Eye Controlled Mouse Using Face Mesh Detection

Anonymous CVPR submission

Paper ID Group 21

Abstract

In this work, we present a novel system for controlling a computer mouse using facial landmark detection via a face mesh. The system leverages OpenCV, MediaPipe, and PyAutoGUI to track key facial features and interpret subtle eye and mouth gestures into mouse commands. In particular, the horizontal and vertical position of the right eye controls cursor movement; right and left eye winks trigger right and left clicks, respectively; the vertical difference between the eyes controls scrolling; and an open mouth toggles mouse control on and off. Experimental results demonstrate that the system performs robustly in real-time scenarios.

1. Introduction

Traditional input devices such as a mouse and keyboard can be limiting in hands-free environments or for users with motor impairments. Recent advances in computer vision have paved the way for alternative interaction methods using facial and eye movements. In this paper, we introduce an eye-controlled mouse system that uses face mesh detection to map specific eye and mouth gestures to mouse functions, thereby providing an intuitive, touch-free control mechanism.

2. Proposed Method

The proposed system consists of the following main components:

1. **Face Mesh Detection:** Using MediaPipe's face mesh solution, the system captures video input from a webcam and detects facial landmarks. Landmarks around the eyes and mouth are used to infer user gestures.

2. **Cursor Movement:** The horizontal and vertical coordinates of the right eye are mapped directly to the screen coordinates to control the cursor position.

3. **Clicking:**
    - **Left Click:** A wink (blink) detected from the left eye triggers a left-click event.
    - **Right Click:** A wink detected from the right eye triggers a right-click event.

4. **Scrolling:** The system compares the vertical positions of the right and left eyes. If the right eye is positioned higher than the left, an upward scroll is performed; if it is lower, a downward scroll is executed.

5. **Mouse Control Toggle:** The system monitors the mouth state. When the mouth is open beyond a predefined threshold, mouse control is toggled on or off.

Table 1. Gesture mapping for mouse control using facial gestures.

Function	Gesture
Right Click	Right eye wink
Left Click	Left eye wink
Scroll Up	Right eye positioned higher than the left eye
Scroll Down	Left eye positioned higher than the right eye
Mouse Toggle	Open mouth to toggle control on/off
Cursor Move	Horizontal and vertical position of the right eye

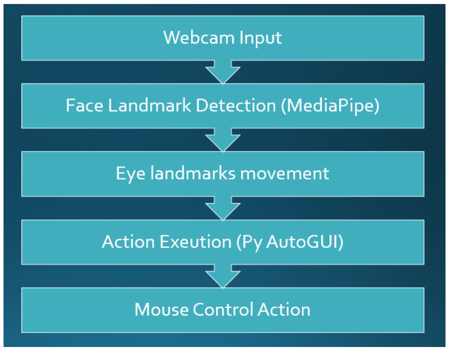
Webcam Input
Face Landmark Detection (MediaPipe)
Eye landmarks movement
Action Exeution (Py AutoGUI)
Mouse Control Action

Figure 1. Execution Steps

3. Experiments and Results

The system was evaluated using a standard webcam under various lighting conditions. The evaluation focused on:

- **Cursor Responsiveness:** Mapping the right eye's position to the screen demonstrated smooth and precise cursor movements.
- **Clicking Accuracy:** Winks from the left and right eyes reliably triggered left and right clicks with appropriate debounce intervals.
- **Scrolling Sensitivity:** A scroll threshold was set to minimize unintended scrolling, and the vertical eye position differential proved effective in controlling the scroll direction.
- **Toggle Control:** The use of mouth openness to toggle the mouse control state allowed the user to easily activate or deactivate the system.

Preliminary tests indicate that the system performs robustly in real-time, making it a promising alternative for hands-free interaction. Future work will focus on improving gesture detection accuracy and extending the system's adaptability to different lighting conditions.

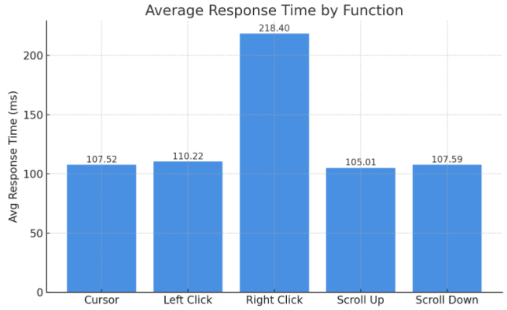

Figure 2. **Bar chart** of average response times per gesture function

Table 2. Average Response Time of Different Gesture Functions

Function	Avg. Response Time (ms)
Cursor	107.52
Left Click	110.22
Right Click	217.37
Scroll Up	105.01
Scroll Down	107.65

4. Appendix: Implementation Code

Below is the Python code that implements the eye-controlled mouse functionality with the new gesture mappings.

```python
import cv2
import mediapipe as mp
import pyautogui
import time
import sys
# Starting the camera
cam = cv2.VideoCapture(0)
face_mesh =
    mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)
screen_width, screen_height = pyautogui.size()
cv2.namedWindow("Eye Controlled Mouse",
    cv2.WINDOW_NORMAL)
# Measuring response time
gesture_data = {
    "Cursor":      {"attempts": 0, "success": 0,
        "response": 0},
    "Left_Click":  {"attempts": 0, "success": 0,
        "response": 0},
    "Right_Click": {"attempts": 0, "success": 0,
        "response": 0},
    "Scroll_Up":   {"attempts": 0, "success": 0,
        "response": 0},
    "Scroll_Down": {"attempts": 0, "success": 0,
        "response": 0}
}

previous_left_state  = False
previous_right_state = False
last_click_time  = time.time()
last_scroll_time = time.time()
last_stat_print  = time.time()

#values for clicking and exiting

SCROLL_THRESHOLD = 0.05
DEBOUNCE_TIME    = 0.5
MOUTH_OPEN_THRESHOLD = 0.05

#function that updates the stats

def update_stats(gesture, success, start_ts):
    gesture_data[gesture]["attempts"] += 1
    if success:
        gesture_data[gesture]["success"] += 1
        gesture_data[gesture]["response"] +=
            (time.time() - start_ts) * 1000


#printing the stats
def print_stats():
    print("\nGesture Accuracy Report:")
    for key, s in gesture_data.items():
        at, su = s["attempts"], s["success"]
        if at > 0:
            accuracy = su / at * 100
            avereage = (s["response"] / su) if
                su > 0 else 0
            print(f"  {key:12s}  Acc:
                {accuracy:5.1f}% ({su}/{at})
                Avg resp: {avereage:.2f} ms")
        else:
            print(f"  {key:12s}  No attempts
                yet")
    print("-" * 50)

print("Eye Controlled Mouse is active")
print("Open your mouth wide to exit the program")


start_time = time.time()

#Main loop
while True:
    ret, frame = cam.read()
    if not ret:
        break
    frame = cv2.flip(frame, 1)
    rgb   = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result   = face_mesh.process(rgb)
    h, w, _ = frame.shape

    if result.multi_face_landmarks:
        landmarks = result.multi_face_landmarks[0].landmark

        #checks if the mouth is open
        mouth_open = False
        try:
            if (landmarks[14].y - landmarks[13].y) > MOUTH_OPEN_THRESHOLD:
                mouth_open = True
                print("Mouth open detected - exiting program")
                break
        except:
            pass  #error handling for open mouth detection

        for index in [145, 159, 374, 386]:
            cx, cy = int(landmarks[index].x * w), int(landmarks[index].y * h)
            cv2.circle(frame, (cx, cy), 3, (0, 255, 255), -1)


        # Moving cursor with respect to the position of the user’s eye landmarks
        t0 = time.time()
        gaze = landmarks[477]
        screen_x, screen_y = gaze.x * screen_width, gaze.y * screen_height
        pyautogui.moveTo(screen_x, screen_y)
        update_stats("Cursor", True, t0)
        cv2.circle(frame, (int(gaze.x*w), int(gaze.y*h)), 3, (0,255,0), -1)

        # Left click (left eye wink)
        left_closed = (landmarks[145].y - landmarks[159].y) < 0.006
        if left_closed and not previous_left_state and (time.time()-last_click_time)>DEBOUNCE_TIME:
            t0 = time.time()
            pyautogui.click()
            update_stats("Left_Click", True, t0)
            last_click_time = time.time()
        previous_left_state = left_closed

        # Right click (right eye wink)
        right_closed = (landmarks[374].y - landmarks[386].y) < 0.006
        if right_closed and not previous_right_state and (time.time()-last_click_time)>DEBOUNCE_TIME:
            t0 = time.time()
            pyautogui.rightClick()
            update_stats("Right_Click", True, t0)
            last_click_time = time.time()
        previous_right_state = right_closed

        # Scroll up/down
        if (time.time() - last_scroll_time) > DEBOUNCE_TIME:
            tilt = landmarks[374].y - landmarks[145].y
            if abs(tilt) > SCROLL_THRESHOLD:
                t0 = time.time()
                if tilt < 0:
                    pyautogui.scroll(300)
                    update_stats("Scroll_Up", True, t0)
                else:
                    pyautogui.scroll(-300)
                    update_stats("Scroll_Down", True, t0)
                last_scroll_time = time.time()


    cv2.imshow("Eye Controlled Mouse", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord(’q’) or key == 27:  # q or ESC key to exit
        break

    # makes sure that the program prints the stats after an interval of 1 min
    if (time.time() - last_stat_print) > 60:
        print_stats()
        last_stat_print = time.time()

    # Alternative exit method - if run for 5 minutes, automatically exit
    if time.time() - start_time > 300:  # 5 minutes = 300 seconds
        print("Session time limit reached - exiting")
        break
print("\nFinal Accuracy Report:")
print_stats()
cam.release()
cv2.destroyAllWindows()
```

Listing 1. Python Code for Eye Controlled Mouse with Gesture Toggle

5. References

1. https://www.linkedin.com/pulse/hands-free-computing-eye-controlled-mouse-using-opencv-pandey-cwhef/

2. https://pyautogui.readthedocs.io/

3. https://app.readytensor.ai/publications/eyecontrolled-mouse-

system-using-realtime-face-mesh-detection-h2mArbWhraSU