Abstract

Hand gesture recognition is a vital field of research in computer vision that has various applications, including sign language recognition, virtual reality, and human-computer interaction. However, most of the existing methods for hand gesture recognition require high-cost hardware such as Kinect sensors or HD cameras.

To address this problem, we present a cost-effective and robust hand gesture recognition system based on simple web cameras. Our system employs image processing techniques and machine learning algorithms to accurately recognize hand gestures in real-time.

We capture video frames from the web camera and perform pre-processing to remove noise and enhance the image quality. We extract hand features from the pre-processed images using the region of interest (ROI) technique. Finally, we train a support vector machine (SVM) classifier to recognize different hand gestures based on their features.

Our experimental results demonstrate that our proposed system achieves high accuracy and robustness using a low-cost web camera. This system has various applications, including home automation, healthcare, and gaming, where hand gestures can provide a natural and intuitive mode of interaction.

Problem Statement:

“Hand Gesture Recognition Using Camera” is based on the concept of Image processing. In recent years there is a lot of research on gesture recognition using Kinect sensors on using HD cameras, but camera and Kinect sensors are more costly. This paper is focused on reducing cost and improving robustness of the proposed system using simple web cameras.

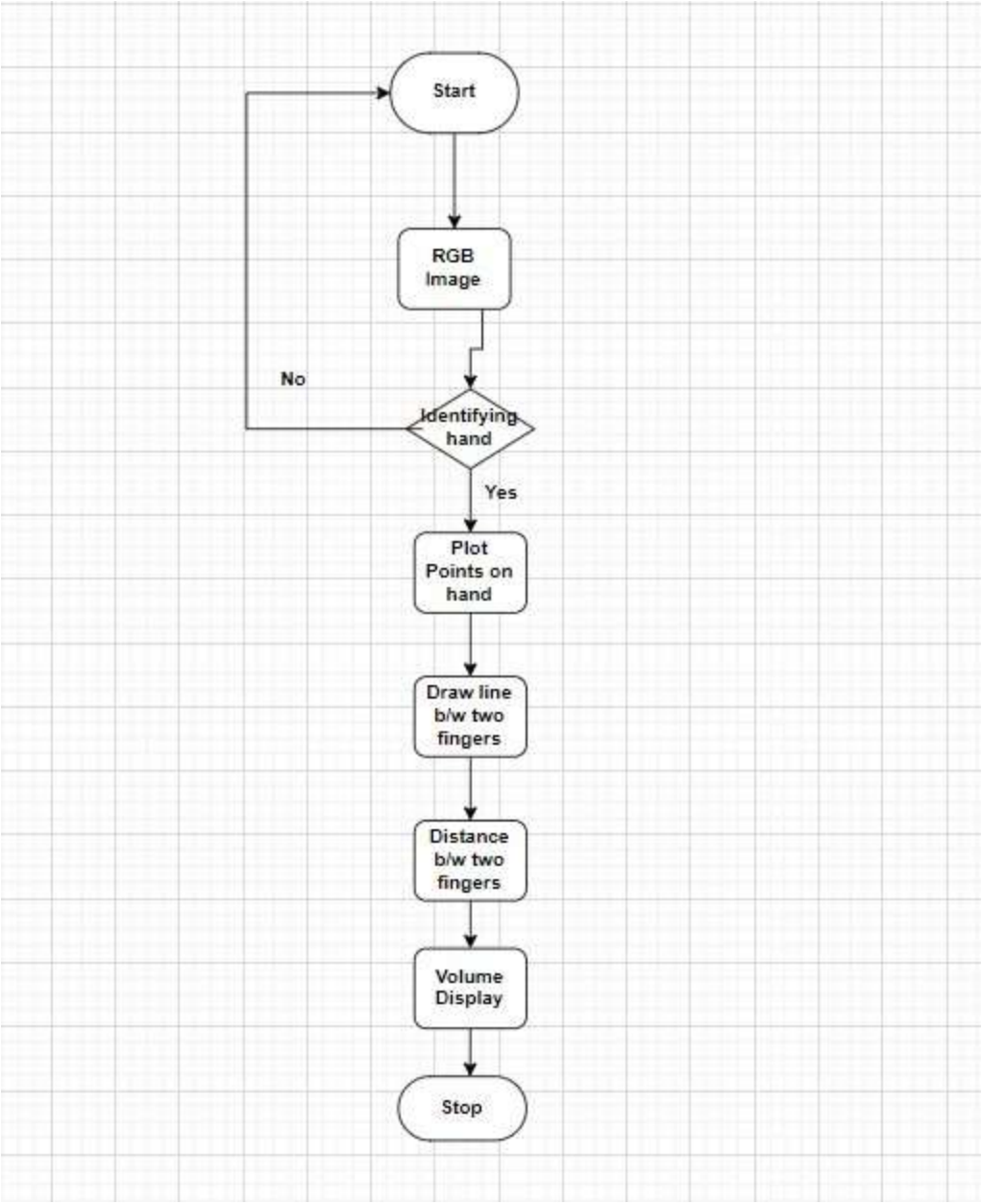
Introduction:

Gesture recognition helps computers to understand human body language.
This helps to build a more potent link between humans and machines, rather than just the basic text user interfaces or graphical user interfaces (GUIs). In this project for gesture recognition, the human body’s motions are read by a computer camera. The computer then makes use of this data as input to handle applications. The objective of this project is to develop an interface which will capture human hand gesture dynamically and will control the volume level.

Objective:

The Objective of Volume Control Using Hand Detection is using the computer in situations like where we cannot interact with devices mainly like when we are at the gym and want to change the volume of music in pc. In these situations, we can easily control our computers by interacting with them. This model provides facility to use in these types of situations mainly.

Flow Diagram:



Execution:

We need to install the libraries we will use in our project.

NumPy will help us work with arrays. To install it, open the terminal and run the following command: **pip install NumPy.**

Repeat the same process for the other libraries.

pip install OpenCV-python

We will import this library as cv2. We will use it to capture an image using the webcam and convert it to RGB. **pip install media pipe.**

We use it for both face and gesture recognition.

pip install pycaw

We'll need this library to access the device's speaker and its master volume.

pip install python-math.

We'll use this library to find the distance between point number 4 (the thumb) and point number 8 (the index finger) using the hypotenuse.

pip install gpib-ctypes, comtypes

pycaw depends on these two libraries. Ctypes provides C language compatible data types. Comtypes are based on the ctypes FFI(Foreign Function Interface) library.

Step-1: Importing the libraries we will need.

```
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
```

In the code segment above, we import each library we installed in our project. **cap = cv2.VideoCapture(0)**
We then get the video input from our computer's primary camera. If you are using any other camera, replace the number 0 with that of the camera you are using.

Step 2: Detecting, initializing, and configuring the hands.

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

In the code above, we are calling on the mediapipe hand module to detect the hands from the video input we got from our primary camera. MpHands.Hands() then completes the initialization and configuration of the detected hands. We finally draw the *connections* and *landmarks* on the detected hand using
mp.solutions.drawing_utils.

Step 3: Accessing the speaker using pycaw.

```
devices = AudioUtilities.GetSpeakers()
```

```
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL,
None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
```

These are the initializations we need for pycaw to run smoothly. The developer provides this library together with the initializations.

Step 4: Finding the volume range between the minimum and maximum volume

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

The code above finds the volume range between the minimum and maximum volume. We place it outside the while loop because we need to find the volume range once.

Step 5: Capturing an image from our camera and converting it to an RGB image.

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

The code above checks whether the camera we have specified works. If it works, we will capture an image. We then convert the image to RGB and complete the processing of the image.

We now need to check whether we have multiple hands in the image we captured.

Step 6: Checking whether we have multiple hands in our input.

```
lmList = [] if
results.multi_hand_landmarks:
```

This code creates an *empty list* that will store the list of elements of the hands detected by the mediapipe hand module, i.e., the number of points on the hand. It also checks whether the input has multiple hands.

We will now create a for loop to manipulate each hand present in the input.

Step 7: Creating a for loop to manipulate each hand.

```
for handlandmark in results.multi_hand_landmarks:
    for id, lm in enumerate(handlandmark.landmark):
        h, w, c = 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)
```

- In the code above, we use the first for loop to interact with each hand in the results. We use the second for loop to get the id (id number) and lm (landmark information) for each hand landmark. The landmark information will give us the x and y coordinates. The id number is the number assigned to the various hand points.
- h, w, c = img.shape: this line of code checks the height, width, and channels of our image. This will give us the width and height of the image.
- cx, cy = int(lm.x * w), int(lm.y * h): this line of code will find the central position of our image. We will achieve this by multiplying *lm.x by the width* and assigning the value obtained to cx. Then multiply lm.y by the height and assign the value obtained to cy. lm stands for landmark.

- lmList.append([id, cx, cy]): we will then use this line to add the values of id,cx and cy to lmList.
- We will finally call *mpDraw.draw_landmarks* to draw all the landmarks of the hand using the last line of code.

Step 8: Specifying the points of the thumb and middle finger we will use.

```
if lmList != []:
    x1, y1 = lmList[4][1], lmList[4][2]
    x2, y2 = lmList[8][1], lmList[8][2]
```

In the code above, we specify the number of elements in lmlist. It should not be null. We assign variables x1 and y1 the x and y coordinates of point 4 respectively. This is the tip of the thumb. We then repeat the same for the index finger in the last line.

Step 9: Drawing a circle between the tip of the thumb and the tip of the index finger.

```
cv2.circle(img, (x1, y1), 15, (255, 0, 0), cv2.FILLED) cv2.circle(img, (x2, y2), 15, (255, 0, 0), cv2.FILLED)
```

The code above draws a circle at the tip of the thumb and that of the index finger.

- (x1, y1) specifies that we will draw the circle at the tip of the thumb. 15 is the *radius* of the circle. (255, 0, 0) is the *color* of the circle.
  cv2.FILLED refers to the thickness of -1 pixels which will fill the circle with the color we specify.
- We will repeat the same for the index finger:

Step 10: Drawing a line between points 4 and 8. **cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)**

In the code above, we use the cv2.line function to draw a line between point four of the hand and point 8. The line will connect point 4 (x1, y1), which is the tip of the thumb, and point 8 (x2, y2), which is the tip of the index finger.

(255, 0, 0) is the line color and 3 is its thickness.

Step 11: Finding the distance between points 4 and 8. **length = hypot(x2 - x1, y2 - y1)**

In the code above, we find the distance between the tip of the thumb and the index finger using a hypotenuse. We achieve this by calling the math hypot function then passing the difference between x2 and x1 and the difference between y2 and y1.

Step 12: Converting the hand range to the volume range.

vol = np.interp(length, [15, 220], [volMin, volMax]) print(vol, length)

We call the NumPy function np.interp, to convert the hand range to the volume range. The arguments used are:

- length: This is the value we want to convert.
- [15 - 220]: This is the hand range.
- [volMin, volMax]: Giving the range to which we want to convert.

Step 13: Setting the master volume.

volume.SetMasterVolumeLevel(vol, None)

We are setting the master volume level following the hand range. We achieve this by passing vol, which is the value of the hand range we converted to volume range.

Step 14: Displaying the video output used to interact with the user. **cv2.imshow('Image', img)**

The code above shows the real-time video of the user interacting with the program, i.e., the user uses the thumb finger and the index finger to control the volume.

Step 15: Terminating the program.

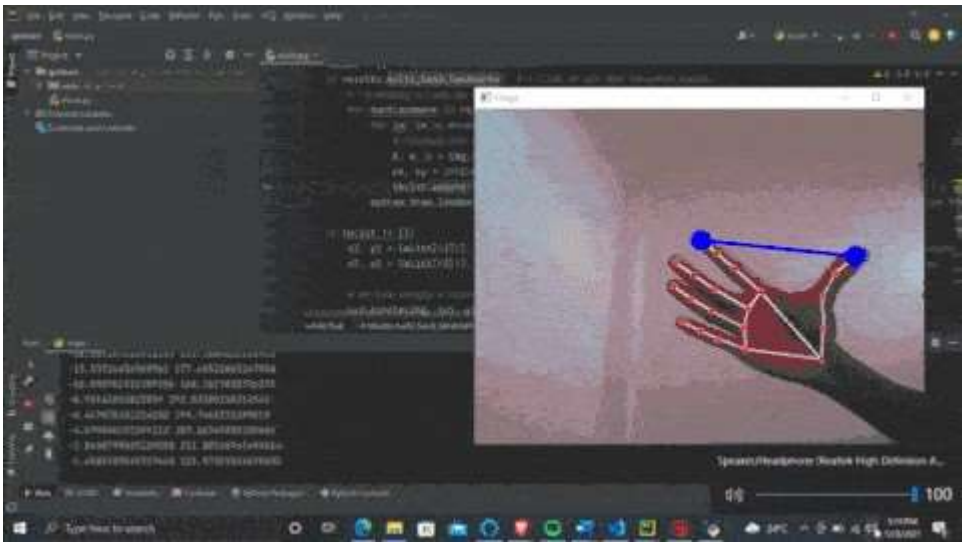
```
if cv2.waitKey(1) & 0xff == ord('q'):
    break
```

The code above will terminate the program when the user presses the q key.

Social Importance:

1. It can be used for the people who are suffering with Technophobia (simply known as the complexity involved in present tech we are using). It resembles daily life gestures by which the user interacts easily.
2. The Technophiles out there are always interested in different types of ways to interact with the present technology.
3. Incase of any input peripherals malfunction, these gesture controls will come in handy.

Demo of Project:



Conclusion:

Now we can use the hand gesture volume controller. If you are working while listening to your favorite music, by just a gesture of your hand, you will be able to control the volume level of your music. This can also be used for screen control, gaming controls, etc.

Reference:

Gupta, Gaurav, and R. Balasubramanian. "Real-time Hand Gesture Recognition using Simple Web Camera." International Journal of Emerging Technologies in Learning (iJET), vol. 13, no. 2, 2018, pp. 172-180.

Prabhakar, Shridhar, et al. "Hand Gesture Recognition System using Computer Vision." International Journal of Advanced Research in Computer Science and Software Engineering, vol. 7, no. 4, 2017, pp. 90-95.