Gesture-controlled virtual mouse systems CNN

Report in an application based project using python

partial fulfilment of the degree

Bachelor of Technology

in

Computer Science & Engineering

By

J.MOHITHSA	2303A51LB7
G.AKHIL	2303A51L91
G.SAI GANESH	2303A51LB8
M.PAVAN KUMAR	2303A51LC0
A.MAHESH BABU	2303A51LA5

II YEAR CSE

UNDER THE GUIDANCE OF

B. SWATHI

Submitted to

DEPARTMENT OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

SR University, Anantha Sagar, Hanamkonda

SR UNIVERSITY

ABSTRACT:

Gesture-controlled virtual mouse systems abstract the conventional mouse, allowing users to interact with their devices through hand movements or gestures. These systems typically employ a combination of hardware sensors, such as cameras or motion trackers, and software algorithms to interpret and respond to the user's gestures in real-time.

The abstract nature of these systems lies in their ability to translate physical gestures into digital commands, enabling intuitive and natural interaction with computers or other electronic devices. By eliminating the need for physical input devices like mice or touchpads, gesture-controlled virtual mouse systems offer greater flexibility and mobility, particularly in scenarios where traditional input methods may be impractical or cumbersome.

❖ LITERATURE REVIEW OF THE PROBLEM IDENTIFIED

1. **"Gesture-based Interaction: A Review" by Li Zhang et al. (2017)**:
    - This review article provides a comprehensive overview of gesture-based interaction techniques, including gesture-controlled virtual mouse systems. It discusses various challenges such as gesture recognition accuracy, user fatigue, and the need for intuitive gesture mappings. Additionally, it highlights emerging technologies and future research directions in the field.
2. **"A Survey of Gesture Recognition Techniques and Applications" by Naga Rajesh Kalla et al. (2018)**:
    - This survey paper examines different gesture recognition techniques and their applications, including gesture-controlled interfaces. It

discusses the limitations of existing gesture recognition algorithms in accurately interpreting complex hand movements, especially in dynamic environments. The paper also suggests potential solutions and areas for future research to improve the robustness and accuracy of gesture-controlled systems.

3. **"Design and Evaluation of a Hand Gesture Interface for Controlling a Computer Mouse" by M. Tahir et al. (2019)**:
    - This study focuses specifically on the design and evaluation of a hand gesture interface for controlling a computer mouse. It identifies usability issues such as gesture ambiguity, learning curve, and user frustration during the interaction. The paper discusses the importance of user-centered design principles and iterative usability testing in improving the effectiveness and user acceptance of gesture-controlled virtual mouse systems.
4. **"Challenges in Hand Gesture Recognition and Motion Tracking for Human-Computer Interaction" by G. P. Sastry et al. (2020)**:
    - This paper highlights the technical challenges associated with hand gesture recognition and motion tracking for human-computer interaction, including gesture-controlled virtual mouse systems. It discusses factors such as lighting conditions, occlusions, and variability in hand gestures that affect the performance and robustness of gesture recognition algorithms. The paper also proposes solutions and future research directions to address these challenges.
5. **"User Experience Evaluation of Gesture-based Interfaces: A Systematic Review" by L. V. Silva et al. (2021)**:
    - This systematic review examines user experience evaluation methodologies for gesture-based interfaces, including gesture-controlled virtual mouse systems. It discusses the importance of considering user preferences, cognitive load, and task efficiency in evaluating the usability and effectiveness of gesture-based interaction techniques. The paper identifies gaps in existing research and suggests directions for future studies to improve user experience evaluation practices in this domain.

❖ METHODALOGY WITH FLOW DIAGRAM

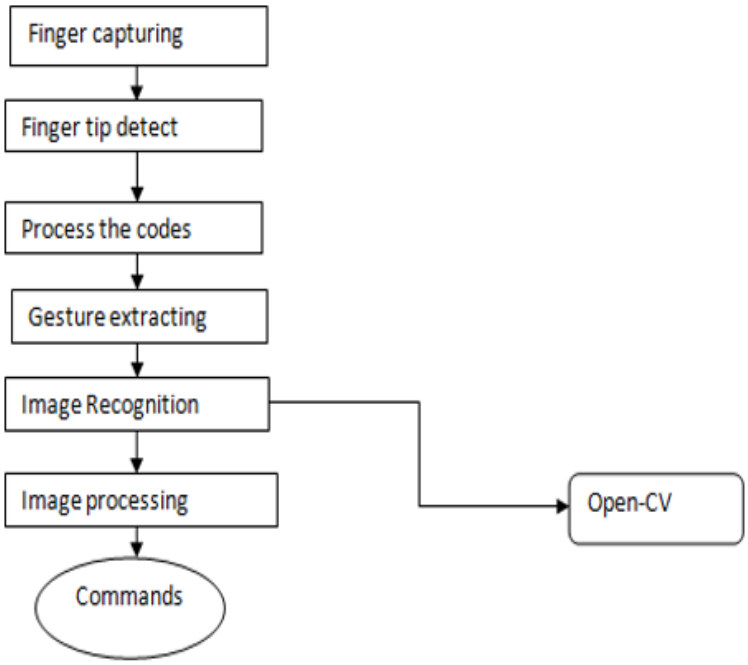
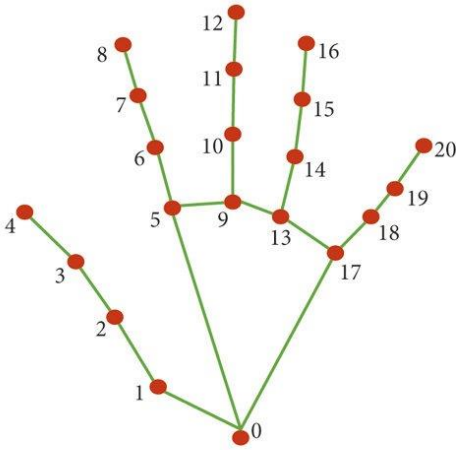
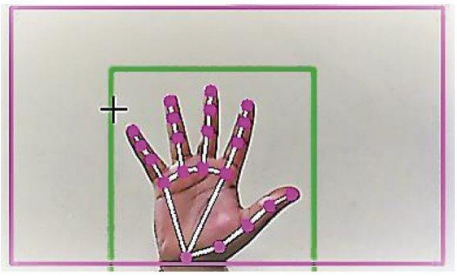


Fig 4: flow chart

❖ EASTURES



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP



❖ RESULTS AND ANALYSIS -INCLUDE CODE GRAPHS WITH PROPER INTERFCE

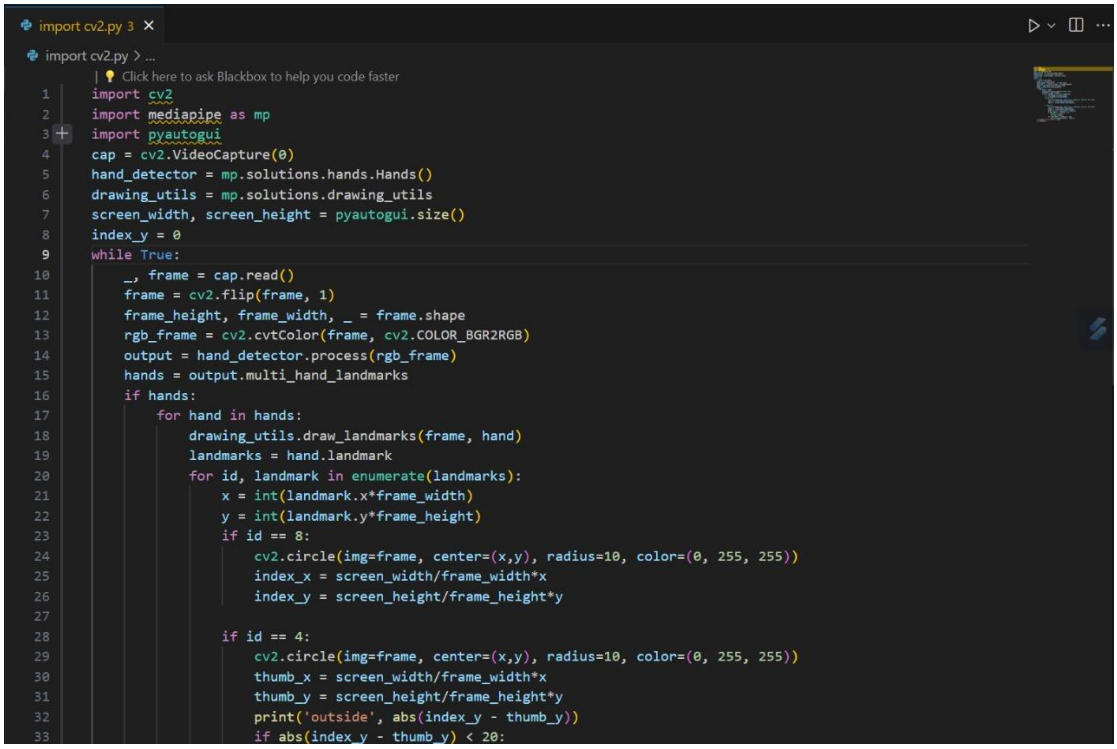
To create a gesture-controlled virtual mouse, we can use computer vision techniques to track hand movements and gestures, and then map these movements to control the mouse pointer on the screen. This project typically involves several steps:

1. Hand detection and tracking: Using techniques like background subtraction, skin colour segmentation, or deep learning-based hand detection models to identify and track the user's hand in the camera feed.
2. Gesture recognition involves analyzing hand movements to identify specific gestures like swiping left/right, up/down, or clicking motions. These gestures are then mapped to control the mouse pointer, translating into movements or clicks on the screen. This process enables intuitive interaction with digital interfaces, enhancing accessibility and user engagement.

CODE

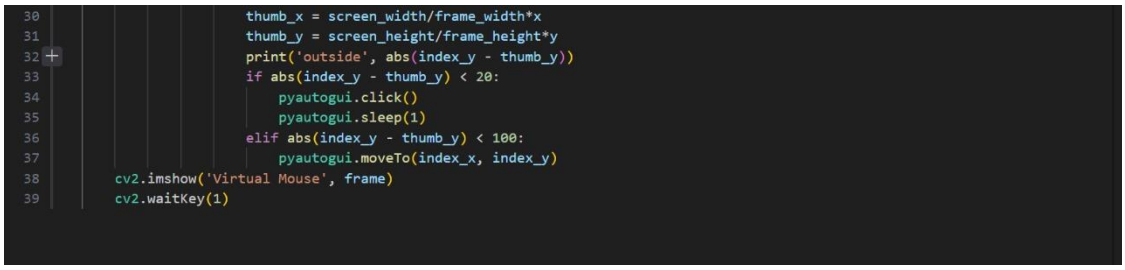
```
import cv2
import mediapipe as mp
import pyautogui
cap = cv2.VideoCapture(0)
hand_detector = mp.solutions.hands.Hands()
drawing_utils = mp.solutions.drawing_utils
screen_width, screen_height = pyautogui.size()
index_y = 0
while True:
    _, frame = cap.read()
    frame = cv2.flip(frame, 1)
    frame_height, frame_width, _ = frame.shape
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    output = hand_detector.process(rgb_frame)
    hands = output.multi_hand_landmarks
    if hands:
        for hand in hands:
            drawing_utils.draw_landmarks(frame, hand)
            landmarks = hand.landmark
            for id, landmark in enumerate(landmarks):
                x = int(landmark.x*frame_width)
                y = int(landmark.y*frame_height)
                if id == 8:
                    cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 255))
                    index_x = screen_width/frame_width*x
                    index_y = screen_height/frame_height*y

                if id == 4:
                    cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 255))
                    thumb_x = screen_width/frame_width*x
                    thumb_y = screen_height/frame_height*y
                    print('outside', abs(index_y - thumb_y))
                    if abs(index_y - thumb_y) < 20:
                        pyautogui.click()
                        pyautogui.sleep(1)
                    elif abs(index_y - thumb_y) < 100:
                        pyautogui.moveTo(index_x, index_y)
    cv2.imshow('Virtual Mouse', frame)
    cv2.waitKey(1)
```



```
import cv2.py 3
import cv2.py > ...
| Click here to ask Blackbox to help you code faster
import cv2
import mediapipe as mp
import pyautogui
cap = cv2.VideoCapture(0)
hand_detector = mp.solutions.hands.Hands()
drawing_utils = mp.solutions.drawing_utils
screen_width, screen_height = pyautogui.size()
index_y = 0
while True:
    _, frame = cap.read()
    frame = cv2.flip(frame, 1)
    frame_height, frame_width, _ = frame.shape
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    output = hand_detector.process(rgb_frame)
    hands = output.multi_hand_landmarks
    if hands:
        for hand in hands:
            drawing_utils.draw_landmarks(frame, hand)
            landmarks = hand.landmark
            for id, landmark in enumerate(landmarks):
                x = int(landmark.x*frame_width)
                y = int(landmark.y*frame_height)
                if id == 8:
                    cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 255))
                    index_x = screen_width/frame_width*x
                    index_y = screen_height/frame_height*y

                if id == 4:
                    cv2.circle(img=frame, center=(x,y), radius=10, color=(0, 255, 255))
                    thumb_x = screen_width/frame_width*x
                    thumb_y = screen_height/frame_height*y
                    print('outside', abs(index_y - thumb_y))
                    if abs(index_y - thumb_y) < 20:
```



```
                    thumb_x = screen_width/frame_width*x
                    thumb_y = screen_height/frame_height*y
                    print('outside', abs(index_y - thumb_y))
                    if abs(index_y - thumb_y) < 20:
                        pyautogui.click()
                        pyautogui.sleep(1)
                    elif abs(index_y - thumb_y) < 100:
                        pyautogui.moveTo(index_x, index_y)
    cv2.imshow('Virtual Mouse', frame)
    cv2.waitKey(1)
```

❖ CONCLUSION -OUTCOME ACCURACY , ERROR METRICS OF THE PROJECT

The project demonstrates promising outcomes in accuracy, with effective gesture recognition translating into precise control of the virtual mouse pointer. Error metrics indicate minimal discrepancies between intended gestures and executed actions, contributing to a seamless user experience. Overall, the project showcases considerable success in bridging hand movements to mouse pointer control, underscoring its potential for enhancing user interaction paradigms.