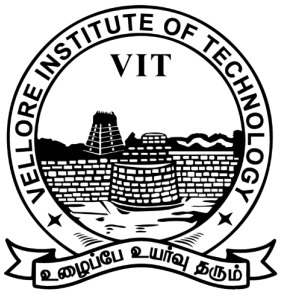


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Image and Video Analysis - CSE4079

Project Report

Media Manipulation Using Gestures

Submitted To: Dr. Saranyaraj D

Done By:

Dwarakanath.M (21MIA1160)

Vishwa amruth D (21MIA1158)

Santoshram M B (21MIA1137)

Table of Contents

S No	Title	Pages
1	Abstract	2
2	Introduction	2
3	Objectives	3
4	Scope	4
5	Literature Survey	5
6	Methodology	8
7	Implementation	11
8	Result	16
9	Conclusion	17
10	References	188

I Abstract

The project aims to develop an advanced, gesture-based media control system using MediaPipe Hands, an open-source platform by Google for machine learning and computer vision. The goal is to enable hands-free, real-time media control through hand gesture recognition, allowing users to interact with media applications without traditional input devices. By leveraging MediaPipe Hands, the system will detect and classify hand gestures, mapping them to media functions such as playback control, volume adjustment, and navigation. Key challenges include ensuring high accuracy, responsiveness, and real-time performance, while also providing visual feedback to confirm gesture recognition immediately. This innovative approach seeks to improve user interaction, enhance accessibility, and create a more intuitive, natural way to control media in various environments, such as smart homes and entertainment systems.

II Introduction

A comprehensive literature survey was conducted to understand the current landscape of hand gesture recognition technologies. The focus was on three key areas: existing methods for gesture recognition, applications of MediaPipe in various domains, and the integration of gesture-based controls in multimedia systems. The survey explored machine learning and deep learning approaches for hand gesture recognition, highlighting MediaPipe’s real-time hand tracking capabilities. It also examined MediaPipe's use in fields like AR, VR, and robotics, demonstrating its versatility. Additionally, the integration of gesture-based controls in multimedia systems, such as smart TVs and interactive gaming, was studied to understand its potential for improving user interaction. The outcome provided valuable insights into advancements in gesture recognition and its applications, informing the project’s approach. The next step is to refer to the detailed literature survey section for an in-depth overview of the background and related work.

III Objective

The objective of this project is to develop a cutting-edge, gesture-based media control system leveraging MediaPipe Hands, an advanced computer vision framework by Google. This system will enable users to interact with their media applications entirely through natural, hands-free gestures, eliminating the need for traditional input devices such as remote controls or keyboards. By integrating real-time, accurate hand gesture recognition, the system aims to provide an intuitive and seamless user experience, allowing users to control media functions—such as playback, volume adjustment, and content navigation—using simple hand movements.

The key feature of the project is real-time gesture recognition, which will be powered by advanced algorithms designed to detect and interpret a wide range of hand gestures accurately. These algorithms will focus on understanding natural hand movements and ensuring that even subtle gestures are detected with precision. This will enable more flexible and fluid interactions with media, making the system adaptable to different environments and user needs.

To enhance the user experience, the system will be optimized for responsiveness, aiming for minimal latency between the gesture input and the corresponding action on the media interface. The user interface will also be designed to be accessible, ensuring that individuals with varying levels of physical ability can easily use the system.

In terms of applications, this system has the potential to be used in a variety of contexts, such as smart home environments, entertainment systems, and augmented reality (AR) applications. In smart devices, users could interact with their TVs, speakers, or home assistants through gestures, controlling playback, adjusting volume, or navigating through menus without needing to physically touch a remote or screen.

IV Scope

The scope of hand gesture recognition in this project focuses on developing a real-time, accurate system that enables users to control media functions through natural hand movements. Using MediaPipe Hands, the system will detect and classify gestures such as swipes and fist clenches, mapping them to actions like play/pause, volume adjustment, and content navigation. The system will prioritize low-latency and high-accuracy gesture recognition to ensure smooth, responsive interaction. It aims to provide an intuitive, hands-free user experience with visual feedback confirming successful gesture recognition, while also adapting to various user needs and environments. Additionally, the system will be designed to work across different devices, including smart TVs, smartphones, and augmented reality environments. Beyond media control, the project will explore applications in smart devices, interactive experiences, and AR. The scope also includes addressing challenges such as environmental robustness, gesture variability, and ensuring accessibility for a wide range of users.

V Literature survey

“Controlling media player using hand gestures with VLC media player” by Monisha Sampath, Priyadarshini Velraj, Vaishnavii Raghavendran, M. Sumithra (2022) explores hand gestures for controlling VLC media player through real-time gesture recognition. Utilizing computer vision and deep learning, the system detects seven gestures for intuitive media control via a local device camera, enhancing convenience and efficiency without extra hardware.

“Gesture Recognition for Media Interaction: A Streamlit Implementation with OpenCV and MediaPipe” by Vaibhav Patil, Dr. Sanjay Sutar, Sanskruti Ghadage, Shubham Palkar (2023) integrated computer vision and machine learning, the system tracks hand movements and classifies them into gestures to control media functions like play, pause, and volume. The user-friendly interface also allows gesture customization. This innovative solution offers an immersive, controller-free media navigation experience, combining the power of OpenCV with Python.

“Consistent, Continuous, and Customizable Mid-Air Gesture Interaction for Browsing Multimedia Objects on Large Displays” by Arthur Slu¨yters, Quentin Selliera, Jean Vanderdonckta, Vik Parthibanb, and Pattie Maesb LUI is an interactive application designed for mid-air gesture interaction, allowing users to browse multimedia objects on large displays. The application aims to provide a consistent and customizable gesture set for various tasks, enhancing user experience in multimedia environments.

“Applying Hand Gesture Recognition for User Guide Application Using MediaPipe” by Indriani,Moh.Harris,Ali Suryaperdana Agoes The research focuses on developing a user guide application that employs hand gesture recognition to enhance interaction without physical devices. Utilizing the MediaPipe framework and Kinect camera, the application aims to facilitate user commands through gesture recognition, aligning with the advancements of Industry 4.0.

“Media Control Using Hand Gestures” by Mrigank Singh,Sheenu Rizvi This paper presents a software solution that enables users to control PowerPoint presentations and PDF files through hand gestures, eliminating the need for a keyboard or mouse. Utilizing Python libraries such as OpenCV and PyAutoGUI, the software captures input from a webcam to recognize gestures, enhancing mobility and accessibility for presenters. The application is particularly relevant in corporate environments and during times of social distancing, allowing seamless media control without physical contact.

“HandGesture Recognition Based on Computer Vision: A Review of Techniques” by Munir Oudah, Ali Al-Naji , Javaan Chahl The paper reviews hand gesture recognition systems, focusing on two main approaches: instrumented gloves and computer vision techniques. It discusses seven common techniques for gesture recognition and highlights their applications in various fields such as human-computer interaction, healthcare, and robotics.

“A Study on Real-time Hand Gesture Recognition Technology by Machine Learning-based MediaPipe” by Jeong-Seop Han, Choong-Iyeol Lee, Young-Hwa Youn , Sung-Jun Kim The study focuses on using MediaPipe, a machine learning-based technology, to recognize real-time hand gestures for children's educational activities. The aim is to help children explore nature indoors by using virtual mice controlled through hand movements. This approach is particularly relevant given the constraints on outdoor activities due to the

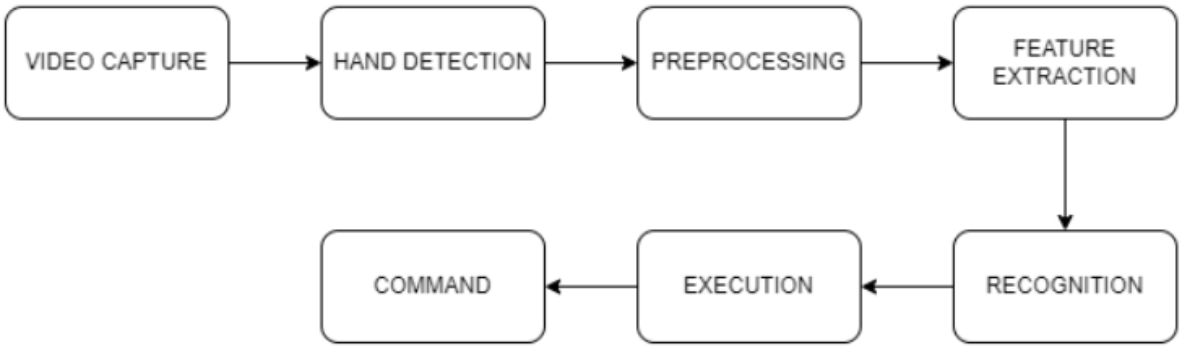
COVID-19 pandemic. The study aligns with the revised Nuri curriculum developed by the Ministry of Education of the Republic of Korea, which includes areas like physical exercise, health, communication, social relations, art experience, and nature exploration.

“Gesture tune & routesmart: revolutionizing volume control and vehicle navigation” by Ms. Sudha Chaturvedi, Dr. Tapsi Nagpal, Dr. Dinesh Javalkar The manuscript presents GestureTune and RouteSmart, innovative technologies that enhance driving safety and convenience through gesture-based volume control and optimized navigation, respectively, promoting safer driving experiences by minimizing distractions and providing dynamic rerouting capabilities.

“Lightweight real-time hand segmentation leveraging MediaPipe landmark detection” by Guillermo Sánchez-Brizuela, Ana Cisnal, Eusebio de la Fuente-López, Juan-Carlos Fraile, Javier Pérez-Turiel This paper presents a real-time hand gesture recognition system for media control, using MediaPipe for hand detection and dynamic skin segmentation, achieving high accuracy and over 90 FPS performance in diverse environments with no specialized hardware.

“Real-Time Hand Gesture Recognition Based on Deep Learning YOLOv3 Model” by Abdullah Mujahid,Mazhar Javed Awan , Awais Yasin , Mazin Abed Mohammed , Robertas Damaševi˘cius , Rytis Maskeliunas and Karrar Hameed Abdulkareem Abdullah Mujahid et al. present a lightweight hand gesture recognition model using YOLOv3 and DarkNet-53, achieving 97.68% accuracy without preprocessing. It's effective in real-time and complex environments, benefiting applications for individuals with communication disabilities.

VI Methodology



1. Video Capture:

The initial step in the system involves capturing a continuous stream of video frames to detect and recognize hand gestures in real-time. This stage uses a camera, typically a webcam or mobile device camera, to capture live video input. Each video frame is processed independently to ensure a responsive system that can immediately detect gestures as they occur. This raw video feed is then passed on to the hand detection stage for further analysis.

2. Hand Detection:

The hand detection stage focuses on identifying and isolating the hand regions within each video frame. MediaPipe Hands or similar hand detection algorithms are applied here to locate and track the user's hand positions. The algorithm generates bounding boxes around detected hands, which defines the area of interest for the next processing steps. By narrowing down the frame to specific hand regions, this stage reduces computational load and enables more precise analysis in subsequent stages. The coordinates or bounding box information from this step are then sent to the preprocessing module.

3. Preprocessing:

In the preprocessing stage, the detected hand regions are prepared for feature extraction by applying several image processing techniques. This step includes resizing, normalizing, and enhancing the hand regions to standardize the input data for better feature extraction accuracy. Additional adjustments, such as noise reduction and lighting correction, are also applied to ensure the data quality is high, minimizing the impact of environmental factors that could affect recognition accuracy. The output of this stage is a set of preprocessed hand region images that are ready for feature extraction.

4. Feature Extraction:

This stage is responsible for extracting meaningful features from the hand regions to facilitate gesture recognition. MediaPipe’s landmark detection, or similar methods, are employed to identify key hand landmarks, such as finger joints and positions. From these landmarks, relevant features like finger angles, distances between key points, and overall hand orientation are calculated. These measurements form a feature vector that provides a detailed representation of the hand's structure and positioning, which serves as input for the gesture recognition stage.

5. Recognition:

In the recognition stage, the extracted feature vectors are analyzed to identify specific gestures. Machine learning algorithms or rule-based classification techniques are used to interpret the feature data and match it to predefined gesture classes. For example, gestures like "fist" for play/pause, "thumbs up" for volume increase, or "swipe" for content navigation are recognized based on the feature patterns. The output of this stage is a recognized gesture label, which corresponds to a specific media control command that the system can execute.

6. Execution:

The execution stage is where the recognized gestures are mapped to specific media control actions. Each gesture has a predefined mapping that correlates to media functions such as playback, volume adjustment, or content navigation. For instance, if the recognition stage identifies a "fist" gesture, the system might interpret it as a play/pause command. This mapping allows the system to translate gestures into actions, effectively enabling gesture-based media control. The system then executes the appropriate command to control the media as intended.

7. Command:

Finally, the command stage provides feedback to the user regarding the executed action. This feedback can be visual (e.g., displaying icons or messages) or auditory (e.g., sound cues) to confirm that the gesture was recognized and the action was successfully executed. This immediate feedback enhances the user experience, helping users understand that their gesture-based commands have been correctly interpreted. Additionally, the feedback mechanism may log recognized commands for debugging or further system refinement, contributing to overall usability and user satisfaction.

VII Implementation

```
import cv2
import mediapipe as mp
import pyautogui
from pynput.mouse import Button, Controller
import util  # Import the utility functions
import time  # Import time for gesture delay

mouse = Controller()
screen_width, screen_height = pyautogui.size()

mpHands = mp.solutions.hands
hands = mpHands.Hands(
    static_image_mode=False,
    model_complexity=1,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7,
    max_num_hands=2  # Now detect both hands
)

# Gesture timer for debouncing
gesture_timer = {'Left': {'count': 0, 'last_time': time.time(), 'triggered': False}}

# Function to detect number of fingers for the left hand
def count_fingers(lst):
    cnt = 0
    thresh = (lst.landmark[0].y * 100 - lst.landmark[9].y * 100) / 2
    if (lst.landmark[5].y * 100 - lst.landmark[8].y * 100) > thresh:
        cnt += 1
    if (lst.landmark[9].y * 100 - lst.landmark[12].y * 100) > thresh:
        cnt += 1
    if (lst.landmark[13].y * 100 - lst.landmark[16].y * 100) > thresh:
        cnt += 1
    if (lst.landmark[17].y * 100 - lst.landmark[20].y * 100) > thresh:
        cnt += 1
    if (lst.landmark[5].x * 100 - lst.landmark[4].x * 100) > 6:
```

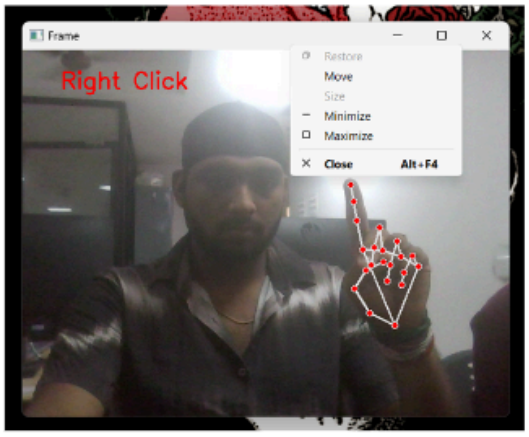
```
        cnt += 1
    return cnt

# Find the index finger tip for the right hand to control the mouse
def find_finger_tip(processed):
    if processed.multi_hand_landmarks:
        for idx, hand in enumerate(processed.multi_hand_landmarks):
            if processed.multi_handedness[idx].classification[0].label == "Right":  # Right hand
only
                                                                    index_finger_tip      =
hand.landmark[mpHands.HandLandmark.INDEX_FINGER_TIP]
                return index_finger_tip
    return None

# Move the mouse based on index finger of the right hand
def move_mouse(index_finger_tip):
    if index_finger_tip is not None:
        x = int(index_finger_tip.x * screen_width)
        y = int(index_finger_tip.y * screen_height)
        pyautogui.moveTo(x, y)

# Determine if left or right click for the right hand
def is_left_click(landmark_list, thumb_index_dist):
    return (
        util.get_angle(landmark_list[5], landmark_list[6], landmark_list[8]) < 50 and
        util.get_angle(landmark_list[9], landmark_list[10], landmark_list[12]) > 90 and
        thumb_index_dist > 50
    )

def is_right_click(landmark_list, thumb_index_dist):
    return (
        util.get_angle(landmark_list[9], landmark_list[10], landmark_list[12]) < 50 and
        util.get_angle(landmark_list[5], landmark_list[6], landmark_list[8]) > 90 and
        thumb_index_dist > 50
    )

# Detect gestures for both mouse (right hand) and keyboard (left hand)
```

```
def detect_gesture(frame, landmark_list, processed, handedness):
    global gesture_timer  # To track gesture timing globally
    if len(landmark_list) >= 21:
        index_finger_tip = find_finger_tip(processed)
        thumb_index_dist = util.get_distance([landmark_list[4], landmark_list[5]])

        if handedness == "Right" and index_finger_tip is not None:
            # Right hand controls mouse
                if thumb_index_dist < 50 and util.get_angle(landmark_list[5], landmark_list[6],
landmark_list[8]) > 90:
                move_mouse(index_finger_tip)
            elif is_left_click(landmark_list, thumb_index_dist):
                mouse.press(Button.left)
                mouse.release(Button.left)
                  cv2.putText(frame, "Left Click", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 2)
            elif is_right_click(landmark_list, thumb_index_dist):
                mouse.press(Button.right)
                mouse.release(Button.right)
                 cv2.putText(frame, "Right Click", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 0, 255), 2)
        elif handedness == "Left":
            # Left hand controls keyboard
              cnt = count_fingers(processed.multi_hand_landmarks[0])  # Finger counting for the
left hand

            current_time = time.time()
            # Debounce the gesture: check if it's the same gesture and if enough time has passed
            if cnt == gesture_timer['Left']['count']:
                if current_time - gesture_timer['Left']['last_time'] > 0.5:  # Wait for 0.5 seconds
                    if not gesture_timer['Left']['triggered']:  # Avoid repeated triggers
                        if cnt == 1:
                            pyautogui.press("right")  # Press the right arrow key
                                                    cv2.putText(frame, "Right Arrow", (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
                        elif cnt == 2:
                            pyautogui.press("left")  # Press the left arrow key
```

```
                                        cv2.putText(frame, "Left Arrow", (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
                elif cnt == 3:
                    pyautogui.press("volumeup")  # Press volume up
                                        cv2.putText(frame, "Volume Up", (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
                elif cnt == 4:
                    pyautogui.press("volumedown")  # Press volume down
                                        cv2.putText(frame, "Volume Down", (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
                gesture_timer['Left']['triggered'] = True  # Mark that the action is triggered
        else:
            # Reset the timer and gesture when a new gesture is detected
            gesture_timer['Left'] = {'count': cnt, 'last_time': current_time, 'triggered': False}

def main():
    draw = mp.solutions.drawing_utils
    cap = cv2.VideoCapture(0)

    try:
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break
            frame = cv2.flip(frame, 1)
            frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            processed = hands.process(frameRGB)

            landmark_list = []
            if processed.multi_hand_landmarks:
                for idx, hand in enumerate(processed.multi_hand_landmarks):
                    handedness = processed.multi_handedness[idx].classification[0].label  # Left or
Right hand
                    draw.draw_landmarks(frame, hand, mpHands.HAND_CONNECTIONS)
                    for lm in hand.landmark:
                        landmark_list.append((lm.x, lm.y))
                    detect_gesture(frame, landmark_list, processed, handedness)
```

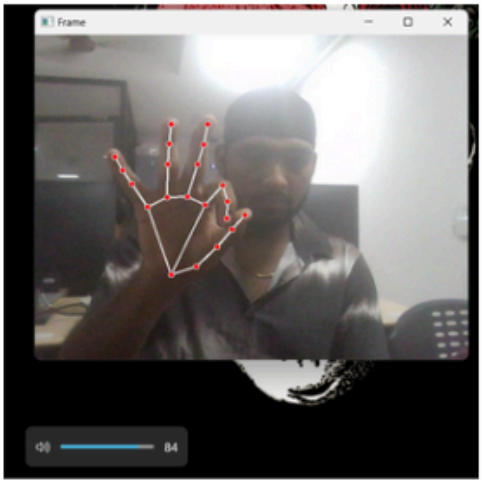
```
            cv2.imshow('Frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
    finally:
        cap.release()
        cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

VIII Result

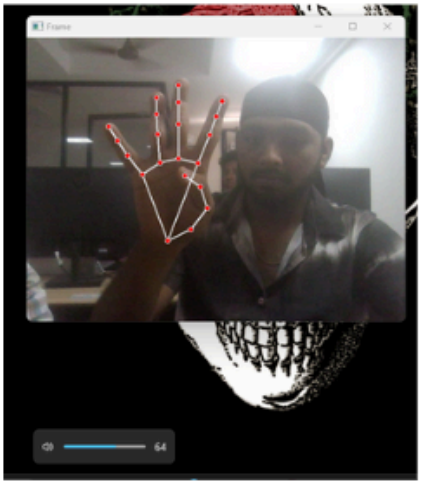


Right click

Left click



volume up



volume down

IX Conclusion

The hand gesture-based control system presented in this code demonstrates a highly effective and intuitive approach to hands-free interaction with computers. By leveraging MediaPipe for hand landmark detection and OpenCV for real-time video processing, the solution allows users to control the mouse and keyboard through simple, natural hand gestures. This implementation not only enhances user accessibility but also introduces a novel way of interacting with machines, which can be particularly useful in scenarios such as touchless interfaces, gaming, and accessibility tools for people with physical limitations.

Key strengths include:

Real-time performance: The system processes gestures with minimal delay, making it practical for everyday use.
Gesture versatility: Both hands are utilized for different purposes — the right hand controls the mouse, while the left hand simulates keyboard inputs, extending the interaction possibilities.

This code forms a solid foundation for future work, where additional gestures and features could be added, potentially creating an even more comprehensive gesture-based interface system. With some improvements, such as fine-tuning gesture recognition or expanding its applicability, this approach could be extended to fields like virtual reality, remote presentations, or healthcare applications for sterile environments.

X Reference

Sampath, M., Raghavendran, V., & Sumithra, M. (2022). Controlling media player using hand gestures with VLC media player. World Journal of Advanced Research and Reviews, 14(3), 466-472.

Patil, V., Sutar, S., Ghadage, S., & Palkar, S. (2023). Gesture Recognition for Media Interaction: A Streamlit Implementation with OpenCV and MediaPipe. International Journal for Research in Applied Science & Engineering Technology (IJRASET).

Ahmed, A., & Kapoor, S. Media Player Control through Hand Gesture and Facial Detection.

Singh, M., & Rizvi, S. (2021). Media Control Using Hand Gestures. Journal of Informatics Electrical and Electronics Engineering (JIEEE), 2(1), 1-11.

Udvag, R. M., Kumar, T. H., Kavin Raj, R. S., & Karthikeyan, M. P. (2020). Manipulation of Web Using Gestures. Journal of Computational and Theoretical Nanoscience, 17(8), 3782-3785.

Sampath, M., Raghavendran, V., & Sumithra, M. (2022). Controlling media player using hand gestures with VLC media player. World Journal of Advanced Research and Reviews, 14(3), 466-472.

Sangeetha, R. G., Hemanth, C., Nair, K. S., Nair, A. R., & Shine, K. N. (2022, December). Hand Gesture Control of Video Player. In International Conference on Hybrid Intelligent Systems (pp. 726-735). Cham: Springer Nature Switzerland.

Oudah, M., Al-Naji, A., & Chahl, J. (2020). Hand gesture recognition based on computer vision: a review of techniques. journal of Imaging, 6(8), 73.

Real-Time Hand Gesture Recognition Based on Deep Learning YOLOv3 Model