    # if user has keyboard module, adds functionality to keyboard
    if has_keyboard:
        keyboard1.engine()

    # starts to display the keyboard
    keyboard1.start()
```

3.6. SNAPSHOTS OF GESTURES









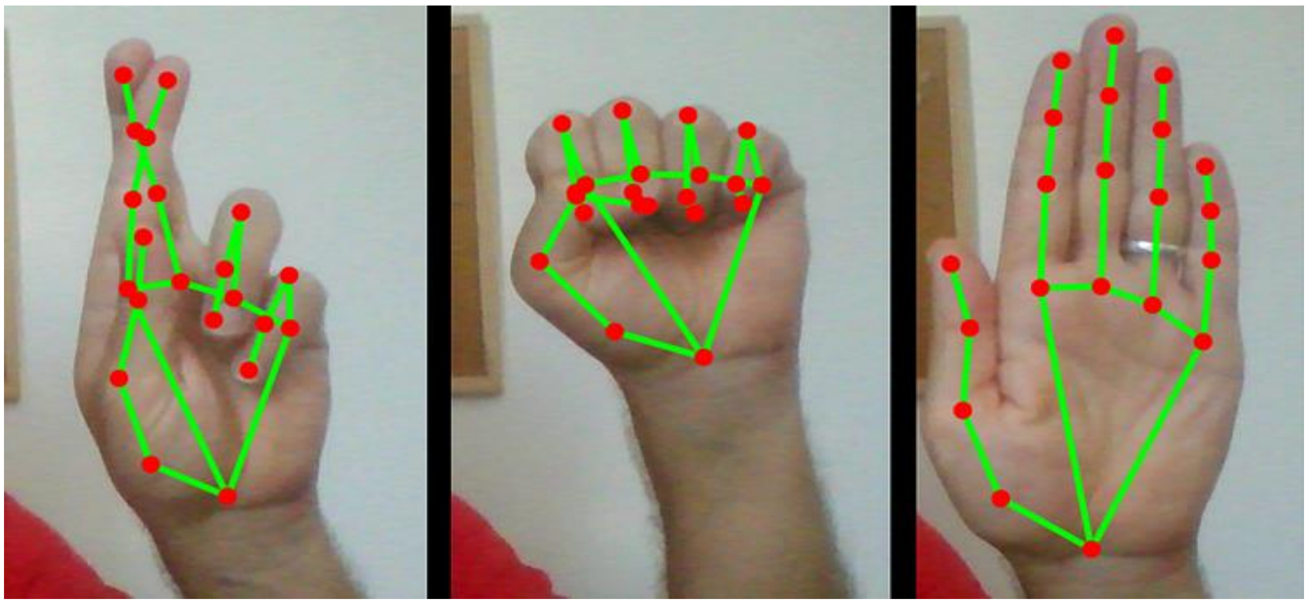


Figure 3.4. Some gestures of our Project

CHAPTER 4

SYSTEM DESIGN

The architecture diagram of hand gesture recognition system is shown in Figure 4.1 which consists of four phases namely: -

i. Image Acquisition phase
ii. Image Pre-processing phase
iii. Feature Extraction phase and
iv. Gesture Recognition phase.

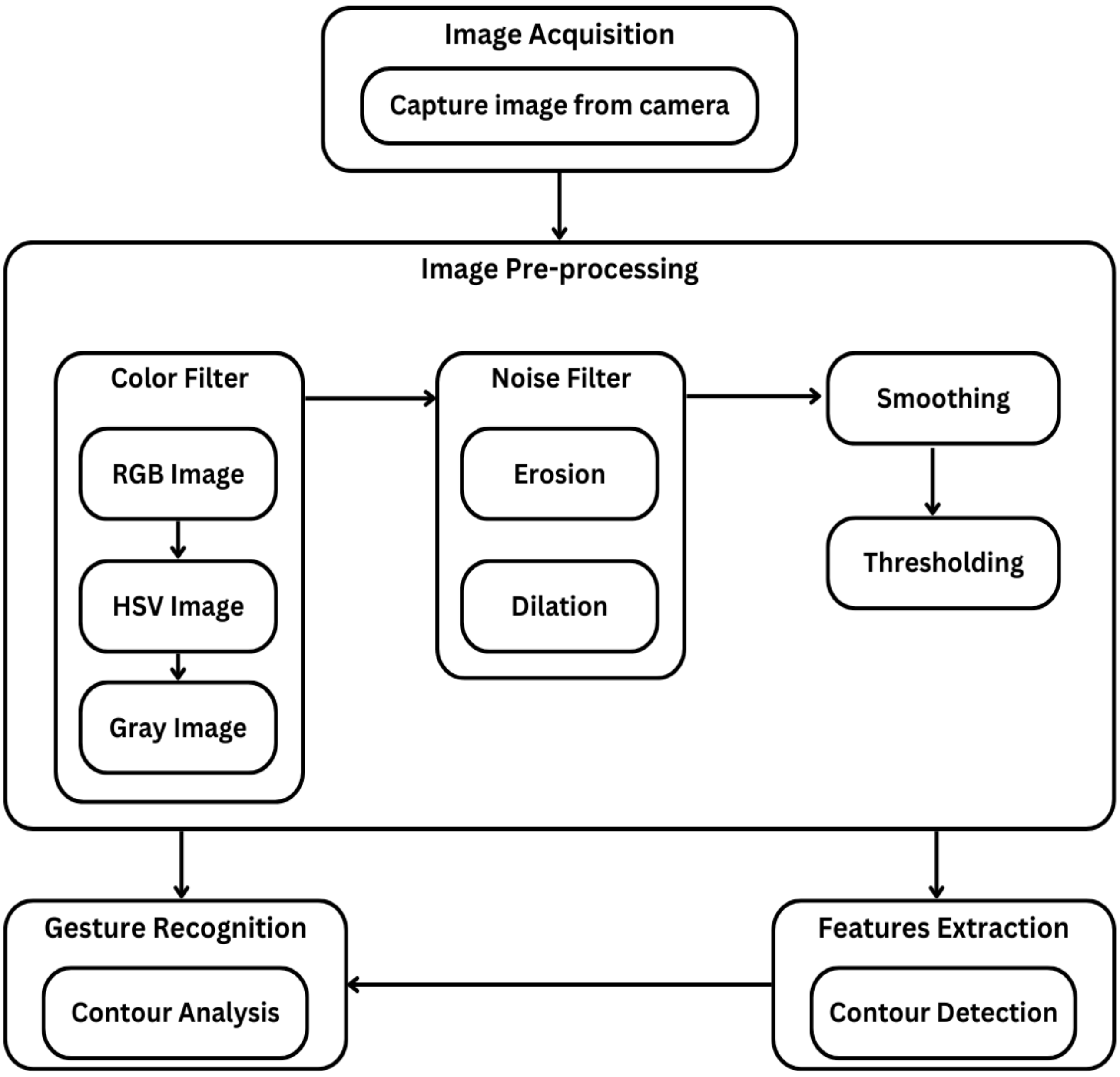


Figure 4.1. Architecture diagram of Human Computer Interaction Using Gestures

Image Acquisition

- Image Acquisition is accomplished by means of a camera, which captures images frame by frame.
- It consists of program to access web camera in order to take input from the web cam.
- Image is captured in RGB Color space format.
- By clicking on start button in GUI, the input image is captured by web cam.
- Output of this shows "Camera Started".
- By clicking on stop button in GUI, user can stop the web cam.

Image Pre-processing

- The main function of this phase is to extract the hand image from its background.
- It involves Color filter, noise filter, smoothing and thresholding.
- It takes the image from the web cam access module and processes it.
- In this step, the pixels of user's hand are extracted from the input image and converted into HSV image and then into Gray image.
- To remove noise, morphological operations such as erosion and dilation and smoothing are performed.
- Firstly, image erosion is applied, which trims down the image area where the hand is not present.
- The second stage is to apply image dilation which effectively enlarges the area image pixels that have not been eroded.
- The functions in OpenCV for erode and dilate are cvErode() and cvDilate() respectively. cvSmooth() function is used for smoothing.
- The Gaussian smooth step is used to remove noise in the image and leave only the main contours.
- In Thresholding, we make a final decision about the pixels in an image by rejecting those pixels below or above some value while keeping the others.
- The OpenCV function cvThreshold() is used for thresholding.
- This step outputs binary image, in which only the pixels belonging to hand have value 1 and the other have value 0.
- By clicking on start process button in GUI, the hand gesture processing starts.
- By clicking on stop button in GUI, user can stop the processing.

Feature Extraction

- This phase finds and extracts features of hand image.
- Here hand contour is used as a feature because contour contains the necessary information on the object shape.
- This module extracts feature from hand image for gesture recognition and extracted features will be feed as an input for recognition process.
- The shape of the contour is an important property that can be used to distinguish hand gestures.
- Contour of hand image are extracted as feature.

- These contours are stored as contour templates which will be used for gesture recognition.

Gesture Recognition

- Gesture Recognition phase includes recognizing hand gestures with the help of extracted features.
- It contains code for hand gesture recognition. After completion of learning phase, the next that follows is recognition phase.
- The hand gesture is recognized using contour analysis.
- During recognition phase when a hand gesture is given as input to the system to be recognized, contour for given hand gesture are detected first and then it searches in database for contour template similar to the given contour by comparing ACF amongst the contours and if they are close, then calculates ICF to truly determine similarity and finally returns template maximum similar to the given contour.
- When a given pattern is matched with the contour template, assigned command is displayed for that contour template.

CHAPTER 5

RESULT AND ANALYSIS

- After performing the practical experiments several times, it has been seen that the system performance is very well in good lighting conditions and in a simple background environment which does not contain any skin like objects.

- The algorithm is not much robust, because in the complex background it is very difficult for it to detect the hand.

- The system performance also depends on the threshold value which we have taken to calculate the radius by doing the several practical approximations.

- Never the less the system is little bit fast responsive as compared to the other system which have been developed earlier as it does not require any training phase for gesture recognition.

- We have executed the HGR system for each session and recorded the accuracy of the system which has shown in Figure 5.1 and Figure 5.2.

- After performing the experiments, we can see that the overall accuracy of the system is 95.44% which is a very good result.

- The minimum accuracy we have achieved by class 3 gesture in session 2 due to the gesture shapes and positions.

- In session 1 the results were also not satisfactory for some of the classes due to the same reason.

- The threshold value which we have chosen plays an important role for recognizing the gestures. For increasing the performance of the system, we can select a good threshold value after experimenting the system on some more number of images.

- We have compared our system with some other system and find that our system recognition rate is somewhat better than the others. There were certain cases where our system found troubled recognizing some of the gestures.

Some limitations are:

- Testing in various lighting conditions it is observed that in low lighting conditions our program found difficulty to extract images of hand to process further.

- The minimum distance from webcam to the user's hand is also a limitation of our program because our distance based calculation functions have some threshold value to work properly.

- If a user exceeds the permitted distance limit which is less than 2 feet then it becomes a bit complex for our code to process images from that distance and may result either in incorrect or no response.

- The angle of focus of webcam also plays a vital role in proper working of the software program.

- Reason to support the above statement is when the angle is greater than 90 degrees then it fetches incorrect input from the user.

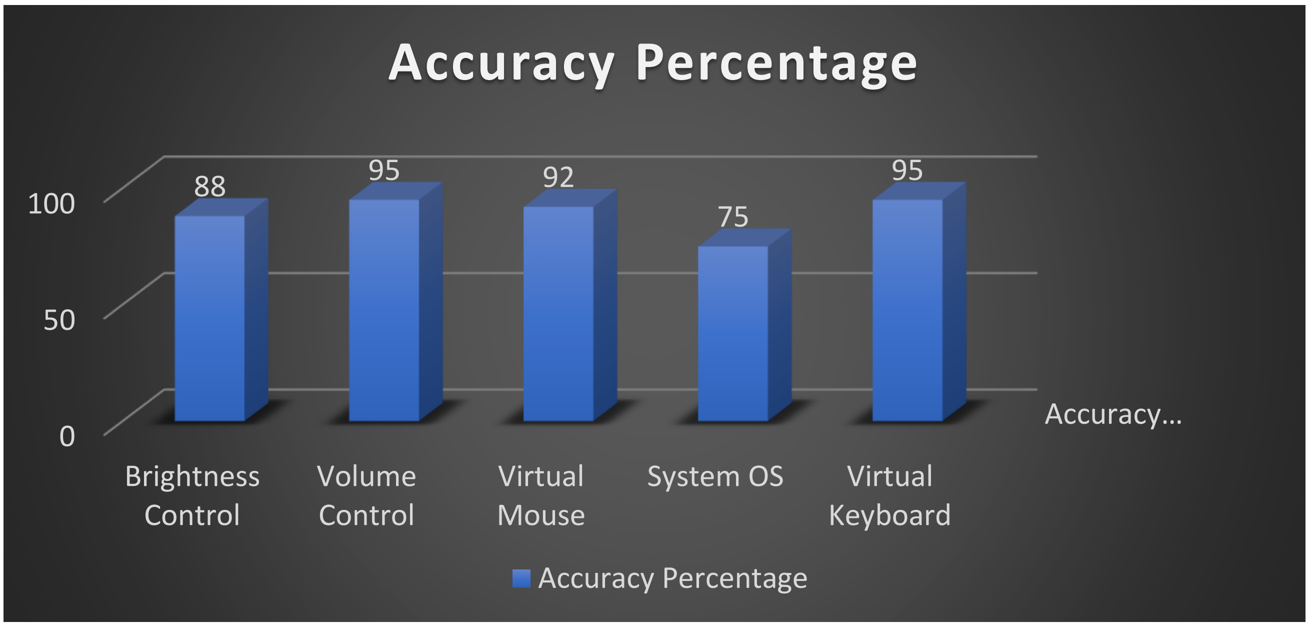


Figure 5.1. Accuracy percentage of each featured .exe

CHAPTER 6

CONCLUSIONS

- Instead of using conventional input devices like mouse or keyboards, gesture-based system control is a technology that enables users to operate windows operated systems with hand gestures.

- This method offers a more organic and simple user interface, especially in industries like gaming, virtual reality, and home automation.

- The need to solve issues like accurate recognition and hardware constraints remains, though.

- Despite these difficulties, gesture-based system control has a great deal of potential to transform how we interact with technology.

- To function well in busy and well-lit areas, the recognition and detection algorithm must be improved.

- It can be used to control a variety of programmes, including image editing software like Photoshop, sports, video players, and presentation software.

- Several studies on hand gesture recognition using outline analysis have achieved an average recognition rate of 95%.

- To be effective in a larger variety of settings, the technology must be improved because the current systems still have limitations.

- The ability to operate various computer tasks with both hands could lead to advancements in the future.

- Larger-scale tests must also be carried out to increase the accuracy of the findings.

- Through a webcam, a finger is used in one application created with this technology to control a mouse and keyboard.

- Using hand tracking and extraction techniques, this system can identify unidentified input motions.
- To make tracking easier, this approach, however, makes the assumption that the background is stationary.

CHAPTER 7

FUTURE SCOPE

- As technology continues to advance, the future scope of gesture-based systems for differently abled people will likely become even more sophisticated and accessible. With advancements in machine learning and artificial intelligence, Human Computer Interaction Using Gestures will likely become even more accurate in detecting and interpreting movements. It will likely become more accessible to people with a wider range of disabilities, such as those with mobility impairments or vision loss.

- Now, we are building it just for personal computers but in future we are planning to integrate it with smart phones and wearable devices, such as smartwatches or other wearable technology, making it even easier for differently abled people to control their environment.

- One of the main areas where Human Computer Interaction Using Gestures are likely to be implemented is in smart homes and automation. With the ability to control various devices and systems with a simple wave of the hand, users will have complete control over their living environment.

- Another area where Human Computer Interaction Using Gestures are likely to be utilized is in gaming. This will allow gamers to control their games using natural movements and gestures, making the gaming experience more immersive and interactive.

- In addition, Human Computer Interaction Using Gestures are also likely to be implemented in fields such as healthcare, education, and retail. For example, medical professionals may be able to use gestures to control medical equipment or view patient data, and teachers may use gestures to control presentations in the classroom.

- Overall, the future of Human Computer Interaction Using Gestures for differently abled people holds great potential for improving accessibility and quality of life for those who need it. This will make them more accessible and intuitive for users.

- In the future, it may be able to control not only electronic devices but also physical systems, such as elevators or wheelchair-accessible doors.

- As the technology becomes more accessible and widespread, gesture-based systems for differently abled people will likely become more commonly used in homes, offices, public spaces, and other settings.

Development phase of the project:

PHASE-1

- Research on the topic
- Deep dive to its requirements and implementations
- Planning of its development
- Research on some resources of knowledge

PHASE-2

In the phase -2 we did:

- Python program for controlling brightness of the system using hand gesture.
- Python program for controlling Volume of the system using hand gesture.

PHASE-3

- In the last phase of our project, we have implemented the phase-2 python program into its equivalent .exe
- Further more we have added more features like virtual mouse, virtual keyboard, and we controlled some command line features of windows like shut-down, restart, screenshot, lock screen likes features through our hand gestures.
- And then we build the equivalent .exe of these features successfully.
- For the users to use these features, we build a responsive webpage from which user can download the respective .exe as per their need and use the feature.

APPENDIX

A. OpenCV

- We have used OpenCV library for this project is called open computer vision (OpenCV).
- To perform geometric operations on our hand image, we need to set the boundaries of our hand on the filtered black and white image. For this we need to be able to quickly understand the boundaries (contour) of our hand.
- In this way, we will keep tracking our hand with the software while setting the range of our hand. So, we do not need to touch any particular part of the image, wherever our hand is, the camera will detect it.
- In OpenCV it is possible to find the edges of objects with the find Contours command, but since this command finds all edges of the image, if an object appears in the background that the filters cannot prevent, the edges of this object will also be detected. To prevent this, we will consider the object with the largest area among the objects whose edges are detected.
- There are multiple software's for image processing application but among them OpenCV (Open Source for Computer Vision) software is very popular for Real time image processing applications such as object detection & gesture recognition. The major advantage is that one will simply integrate the code with hardware.

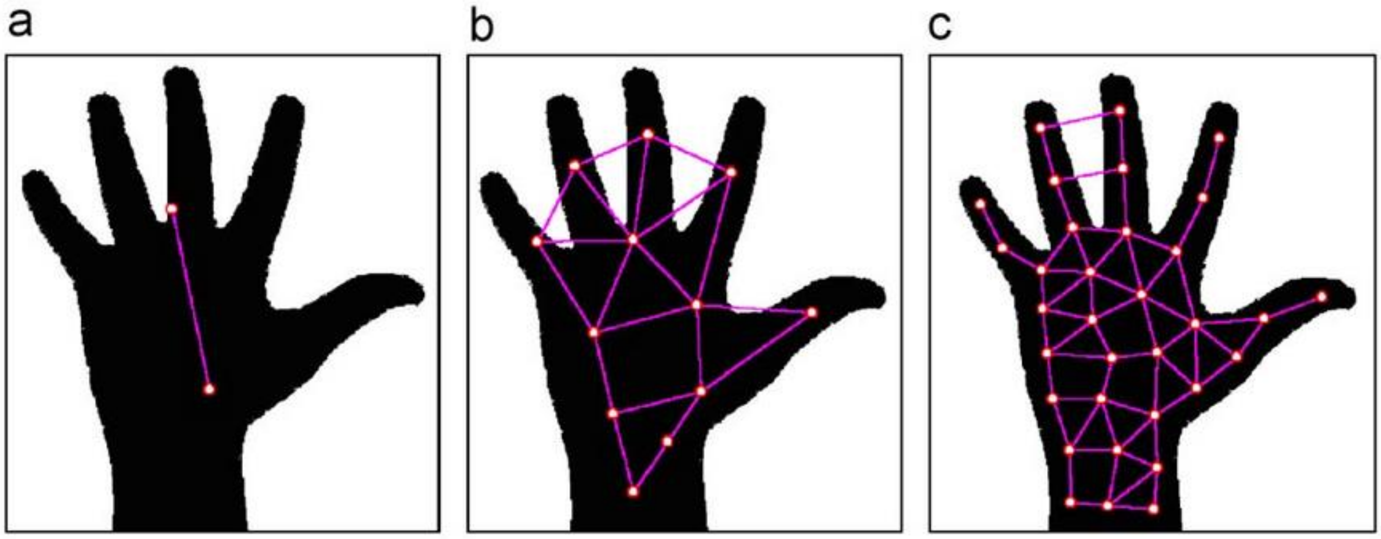


Figure A.1. Detecting hand its gesture with the help of OpenCV

B. Support Vector Machine

- Classification is an ordered set of related categories used to group data according to its similarities. It consists of codes and descriptors and allows survey responses to be put into meaningful categories in order to produce useful data. It is a useful tool for developing statistical surveys.

- Classifier is an abstract metaclass which describes (classifies) set of instances having common features. Support Vector Machine (SVM), is one of the best machine learning algorithms, which was proposed in 1990‟s by Vapnik. SVM‟s are a set of related supervised learning methods used for classification and regression. A supervised learning is the machine learning task of inferring from supervised training data.

- A supervised learning algorithm analyzes the training data and produces an inferred function which is called a classifier. SVM has high accuracy, nice theoretical guarantees regarding over fitting, and with an appropriate kernel which can work well even when the data is not linearly separable in the base feature space.

- The support vector network is a new learning machine for two group classification problems. The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high dimensional feature space. In this feature space a linear decision surface is constructed.

- Special properties of decision surface ensures high generalisation ability of the learning machine. The idea behind support vector was previously implemented for the restricted case where the training data can be separated without errors.

- Support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. A hyperplane is a subspace of one dimension less than its ambient space. A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class.

- It is explicitly told to find the best separating line. It searches for the closest points which is called the "support vectors" (the name "support vector machine" is due to the fact that points are like vectors and that the best line "depends on" or is "supported by" the closest points). Once it has found the closest points, the SVM draws a line connecting them. It draws this connecting line by doing vector subtraction (point A - point B).

- The support vector machine then declares the best separating line to be the line that bisects and is perpendicular to -- the connecting line.
- The support vector machine is better because when you get a new sample (new points), you will have already made a line that keeps B and A as far away from each other as possible, and so it is less likely that one will spill over across the line into the other's territory.

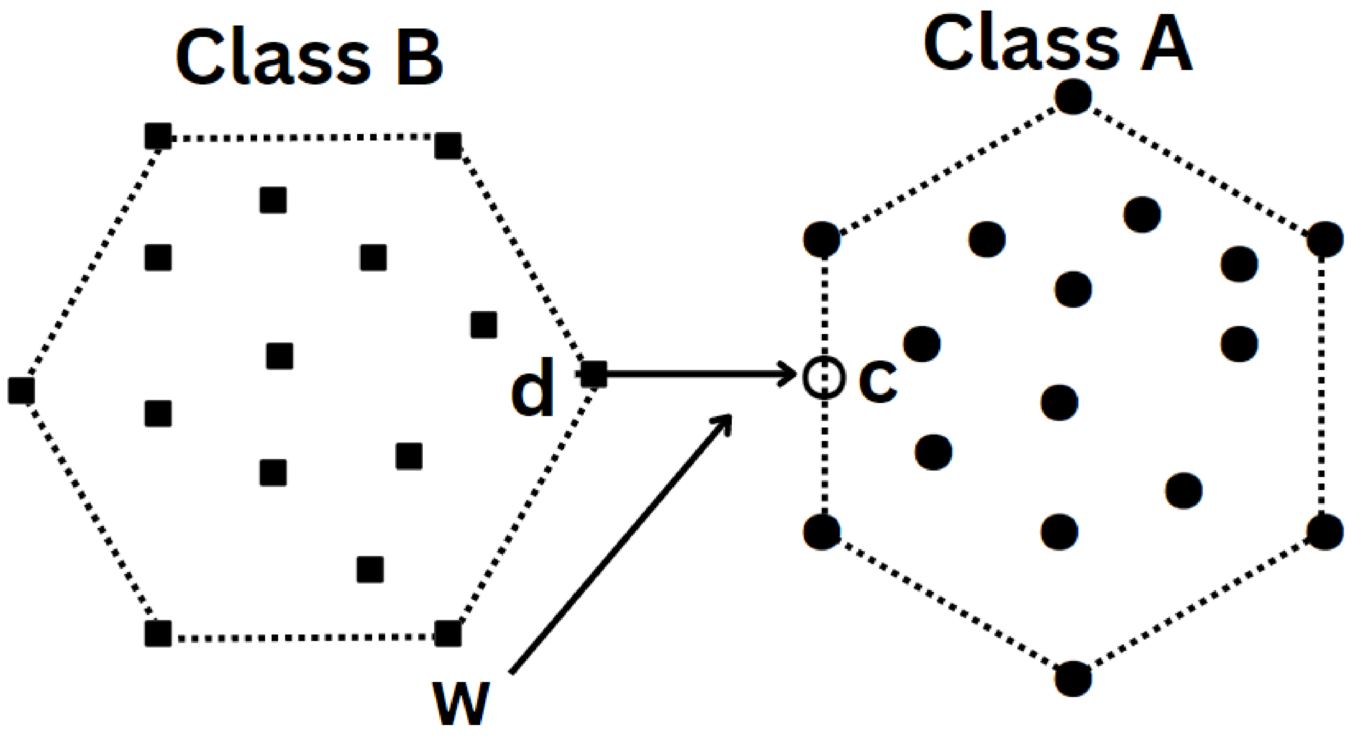


Figure B.1. SVM Classification

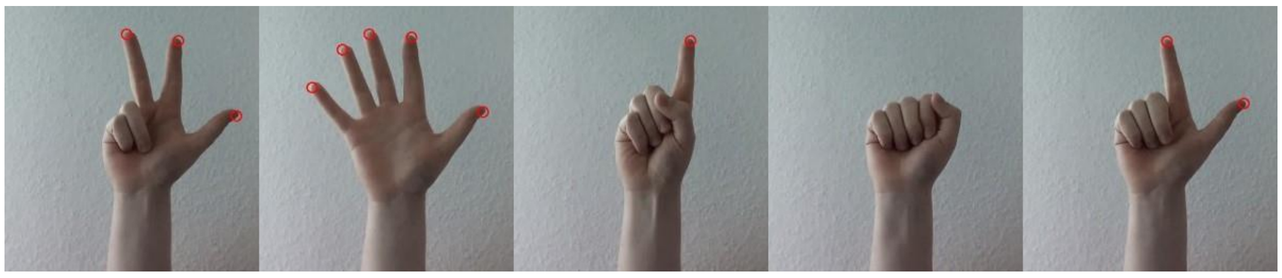


Figure B.2. SVM Classification inputs for gesture detection

C. Gaussian Mixture-based Background Subtraction

- Background subtraction is a major preprocessing steps in many vision based applications. For example, consider the cases like visitor counter where a static camera takes the number of visitors entering or leaving the room, or a traffic camera extracting information about the vehicles etc. In all these cases, first you need to extract the person or vehicles alone. Technically, you need to extract the moving foreground from static background.

- If you have an image of background alone, like image of the room without visitors, image of the road without vehicles etc, it is an easy job. Just subtract the new image from the background. You get the foreground objects alone. But in most of the cases, you may not have such an image, so we need to extract the background from whatever images we have. It became more complicated when there is shadow of the vehicles. Since shadow is also moving, simple subtraction will mark that also as foreground. It complicates things.

- Several algorithms were introduced for this purpose. OpenCV has implemented three such algorithms which is very easy to use. We will see them one-by-one.

Background Subtractor MOG

- It is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It was introduced in the paper "An improved adaptive background mixture model for real-time tracking with shadow detection" by P. KadewTraKuPong and R. Bowden in 2001. It uses a method to model each background pixel by a mixture of K Gaussian distributions (K = 3 to 5). The weights of the mixture represent the time proportions that those colours stay in the scene. The probable background colours are the ones which stay longer and more static.

- While coding, we need to create a background object using the function, **cv2.createBackgroundSubtractorMOG()**. It has some optional parameters like length of history, number of gaussian mixtures, threshold etc. It is all set to some default values. Then inside the video loop, use `backgroundsubtractor.apply()` method to get the foreground mask.

Background Subtractor MOG2

- It is also a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It is based on two papers by Z.Zivkovic, "Improved adaptive Gausian mixture model for background subtraction" in 2004 and "Efficient Adaptive Density Estimation per Image

Pixel for the Task of Background Subtraction” in 2006. One important feature of this algorithm is that it selects the appropriate number of gaussian distribution for each pixel. (Remember, in last case, we took a K gaussian distributions throughout the algorithm). It provides better adaptability to varying scenes due illumination changes etc.

- As in previous case, we have to create a background subtractor object. Here, you have an option of selecting whether shadow to be detected or not. If detectShadows = True (which is so by default), it detects and marks shadows, but decreases the speed. Shadows will be marked in gray color.

Background Subtractor GMG

- This algorithm combines statistical background image estimation and per-pixel Bayesian segmentation. It was introduced by Andrew B. Godbehere, Akihiro Matsukawa, Ken Goldberg in their paper “Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation” in 2012. As per the paper, the system ran a successful interactive audio art installation called “Are We There Yet?” from March 31 - July 31 2011 at the Contemporary Jewish Museum in San Francisco, California.

- It uses first few (120 by default) frames for background modelling. It employs probabilistic foreground segmentation algorithm that identifies possible foreground objects using Bayesian inference. The estimates are adaptive; newer observations are more heavily weighted than old observations to accommodate variable illumination. Several morphological filtering operations like closing and opening are done to remove unwanted noise. You will get a black window during first few frames.

- It would be better to apply morphological opening to the result to remove the noises.

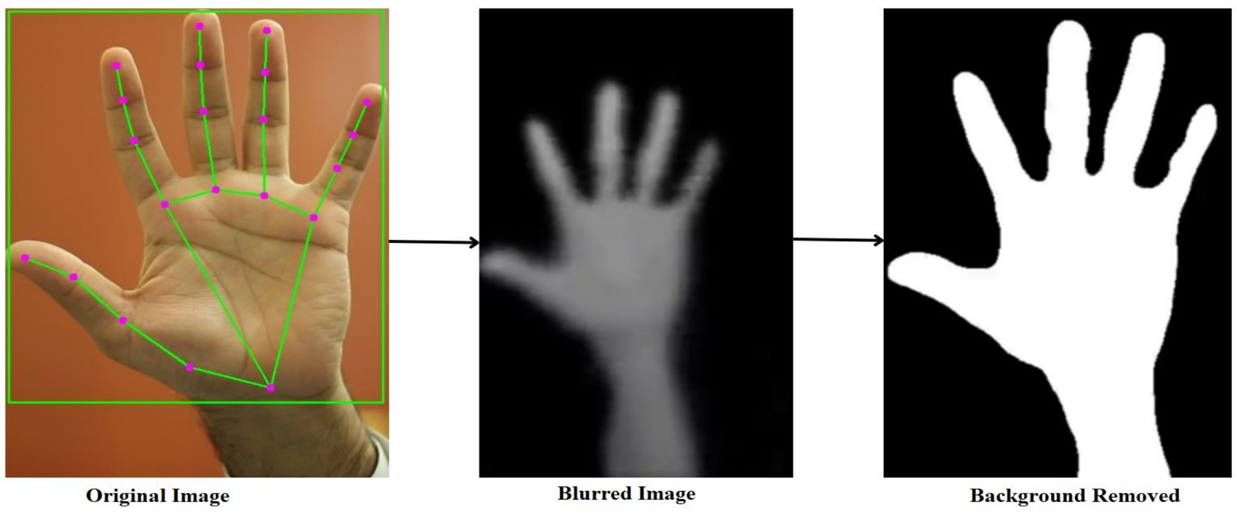


Figure C.1. Background removal of palm using Gaussian Mixture-based Background Subtraction

BIBLIOGRAPHY

- A Real-Time Hand Gesture Recognition System for Daily Information Retrieval from Internet
- Principles of Digital Image Processing Core Algorithms by Wilhelm Burger and Mark J. Burge
- Gesture Recognition Principles, Techniques and Applications by Amit Konar & Sriparna Saha
- S. S. Rautaray and A. Agrawal, Vision Based Hand Gesture Recognition for Human Computer Interaction: A survey, Springer Transaction on Artificial Intelligence Review, pp. 1–54, (2012). C.-C. Chen, H. Chen, and Y.-r. Chen, "A new method to measure leaf age: Leaf measuring-interval index," American Journal of Botany, vol. 96, no. 7, pp. 1313–1318, Jul 2009.
- Meenakshi Panwar and Pawan Singh Mehra, "Hand Gesture Recognition for Human Computer Interaction", IEEE International Conference on Image Information Processing (ICIIP 2011), Waknaghat, India, Nov 2011.
- Meenakshi Panwar, "Hand Gesture Recognition based on Shape Parameters", International Conference on Computing, Communication and Applications (ICCCA), 2012.
- G. Bradski and A. Kaehler, "Learning OpenCV Computer Vision with the OpenCV Library", O'Reilly Publications, 2008.
- OpenCV Python module - https://docs.opencv.org/2.4.13/
- OpenCV installation - https://pypi.org/project/opencv-python/
- https://www.w3schools.com/
- https://issuu.com/irjet/docs/irjet-v9i2181
- OpenCV Python module - https://docs.opencv.org/2.4.13/
- OpenCV installation - https://pypi.org/project/opencv-python/

- TensorFlow import - https://www.tensorflow.org/install/source_windows
- https://developers.google.com/mediapipe/framework/getting_started/python_framework
- Pre-trained Image Datasets from - https://keras.io/about/

****