

DISRUPTIVE TECHNOLOGIES-1

[23ECH-102]

PROJECT REPORT

ON

**CONTROLLING PERSONAL DEVICES
USING AIR GESTURES**

Submitted by

Students of 23BCG+BIT-117 Group A

PRIYANSHU KASHYAP (23BCG10009)
SIDDHARTH SONAWANE (23BCG10016)

Academic Unit-1

Department of Computer Science Engineering

University Institute of Engineering

Chandigarh University, Gharuan, Mohali, Punjab, India-140413

Introduction:

Gestures may be investigated in 3 principle paradigms: gesture “perception” (i.e., processes involved with perceiving information conveyed from gestures), “interpretation” of gesture content (i.e., assigning appropriate meaning to gestures), or gesture “performance” (i.e., affectively conveying gestures with coordinated hand and finger movements that accompany sematic material in speech).

Air gestures (non-touch gestures) are a new manner of interaction between a person and a device. They allow a user to conveniently interact with devices without holding or touching them. Air gestures apply to devices such as mobile phones, tablets, automobile head units, smart TVs, speakers, and AR/VR devices. The sensor can recognize your gestures best at a distance of 7cm when you perform them at normal speed. Instead of typing with keys or tapping on a touch screen, a motion sensor perceives and interprets movements as the primary source of data input. This is what happens between the time a gesture is made and the computer reacts. A camera feeds image data into a sensing device that is connected to a computer.

Air Gesture technology can also be used in settings where it's essential to keep people's hands clean, for example in healthcare and manufacturing. The ability to control electronic devices without touching them means people performing work that demands cleanliness can get the information they need without getting their hands dirty. Air Gesture will likely be featured in the near future on electronic devices that are used by the general public thus enabling a more pleasant user experience. We believe that Air Gesture is also ideal for situations where people are hesitant to touch devices with their hands because they have become dirty in the course of daily life.

Problem Statements:

We are going to achieve our goal with the help of the following parameters.

1. **Natural Human-Computer Interaction:** Traditional input devices like keyboards and mice may not provide an intuitive or natural interaction with computers.
2. **Accessibility:** Some individuals may have physical disabilities that limit their ability to use conventional input devices.
3. **Virtual and Augmented Reality:** Interacting with virtual or augmented environments can be challenging with traditional input methods.
4. **Gesture-Based Gaming:** Traditional controllers may not provide an immersive gaming experience.
5. **Hands-Free Control:** In certain scenarios, users may need hands-free control over devices or systems.
6. **Public Spaces and Displays:** Traditional interfaces in public spaces may not be user-friendly or efficient.
7. **Communication in Noise-Prone Environments:** Communication in noisy environments can be challenging.

Key Features/Objectives/Benefits:

1. **Non-Verbal Communication:** Hand gestures can be used for non-verbal communication in contexts such as video conferencing or virtual meetings. This adds a layer of expressiveness and engagement to remote interactions.
2. **Entertainment and Gaming:** Hand gestures are widely used in entertainment and gaming, offering a more immersive and interactive experience. Gamers can control characters, navigate virtual worlds, and

perform in-game actions using hand gestures, adding a new dimension to gameplay.

3. **Reduced Physical Strain:** In situations where prolonged use of traditional input devices may lead to physical strain or discomfort, hand gestures provide a more ergonomic alternative. Users can control devices with natural hand movements, reducing the risk of repetitive strain injuries.
4. **Innovative User Interfaces:** Gesture-based interfaces allow for the development of innovative user interfaces, breaking away from traditional input methods. This fosters creativity in design and opens up new possibilities for how users interact with and manipulate digital content.
5. **Hands-Free Operation:** Hand gestures enable hands-free operation of devices, which is particularly valuable in situations where manual input may be impractical or unsafe. This feature is beneficial in various scenarios, including cooking, driving, or medical procedures.
6. **Gesture Recognition Precision:** Advanced gesture recognition technologies enable precise tracking and interpretation of hand movements. This precision enhances the accuracy of input, making it more reliable for controlling devices.
7. **Reduced Learning Curve:** Compared to learning the functions of buttons or keyboard shortcuts, using hand gestures typically has a shorter learning curve. Users can quickly adapt to gestural controls, leading to faster and more efficient interactions.
8. **Gesture Customization:** Many gesture-based systems allow users to customize gestures to their preferences. This personalization enhances the user experience by tailoring controls to individual preferences and habits.

Methodology:

- **IDENTIFY TARGETS :** Will remove unwanted background and segment the objects .
- **TRACK OBJECTS :** Will identify the hand and track its motion in real time .

- **HAND RECOGNITION :** Limbs or structure of hands will be recognised as the input so as to generate the desired output.
- **DESIGNING VOCABULARY:** Defining the vocabulary database for different functionality like move left , right , up , down , clockwise , anticlockwise etc.
- **MAPPING GESTURES:** It works to recognise the hand gestures and give the output for the specified task that was given to it .
- **EXECUTION :** Linking the gesture recognition system with basic device functions.

Software Used:

1.Microsoft Visual Studio Code: Visual Studio Code (VSCode) is a powerful and versatile code editor that has gained immense popularity among developers, and it offers several compelling reasons for using it with Python. First and foremost, VSCode provides excellent support for Python development with features like intelligent code completion, syntax highlighting, and real-time error checking, enhancing your coding experience. The integrated terminal allows you to run Python scripts and commands directly within the editor, streamlining your workflow. VSCode also boasts a rich ecosystem of extensions, and the Python extension is particularly robust offering features like debugging support, virtual environment management, and integration with popular tools like Jupyter notebooks

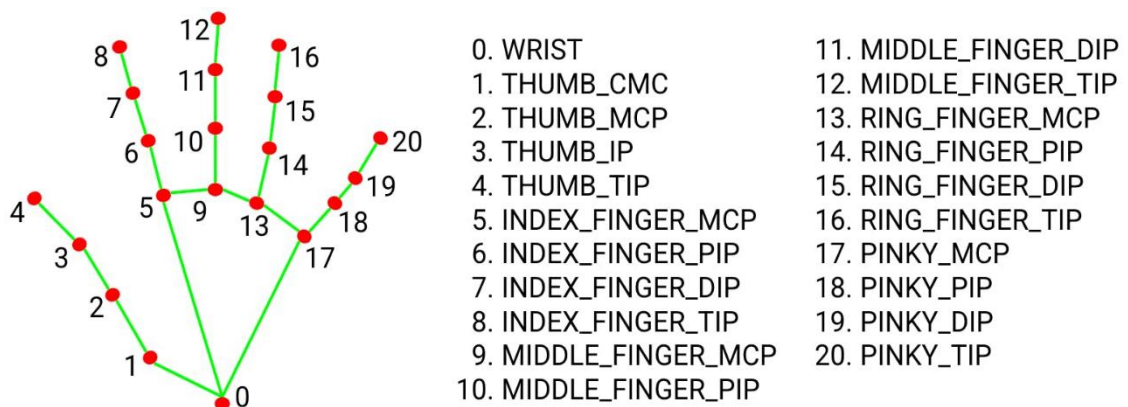
2.Python ide: Python is a versatile and easy-to-learn programming language with a rich ecosystem of libraries and frameworks. Many computer vision and image processing libraries, such as OpenCV and Mediapipe, have well-supported Python APIs, making it convenient for developers to implement hand gesture recognition algorithms.

Python Libraries Used:

1. OpenCV(cv2): for image and video processing. It's a powerful library for computer vision tasks.

2. NumPy: High-performance numerical operations library for arrays and matrices in Python.
3. pycaw: Windows Audio Control Wrapper for managing audio settings and devices.
4. Mediapipe: Google's library for building real-time, cross-platform computer vision applications.
5. comtypes: Python package for interacting with COM (Component Object Model) interfaces.
6. ctypes: Foreign Function Interface (FFI) library for calling C functions from Python.
7. math: Python's standard library for mathematical operations and functions.
8. Screen_brightness_control: API for adjusting screen brightness in Python.

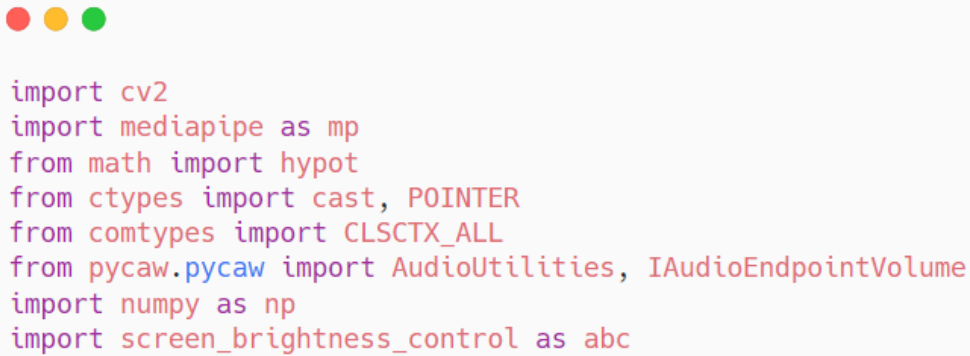
Referral Hand Image:



The above image shows the numbers of the points that MediaPipe uses to refer to different points of the hand. This tutorial will use point 4 and point 8 which are the thumb and the index finger respectively.

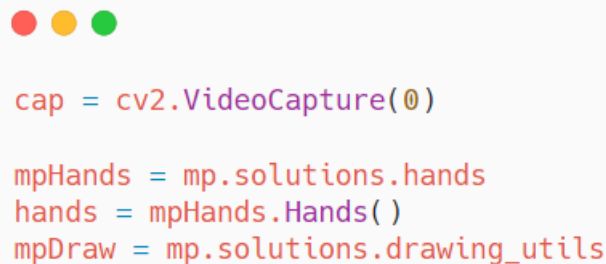
Code:

At first, we need to import all the libraries in to the Program. For this used several Python libraries such as opencv, numpy, pycaw, mediapipe, ctypes, ctypes, math, screen_brightness_control.



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

Setting up Camera and Mediapipe environment for hand detection

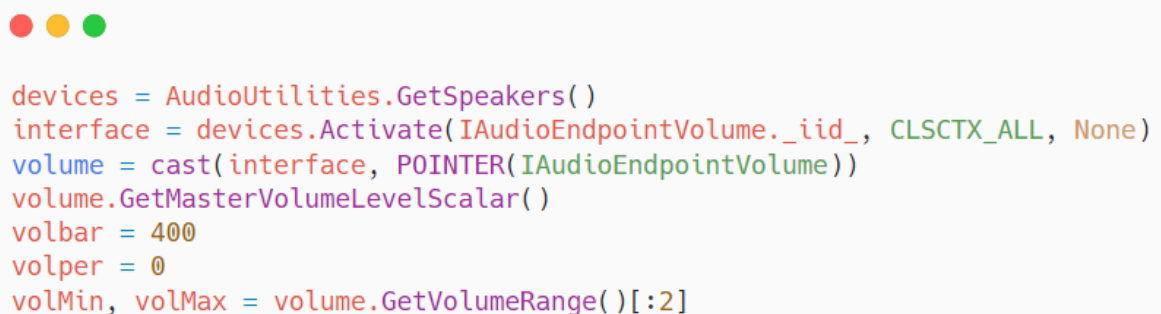


```
cap = cv2.VideoCapture(0)

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

Using pycaw library to interact with the Windows Audio API to get information about the system's audio devices and their volume settings.

Also getting the volume range.



```
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
volume.GetMasterVolumeLevelScalar()
volbar = 400
volper = 0
volMin, volMax = volume.GetVolumeRange()[ :2]
```

This is the part of the program that continuously captures video frames from a camera (cap) and processes these frames to detect hand gestures, presumably for volume control. This method processes the RGB image to detect hand gestures

```

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

    # Hand gesture for volume control
    results = hands.process(imgRGB)
    lmList = []

```

This part is using the MediaPipe library to detect hand landmarks in an image and using the positions of specific landmarks to control the volume. it uses mpDraw.draw_landmarks to draw connections between the landmarks on the image. The volume and length are printed, and rectangles are drawn on the image to represent the volume control. Index finger and thumb is used for controlling volume.(4, 8)

```

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, 110])
    volper = np.interp(length, [30, 350], [0, 50])
    print(f"Volume: {vol}, Length: {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)

```

This code uses the MediaPipe library to detect hand landmarks in an image and then uses the positions of specific landmarks to control the brightness of a device. Thumb and middle finger is used to control brightness.(4, 12)


```

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

if lmList != []:
    x1, y1 = lmList[4][1], lmList[4][2]
    x2, y2 = lmList[12][1], lmList[12][2]
    cv2.circle(img, (x1, y1), 4, (255, 0, 0), cv2.FILLED)
    cv2.circle(img, (x2, y2), 4, (255, 0, 0), cv2.FILLED)
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)

    length = hypot(x2 - x1, y2 - y1)
    bright = np.interp(length, [15, 220], [0, 100])
    print(f"Brightness: {bright}, Length: {length}")
    abc.set_brightness(int(bright))

```

This function is used to display an image in a window

```

cv2.imshow('Hand Gesture Control', img)

```

Close the camera windows if user presses k key.

```

if cv2.waitKey(1) & 0xFF == ord('k'):
    break

```

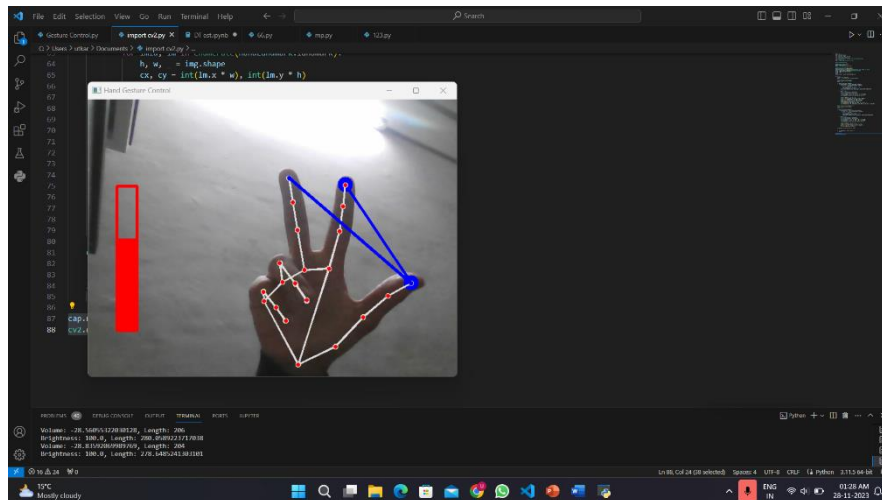
The provided Python code is used to release the video capture resource and close all OpenCV windows.

```

cap.release()
cv2.destroyAllWindows()

```

Results:



Advantages & Limitations:

1. Advantages:

- a) **Hands-free Operation:** Gesture control enables hands-free operation, allowing users to interact with devices without physically touching them. This is particularly useful in situations where touching the device may not be practical or hygienic, such as when cooking, exercising, or when hands are occupied.
- b) **Reduced Wear and Tear:** Touchscreens and physical buttons can experience wear and tear over time due to repeated use. Gesture control eliminates the need for physical contact, reducing the risk of mechanical failure and prolonging the lifespan of the device.
- c) **Accessibility:** Gesture control can enhance accessibility for individuals with physical disabilities or limitations. It provides an alternative input method that doesn't rely on fine motor skills or precise touch, making it more accessible to a broader range of users.
- d) **Aesthetic Appeal:** Gesture control can contribute to the aesthetic appeal of a device or interface. The absence of physical buttons or touch surfaces can result in a sleek and modern design, aligning with the preferences of users who appreciate minimalist aesthetics.

2. Limitations:

- i) Hand gestures may not always be precise and accurate, leading to unintended actions or difficulty in achieving the desired level of control. This can be frustrating for users who require fine-tuned adjustments.
- ii) The effectiveness of gesture control can be affected by environmental factors such as poor lighting conditions or obstacles in the surroundings. This can lead to unreliable performance and decreased user satisfaction.
- iii) Accidental gestures, limited gesture recognition accuracy, and the need for dedicated camera are a few limitations.
- iv) Gesture-based systems often involve cameras or sensors to detect hand movements. This raises privacy concerns as users may be uncomfortable with the idea of their gestures being continuously monitored, potentially capturing unintended information.
- v) Extended use of hand gestures, especially in the air, can lead to fatigue and muscle strain. Holding one's hand in mid-air for prolonged periods may not be as comfortable as using physical buttons or sliders.

Conclusion:

The end result of gesture recognition system is to generate a command based on the dynamic movement of the palm and that is given to the software . Gesture recognition is a topic in computer science and language technology which interpret human gestures via mathematical algorithms. Mid-air gesture-based systems are becoming ubiquitous. Many mid-air gestures control different kinds of interactive devices, applications, and systems. They are, however, still targeted at specific devices in specific domains and are not necessarily consistent across domain boundaries. A comprehensive evaluation of the transferability of gesture vocabulary between domains is also lacking . As part of our analysis, we clustered gestures according to the dimensions of an existing taxonomy to identify their common characteristics in different domains, and we investigated the extent to which existing mid-air gesture sets are consistent across different domains.

References:

1. About the softwares Used in gesture recognition :
<https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/>
2. Real time application of air gestures: https://mediapipe-studio.webapps.google.com/demo/gesture_recognizer
3. In-air gestures around unmodified mobile devices by Jie Song:
<https://dl.acm.org/doi/abs/10.1145/2642918.2647373>
4. Investigating mid-air gestures and handhelds in motion tracked environments
by Ville Mäkelä: <https://dl.acm.org/doi/abs/10.1145/2914920.2915015>

THANK YOU!!