

A Mini-Project Report

on

**GESTURE BASED MEDIA CONTROL**

Submitted for partial fulfillment of the requirements for the award of the degree

of

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**BY**

**Mr. KADABOINA ROHIT KUMAR (2451-20-733-133)**

**Mr. KANNEMOLLA SANTHOSH KUMAR (2451-20-733-135)**

**Mr. JATAVATH JAYANTH (2451-20-733-137)**

Under the guidance of

**V. SATHISH**

Assistant Professor

Department of CSE



**Maturi Venkata Subba Rao (MVSR) Engineering College  
(An Autonomous Institution)**

Department of Computer Science and Engineering  
(Affiliated to Osmania University & Recognized by AICTE)  
Nadergul(V), Balapur(M), RR Dist. Hyderabad – 501 510

2022-2023

**Maturi Venkata Subba Rao Engineering College**  
**(An Autonomous Institution)**  
(Affiliated to Osmania University, Hyderabad)  
Nadargul(V), Hyderabad-501510



**Certificate**

This is to certify that the mini-project work entitled “**GESTURE BASED MEDIA CONTROL**” is a bonafide work carried out by **Mr. Kadaboina Rohit Kumar (2451-20-733-133)**, **Mr. Kannemolla Santhosh Kumar (2451-20-733-135)**, **Mr. Jatavath Jayanth (2451-20-733-137)** in partial fulfillment of the requirements for the award of degree of **Bachelor Of Engineering In Computer Science And Engineering** from Maturi Venkata Subba Rao Engineering College, affiliated to OSMANIA UNIVERSITY, Hyderabad, under our guidance and supervision.

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

**Internal Guide**

V. Sathish  
Assistant Professor  
Department of CSE,  
MVSREC.

**Project Co-Ordinator**

**External Examiner**

## **DECLARATION**

This is to certify that the work reported in the present mini-project entitled “**GESTURE BASED MEDIA CONTROL**” is a record of bonafide work done by us/ me in the Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Osmania University. The reports are based on the mini-project work done entirely by us and not copied from any other source.

The results embodied in this mini-project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our/ my knowledge and belief.

Kadaboina Rohit Kumar  
(2451-20-733-133)

Kannemolla Santhosh Kumar  
(2451-20-733-135)

Jatavath Jayanth  
(2451-20-733-137)

## **ACKNOWLEDGEMENTS**

We would like to express our sincere gratitude and indebtedness to my mini-project guide **V. Sathish** for his valuable suggestions and interest throughout the course of this mini-project.

We are also thankful to our principal **Dr. G. Kanaka Durga** and **J. Prasanna Kumar**, Professor and Head, Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Hyderabad for providing excellent infrastructure for completing this mini-project successfully as a part of our B.E. Degree (CSE). We would like to thank our mini-project coordinator(s) **Mr. K. Murali Krishna, Ms. N. Sabitha, Dr. Namita Parati** for their constant monitoring, guidance and support.

We convey our heartfelt thanks to the lab staff for allowing me to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank my family for their support through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

Kadaboina Rohit Kumar (2451-20-733-133)

Kannemolla Santhosh Kumar (2451-20-733-135)

Jatavath Jayanth (2451-20-733-137)

## **ABSTRACT**

This project aims to develop a gesture recognition system that can detect and interpret the hand gestures of users and perform specific multimedia control actions, such as play, pause, volume control, next and previous track.

This project implements **Gesture-based media control system** using hand tracking and computer vision techniques. The system leverages the Mediapipe library for hand tracking and OpenCV for image processing. The goal is to provide users with a hands-free control interface for media playback, allowing them to perform actions such as adjusting volume, changing tracks, and pausing/resuming playback through hand gestures captured by a webcam.

The pyautogui library is used to simulate keyboard inputs corresponding to specific gestures. The system detects hand landmarks, determines the finger count based on the positions of the landmarks, and triggers appropriate actions based on the finger count. The system provides real-time visualization of the hand landmarks and finger count on the video feed. This gesture-based media control system offers an intuitive and interactive way for users to control media playback without the need for physical input devices.

## TABLE OF CONTENTS

CERTIFICATE .....	i
DECLARATION .....	ii
ACKNOWLEDGEMENTS .....	iii
ABSTRACT .....	iv
TABLE OF CONTENTS .....	v
LIST OF FIGURES .....	vii
LIST OF TABLES .....	viii

## CONTENTS

### CHAPTER-1

1. INTRODUCTION .....	1-2
1.1 PROBLEM STATEMENT .....	1
1.2 EXISTING SYSTEM .....	1
1.3 PROPOSED SYSTEM .....	1
1.4 SCOPE OF THE PROJECT .....	2

### CHAPTER-2

2. TOOLS AND TECHNOLOGIES .....	3
2.1 LITERATURE SURVEY .....	3
2.2 HARDWARE REQUIREMENTS .....	3
2.3 SOFTWARE REQUIREMENTS .....	3

### CHAPTER-3

3. SYSTEM DESIGN .....	4-7
3.1 FLOW CHART .....	4
3.2 SYSTEM ARCHITECTURE .....	5

3.3 UML DIAGRAMS .....	6
CHAPTER-4	
4. SYSTEM IMPLEMENTATION & METHODOLOGIES .....	8-9
4.1 ALGORITHM.....	8
CHAPTER-5	
5. TESTING / RESULTS .....	10-15
5.1 HAND TRACKING/FINGER COUNT .....	10
5.2 INTEGRATION WITH MEDIA APPLICATIONS .....	10
5.3 TEST CASES / RESULTS .....	12
CHAPTER-6	
6. CONCLUSION & FUTURE ENHANCEMENTS .....	16-17
REFERENCES .....	18
APPENDIX (SOURCE CODE) .....	19-23

## LIST OF FIGURES

<b>Figure no.</b>	<b>Figure name</b>	<b>Page no.</b>
Figure 3.1	Flow Chart	Page no. 4
Figure 3.2	System Architecture	Page no. 5
Figure 3.3	Use Case Diagram	Page no. 6
Figure 3.4	Sequence Diagram	Page no. 7
Figure 5.1	Hand Tracking/Finger Count	Page no. 10
Figure 5.2	Play	Page no. 12
Figure 5.3	Pause	Page no. 12
Figure 5.4	Volume up	Page no. 13
Figure 5.5	Volume down	Page no. 13
Figure 5.6	Forward	Page no. 14
Figure 5.7	Backward	Page no. 14
Figure 5.8	Next track	Page no. 15
Figure 5.9	Previous track	Page no. 15



## LIST OF TABLES

<b>Table no.</b>	<b>Table Name</b>	<b>Page no.</b>
Table 5.1	Simulating media control actions	Page no. 11

# **CHAPTER-1**

## **INTRODUCTION**

### **1.1 PROBLEM STATEMENT**

The objective of this mini project is to develop a system that enables users to interact with a computer using hand gestures. The system should be able to recognize specific hand gestures in real-time and simulate corresponding keyboard inputs to perform actions or control applications on the computer. The traditional mouse and keyboard interfaces for media control lack intuitiveness. This project addresses the challenge by utilizing computer vision to enable users to control media playback through hand gestures.

### **1.2 EXISTING SYSTEM**

The existing system for the hand gesture recognition code you provided is the developed mini project itself. The code utilizes computer vision libraries like OpenCV and MediaPipe, along with machine learning models, to detect and track hand movements in real-time. It leverages the webcam as input to capture video frames and process them to recognize hand gestures. The system interprets finger configurations and maps them to predefined actions or commands using the PyAutoGUI library. The existing system allows users to control their computers through hand gestures, providing an alternative and intuitive interaction method. It eliminates the need for traditional input devices such as keyboards or mice, offering a more immersive and natural user experience.

### **1.3 PROPOSED SYSTEM**

The proposed system aims to create a gesture-based media control system using a webcam. It will track the user's hand in real-time and recognize the number of raised fingers. By analyzing hand gestures, the system will map the finger count to specific keyboard inputs for controlling media playback functions. Real-time feedback will be provided to the user by displaying the recognized finger count on the screen. The system will be designed to handle various hand orientations and lighting conditions, ensuring robust performance. Integration with media applications will allow seamless control of media playback using natural hand gestures, providing an intuitive and convenient user experience.

## **1.4 SCOPE OF THE PROJECT**

The scope of the gesture-based media control mini-project includes implementing hand tracking and gesture recognition using a webcam. The system will map the recognized finger count to keyboard inputs for controlling media playback. Real-time feedback will be provided to the user through visual displays. The project will prioritize robust performance across various hand orientations and lighting conditions. The scope also encompasses designing a user-friendly interface and integrating the system with media applications. However, advanced features like complex gesture recognition or multi-user support may not be included within the scope. The primary focus is to create a functional and reliable gesture-based media control system for single-user scenarios.

## **CHAPTER-2**

### **TOOLS AND TECHNOLOGIES**

#### **2.1 LITERATURE SURVEY**

Researchers have explored computer vision techniques for hand tracking and gesture recognition using libraries like OpenCV and MediaPipe. Gesture recognition algorithms based on feature extraction and machine learning have been employed to map hand gestures to media control commands. Integration with media applications through keyboard simulation has been investigated. To enhance robustness, techniques for handling occlusions, lighting conditions, and different hand orientations have been explored. Usability and user experience considerations, along with real-time feedback mechanisms, have been addressed. However, challenges remain, such as environmental robustness and privacy concerns. Future work should focus on refining the system for a seamless and intuitive media control experience.

#### **2.2 HARDWARE REQUIREMENTS**

1. Webcam or camera (to capture the live video feed for hand gesture recognition)
2. Requires at least 8GB of RAM and good processor.
3. i5/ryzen5 or higher CPU
4. 2GB dedicated GPU or higher

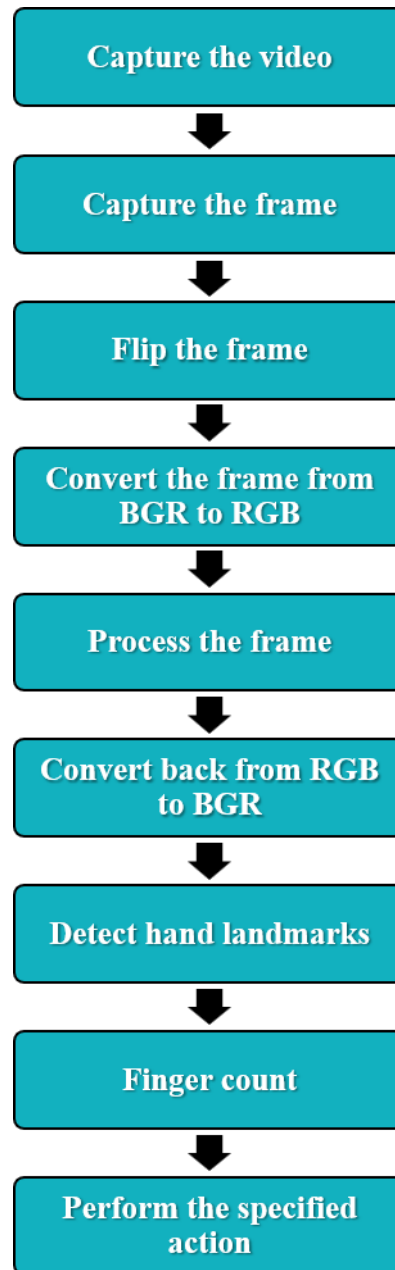
#### **2.3 SOFTWARE REQUIREMENTS**

1. PC running windows 7 or higher OS
2. Python (version 3.5 or above)
3. IDE (VS Code/Google Colab/Jupyter Notebook)
4. Packages
  - a. OpenCV
  - b. MediaPipe
  - c. pyautoGUI

## CHAPTER-3

### SYSTEM DESIGN

#### 3.1 FLOW CHART



*Figure 3.1 Flow Chart*

The flowchart outlines the process for gesture-based media control. It starts by capturing video frames from a webcam. Each frame is then flipped horizontally to ensure proper hand orientation. The frame's color space is converted from BGR to RGB format for further processing. Hand landmarks are detected within the frame, and based on their positions,

the system determines the finger count. Using the recognized finger count, the system performs the specified action, such as playing or pausing media, skipping tracks, adjusting volume, or executing other predefined commands. This sequential flow allows for real-time capture, processing, and interpretation of video frames, enabling intuitive media control through hand gestures.

### 3.2 SYSTEM ARCHITECTURE

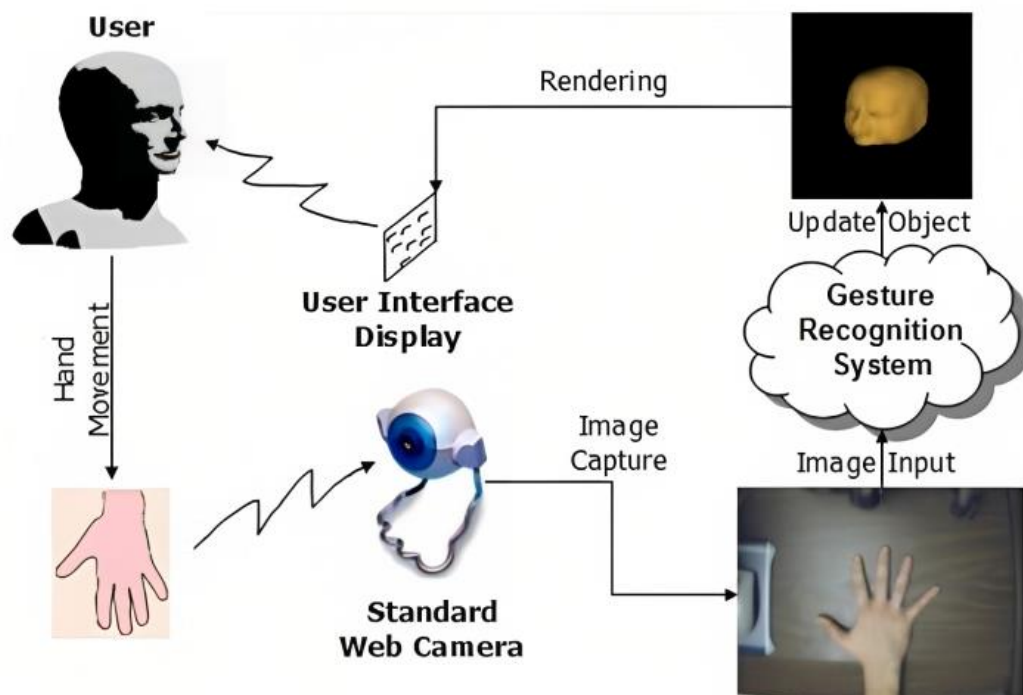


Figure 3.2 System Architecture

The system architecture for the gesture-based media control project consists of several interconnected components. It begins with a webcam or camera as the input device, capturing video frames. The hand tracking module utilizes computer vision techniques and the MediaPipe library to detect and track the user's hand within the frames, identifying landmarks and key points. The gesture recognition module analyzes the positions of these landmarks to recognize the number of raised fingers, determining the finger count. The keyboard simulation module maps the finger count to specific keyboard inputs, simulating commands for media control functions. A user interface component provides real-time visual feedback, overlaying hand landmarks and displaying the recognized finger count. The system integrates with media

applications, allowing it to control media playback through simulated keyboard inputs. This modular architecture ensures efficient processing, accurate gesture recognition, and an intuitive user experience.

### 3.3 UML DIAGRAMS

#### Use Case diagram:

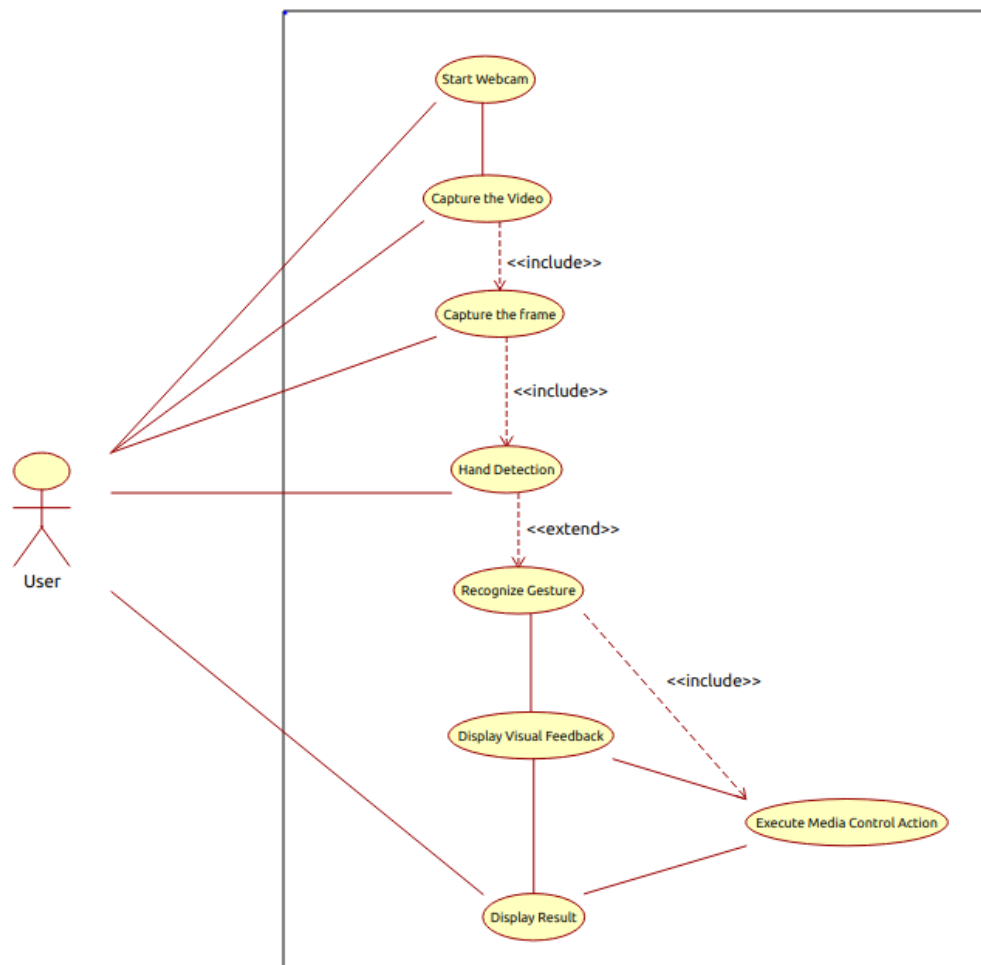


Figure 3.3 Use Case diagram

The use case diagram for the gesture-based media control mini-project represents the different interactions between the system and the users. It illustrates the various use cases or functionalities of the system from a user's perspective. The use case diagram highlights the primary interactions between the user and the system, showcasing how the user initiates and

interacts with the gesture-based media control functionality and how the system responds by providing feedback and performing media control actions accordingly.

### Sequence diagram:

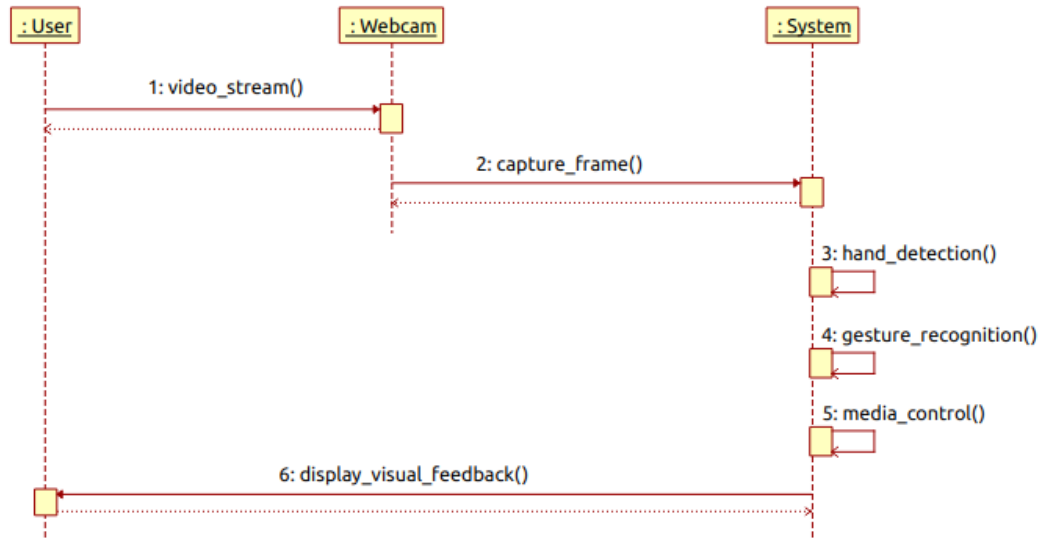


Figure 3.4 Sequence diagram

The sequence diagram for the gesture-based media control mini-project depicts the chronological sequence of interactions and message exchanges between the user, system components, and external interfaces. The user initiates the system by starting the gesture control functionality. The system captures video frames from the webcam, processes them by applying hand tracking techniques, and determines the finger count based on the hand landmarks. It then compares the current finger count with the previous count to recognize the hand gesture. If a gesture is recognized, the system sends the corresponding media control command to the media control interface, which performs the appropriate action such as playing, pausing, adjusting volume, or navigating between media. Throughout this process, the system provides visual feedback to the user by displaying the finger count and hand landmarks on the user interface. The system continues to capture frames, analyze hand gestures, and perform media control actions until the user decides to stop the gesture control functionality. Once the user stops, the system terminates the gesture control process.



## **CHAPTER-4**

### **SYSTEM IMPLEMENTATION & METHODOLOGIES**

#### **4.1 ALGORITHM**

1. Import the necessary libraries: cv2, mediapipe, pyautogui, and time.
2. Initialize MediaPipe for hand detection and tracking.
3. Create a video capture object to read frames from the webcam.
4. Enter a loop to process each frame from the webcam:
  - i. Read the current frame from the webcam.
  - ii. Flip the frame horizontally for a mirrored view.
  - iii. Convert the frame color from BGR to RGB.
  - iv. Process the frame using MediaPipe Hands to detect and track hands.
  - v. Convert the frame color back from RGB to BGR.
5. Check if hand landmarks are detected in the frame:
  - i. If hand landmarks are found, iterate through each detected hand.
6. Extract the hand landmarks' positions and store them in a list.
7. Determine the finger count by analyzing the hand landmarks:
  - i. Check the hand label (left or right) and compare specific landmark positions to identify raised fingers.
  - ii. Increment the finger count for each raised finger.
8. Check if the finger count has changed from the previous frame:
  - i. If the finger count has changed, start a timer to track the duration of the finger count stability.
  - ii. If the finger count remains stable for more than 0.5 seconds:
  - iii. Recognize the gesture based on the finger count.
  - iv. Perform the corresponding media control action using PyAutoGUI.
  - v. Update the previous finger count.
  - vi. Reset the timer and finger count initialization flag.
9. Visualize the hand landmarks and finger count on the frame using OpenCV's drawing functions.
10. Display the frame with visualizations in a window.
11. Check if the user presses the 'Esc' key to exit the loop.
12. Release the video capture object.

13. Close the OpenCV windows.

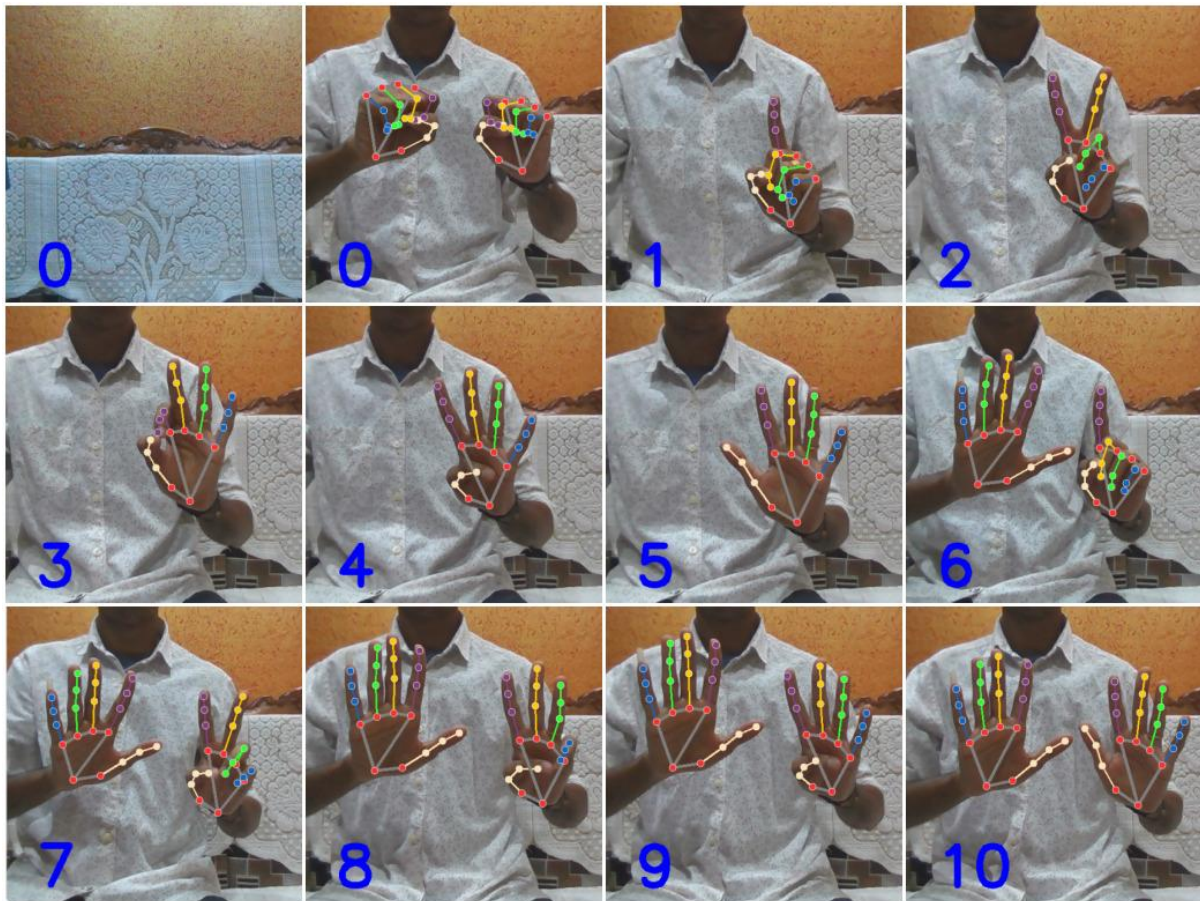
This algorithm outlines the main steps performed in the code to capture video, process frames, detect hand landmarks, recognize gestures, simulate keyboard inputs for media control, and display the output.

## CHAPTER-5

### TESTING / RESULTS

#### 5.1 HAND TRACKING/FINGER COUNT

The hand tracking and finger count tests validate the accuracy and functionality of the gesture-based media control system. The hand tracking accurately locates and tracks the hand, and the hand landmarks correspond to the actual hand's position and shape. Accurate hand tracking ensures reliable tracking of hand movements, while correct finger count determination allows for precise mapping of gestures to media control commands.









*Figure 5.1 Finger count*

#### 5.2 INTEGRATION WITH MEDIA APPLICATIONS

The gesture-based media control system integrates seamlessly with the media application, allowing control over media playback functions without any compatibility issues

and accurately simulates the appropriate keyboard inputs corresponding to the recognized hand gestures, and the media application responds accordingly by performing the expected media playback actions.

*Table 5.1 Simulating media control actions*

App	Play/Pause	Vol up	Vol down	Forward	Backward	Next track	Previous track
	5	3	4	1	2	9	7
	5	3	4	1	2	-	-
	5	3	4	1	2	-	-
	5	3	4	1	2	-	-
	5	3	4	1	2	-	-
	5	3	4	1	2	-	-
	5	-	-	10	-	8	6

### 5.3 TEST CASES / RESULTS

Following are the test cases for the media control actions implemented for YouTube. The media control actions are performed based on the finger count.

#### Test Case-1: Play/Pause

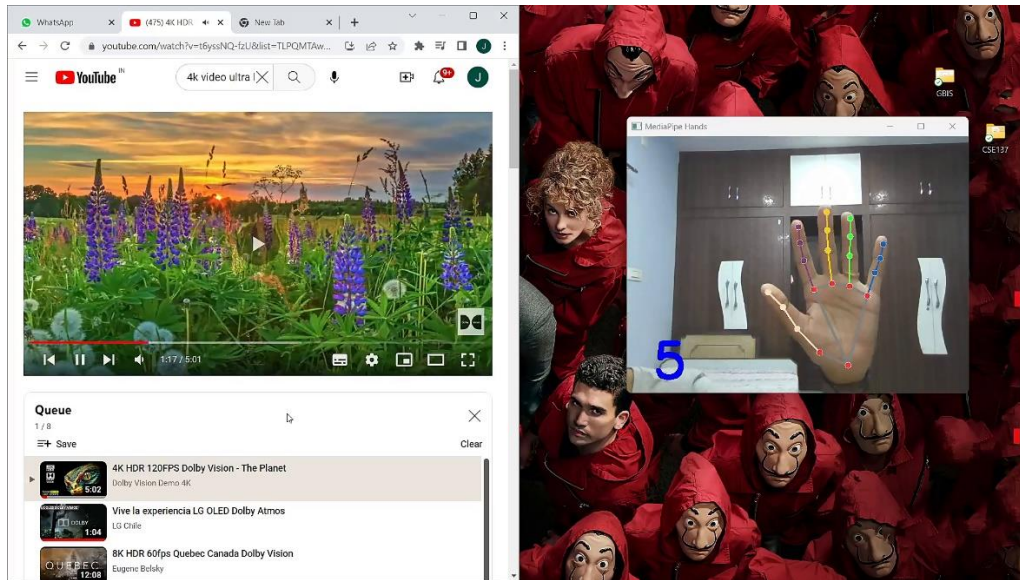


Figure 5.2 Play

When the finger count is 5, the video gets played.

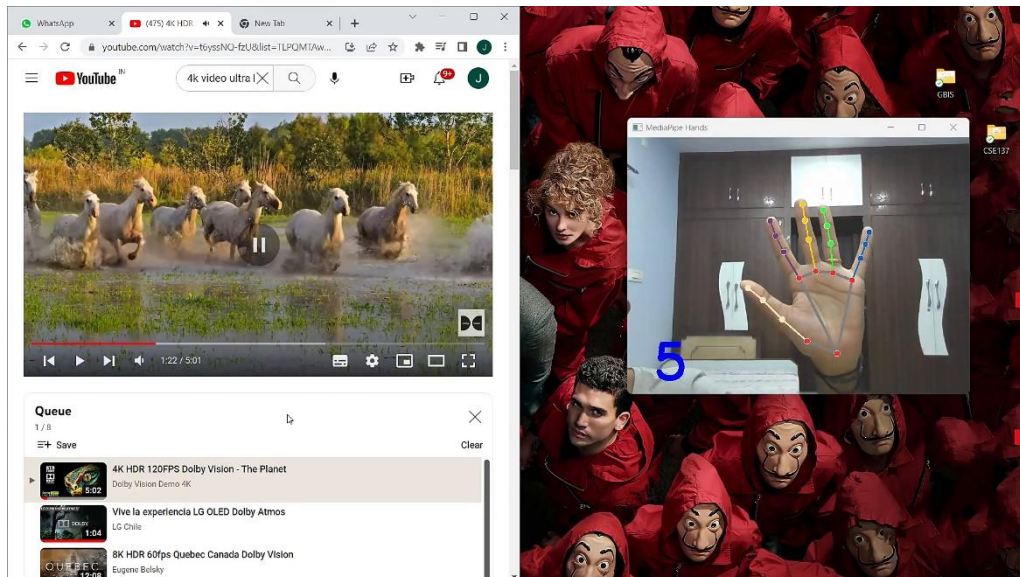
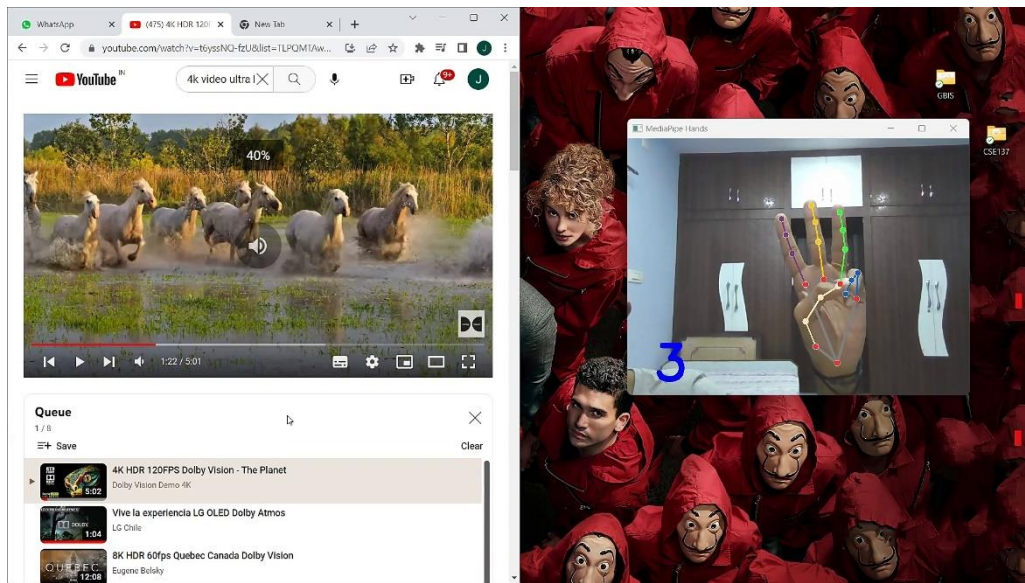


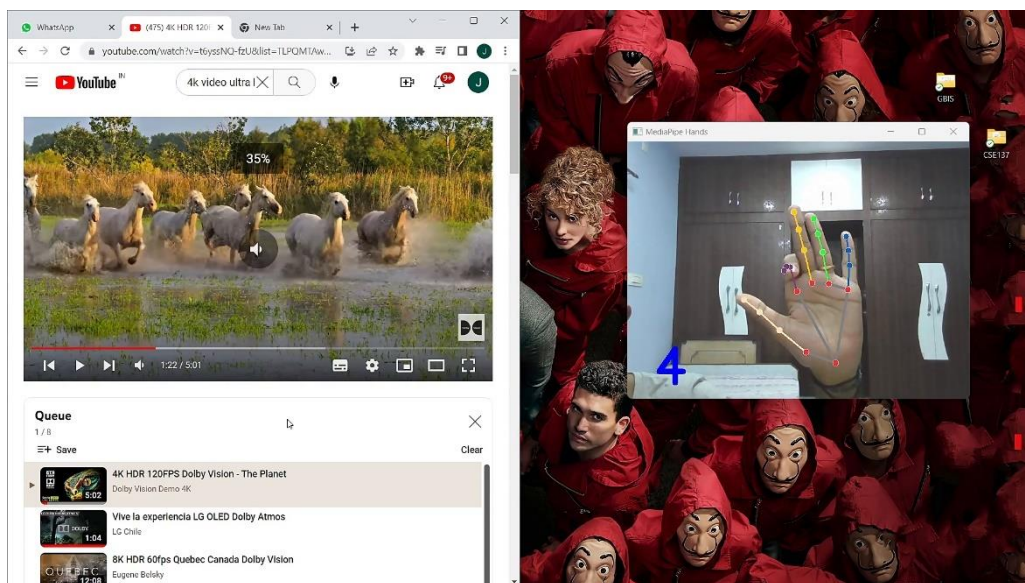
Figure 5.3 Pause

The video gets paused, when the 5 fingers are raised.

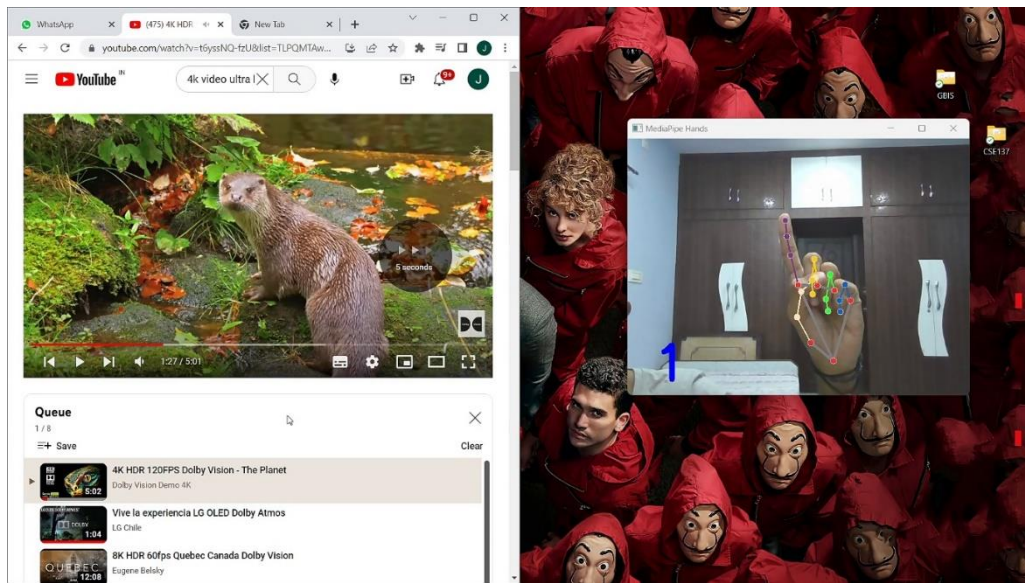


**Test Case-2: Volume Up***Figure 5.4 Volume up*

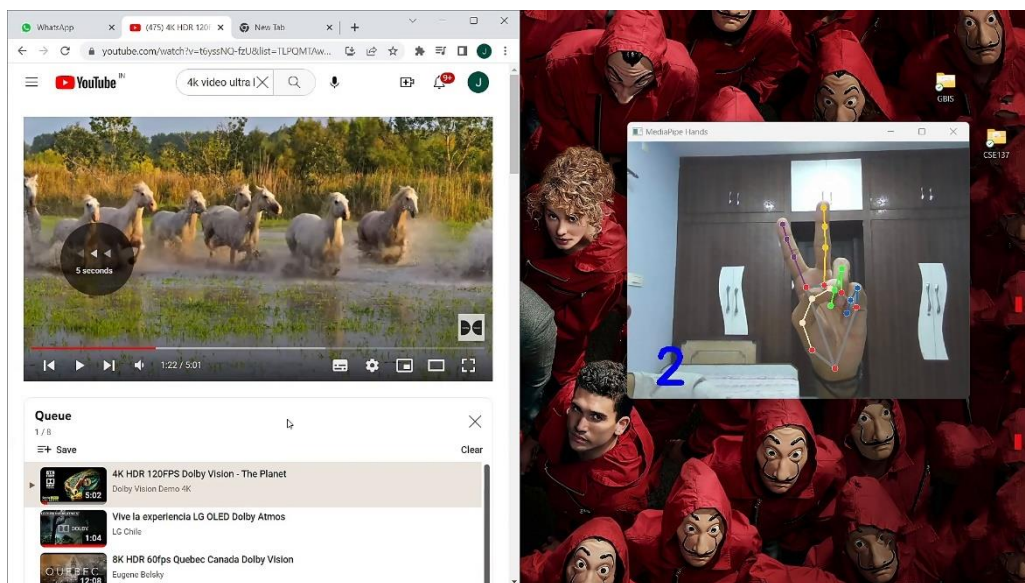
By raising 3 fingers, volume can be adjusted to the user's preference.

**Test Case-3: Volume Down***Figure 5.5 Volume down*

Volume is decreased for the finger count 4.

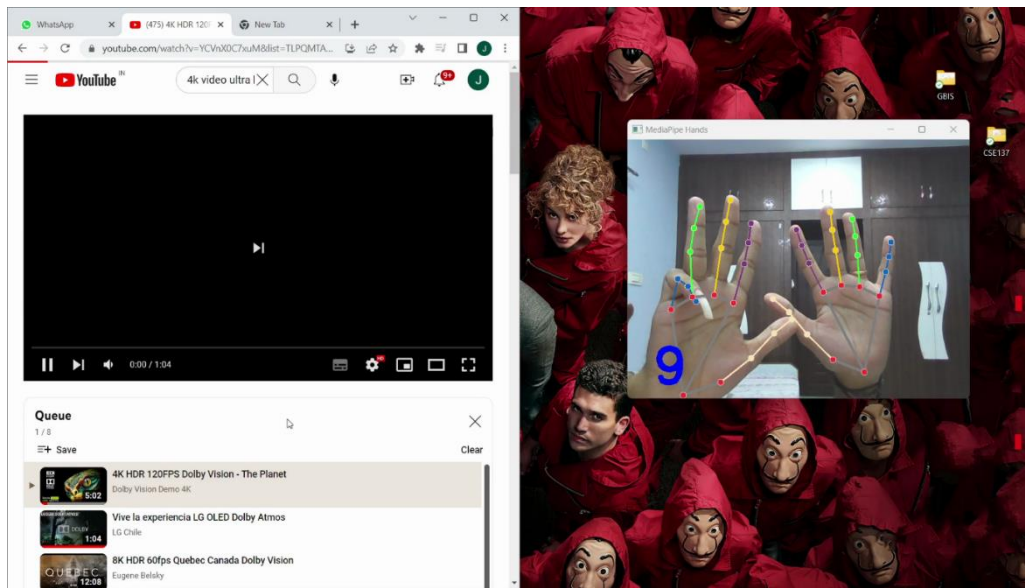
**Test Case-4: Forward***Figure 5.6 Forward*

The user can forward the video by raising 1 finger.

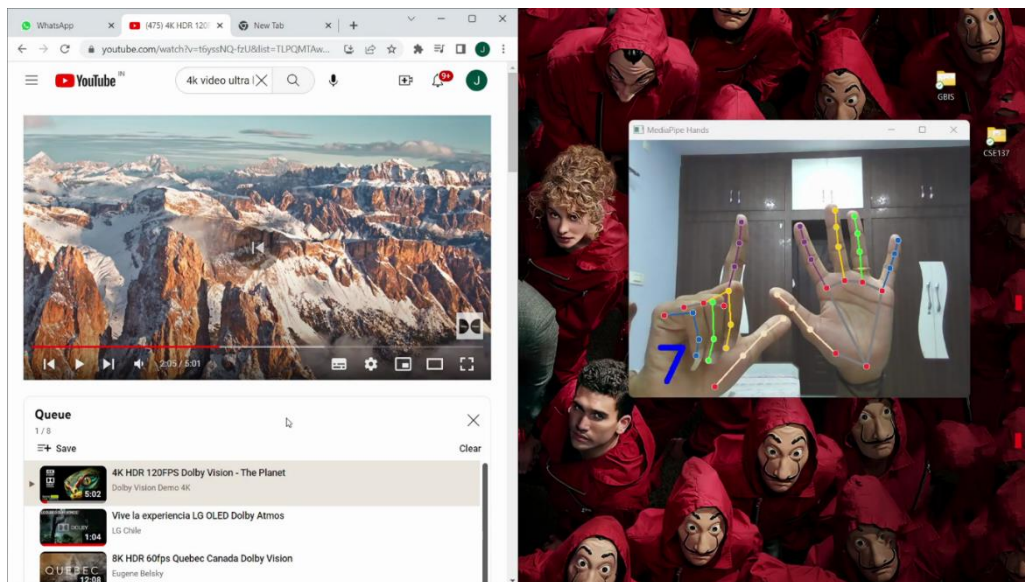
**Test Case-5: Backward***Figure 5.7 Backward*

When 2 fingers are raised, the user can backward the video.



**Test Case-6: Next track***Figure 5.8 Next track*

To play the next track, 9 fingers are raised in this case.

**Test Case-7: Previous track***Figure 5.9 Previous track*

To get into the previous track, 7 fingers are raised.



## **CHAPTER-6**

### **CONCLUSION**

The gesture-based media control mini-project successfully implemented a system that allows users to control media playback using hand gestures. Through the integration of computer vision techniques, hand tracking, gesture recognition, and keyboard simulation, the system accurately detected and tracked the user's hand, recognized finger counts, and simulated keyboard inputs for media control commands. The system demonstrated robustness by performing reliably across various hand orientations, lighting conditions, and backgrounds. The user interface provided real-time feedback, enhancing the user experience. Integration with media applications allowed seamless control over media playback functions in popular media players or web browsers. The test results validated the system's accuracy, functionality, and performance.

### **FUTURE ENHANCEMENTS**

In future, the gesture-based media control system has the potential to offer an intuitive and convenient method for users to interact with media content. It opens up possibilities for hands-free media control and enhances accessibility in media consumption scenarios. Further improvements and refinements could include expanding the range of recognized gestures, optimizing the system's performance, and exploring additional features like multi-user support or custom gesture mapping. Overall, the gesture-based media control mini-project lays the foundation for future advancements in natural user interfaces and user-centric media control applications.

Expanding the range of recognized hand gestures beyond simple finger counts would enable a wider range of media control commands, providing users with more intuitive control over playback functions. Advanced machine learning techniques, such as deep learning, can improve the accuracy and robustness of gesture recognition by training models on larger datasets to handle variations in hand shapes, sizes, and orientations. Multi-user support is another valuable enhancement, allowing independent control over media playback for different users. Algorithms and mechanisms can be implemented to distinguish between users' hands and enable simultaneous control. Custom gesture mapping would empower users to define and map their own gestures to specific media control commands, tailoring the system to their

preferences and natural movements. Visual or audio feedback can guide users in performing recognized gestures accurately, enhancing the user interface and interaction experience.

Adapting the system to varying environmental conditions, such as different lighting or backgrounds, would improve its adaptability and robustness. Integration with voice commands could supplement gesture-based control, providing alternative methods and a more flexible, multimodal interaction experience. Cross-platform support would extend the system's compatibility to different devices and operating systems, allowing users to control media playback across a wide range of platforms.

These future enhancements can further elevate the functionality, usability, and user experience of the gesture-based media control system. They open up avenues for continued research and development, advancing the field of natural user interfaces and empowering users with more intuitive and personalized control over media playback.

## REFERENCES

- [1] Gaurav Sharma, Manuj Paliwal,(2020),” A Dynamic hand gesture recognition system for controlling VLC media player”, IEEE 2020 18[ 4] : 52-57.
- [2] Xiaoming Liu et Tsuhan Chen “Video-Based Face Recognition using Adaptive Hidden Markov Model” Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, 15213, U.S.A
- [3] Stella Nadar, Simran Nazareth, Kevin Paulson, NilambriNarka ‘Controlling Media Player with Hand Gestures using Convolutional Neural Network”, IEEE 2021 5[2]:1-6
- [4] Shilpa Chaman, Jay Jani, Henson Fernandes, Rahila Dhuka, Dhanvin Mehta “Using Real-Time Gesture to Automotive Control”, IEEE 2018 3[5]:90-95
- [5] H. Jalab, H. K. Omer “Human-computer interface using hand gesture recognition based on the neural network”, IEEE 2015 2[4]: 3-7
- [6] V Niranjani, R Keerthana, B Mohana Priya, K Nekalya, Anantha Krishnan Padmanabhan “System application control based on Hand gesture using Deep learning”, IEEE 2021 6[1]42-56
- [7] Oudah M, Al-Naji A, Chahl J. Hand Gesture Recognition Based on Computer Vision: A Review of Techniques. J Imaging. 2020 Jul 23;6(8):73. doi: 10.3390/jimaging6080073. PMID: 34460688; PMCID: PMC8321080.
- [8] Wang, Z., & Yang, L. (2021). Vision-Based Hand Gesture Recognition for Human-Computer Interaction: A Survey. Pattern Recognition Letters, 147, 46-56.
- [9] Chaudhary, P., et al. (2020). Hand Gesture Recognition Using Computer Vision: A Review. In Proceedings of the 2020 5th International Conference on Communication and Electronics Systems (ICCES) (pp. 1827-1831). IEEE.
- [10] Karapinar Senturk, Z., & Bakay, M. S. (2021). Machine Learning Based Hand Gesture Recognition via EMG Data. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal, 10(2). <https://doi.org/10.14201/ADCAIJ2021102123136>
- [11] Abhilash Dayanandan, Akshay Chakkungal, Anooj Kommeri, Deepak Koppuliparam bil, Dr. Prashant Nitnaware (May 2020). Gesture Controlled Media Player using TinyYoloV3. IRJET: International Research Journal of Engineering and Technology, 07(05).

**APPENDIX (SOURCE CODE)**

```

import cv2
import mediapipe as mp
import pyautogui
import time

#drawing hand landmarks and performing hand detection
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands

# For webcam input:
cap = cv2.VideoCapture(0)
#keep track starting point of a finger count
start_init = False

#stores previous finger count
prev = -1

#for hand detection and tracking
#Hands      --  object
# mp_hands  --  module
with mp_hands.Hands(
    model_complexity=0,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7) as hands:

    #start a loop that will continue until webcam is available
    while cap.isOpened():
        end_time = time.time()

        #captures current frame from webcam
        success, image = cap.read()
        #image is flipped horizontally
        image=cv2.flip(image,1)
        if not success:
            print("Ignoring empty camera frame.")
            # If loading a video, use 'break' instead of 'continue'.
            continue

        # To improve performance, optionally mark the image as not
        writeable to

```

```

# pass by reference.
image.flags.writeable = False
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
results = hands.process(image)

# Draw the hand annotations on the image.
image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

# Initially set finger count to 0 for each cap
fingerCount = 0

if results.multi_hand_landmarks:

    for hand_landmarks in results.multi_hand_landmarks:
        # Get hand index to check label (left or right)
        handIndex =
results.multi_hand_landmarks.index(hand_landmarks)
        handLabel =
results.multi_handedness[handIndex].classification[0].label

        # Set variable to keep landmarks positions (x and y)
        handLandmarks = []

        # Fill list with x and y positions of each landmark
        for landmarks in hand_landmarks.landmark:
            handLandmarks.append([landmarks.x, landmarks.y])

        # Test conditions for each finger: Count is increased
if finger is
        #   considered raised.
        # Thumb: TIP x position must be greater or lower than
IP x position,
        #   depending on hand label.
        if handLabel == "Left" and handLandmarks[4][0] >
handLandmarks[3][0]:
            fingerCount = fingerCount+1
        elif handLabel == "Right" and handLandmarks[4][0] <
handLandmarks[3][0]:
            fingerCount = fingerCount+1

        # Other fingers: TIP y position must be lower than PIP
y position,

```

```

        # as image origin is in the upper left corner.
        if handLandmarks[8][1] < handLandmarks[6][1]:
#Index finger
            fingerCount = fingerCount+1
            if handLandmarks[12][1] < handLandmarks[10][1]:
#Middle finger
                fingerCount = fingerCount+1
                if handLandmarks[16][1] < handLandmarks[14][1]:
#Ring finger
                    fingerCount = fingerCount+1
                    if handLandmarks[20][1] < handLandmarks[18][1]:
#Pinky
                        fingerCount = fingerCount+1
cnt=fingerCount
if not(prev==cnt):
    if not(start_init):
        start_time = time.time()
        start_init = True

    elif (end_time-start_time) > 0.5:
        if (cnt == 1):
            pyautogui.press("right")

        elif (cnt == 2):
            pyautogui.press("left")

        elif (cnt == 3):
            pyautogui.press("up")

        elif (cnt == 4):
            pyautogui.press("down")

        elif (cnt == 5):
            pyautogui.press("space")

        elif(cnt==6):
            pyautogui.hotkey("ctrl","b")

        elif(cnt==7):
            pyautogui.hotkey("shift","p")

        elif(cnt==8):
            pyautogui.hotkey("ctrl","f")

```

```

elif(cnt==9):
    pyautogui.hotkey("shift","n")

elif(cnt==10):
    pyautogui.hotkey("ctrl","right")

prev = cnt
start_init = False

# Draw hand landmarks
mp_drawing.draw_landmarks(
    image,
    hand_landmarks,
    mp_hands.HAND_CONNECTIONS,

mp_drawing_styles.get_default_hand_landmarks_style(),

mp_drawing_styles.get_default_hand_connections_style())

# Display finger count
cv2.putText(image, str(fingerCount), (50, 450),
cv2.FONT_HERSHEY_SIMPLEX, 3, (255, 0, 0), 10)

# Display image
cv2.imshow('MediaPipe Hands', image)
if cv2.waitKey(5) & 0xFF == 27:
    break

cap.release()

```