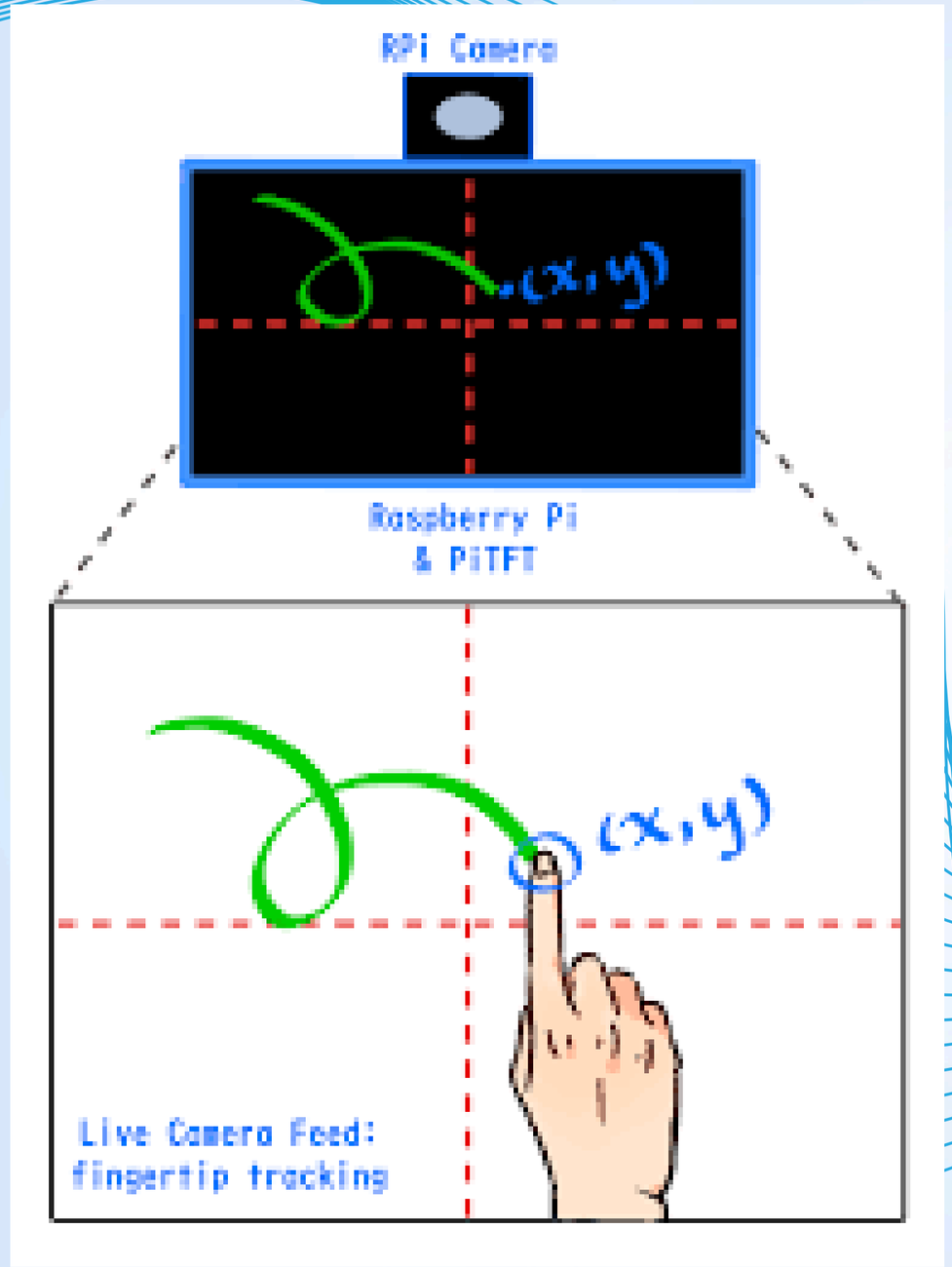


AIR CANVAS USING COMPUTER VISION

INTRODUCTION

What is Air Canvas?

- Air Canvas is an interactive virtual drawing application that allows users to draw in the air using hand gestures, without any physical contact.
- It leverages computer vision techniques to detect hand movements and translate them into digital strokes on a virtual canvas.

How Does It Work?

- Uses OpenCV and MediaPipe to track hand gestures in real-time.
- Recognizes finger positions and converts movements into digital drawings.
- Enables contactless interaction, enhancing user experience in artistic applications.

EXISTING WORK WITH LIMITATION

1. Basic Webcam Drawing Tools
   - Use color markers or fingertip
   - Tracking via OpenCV

   Limitations:
   - Low accuracy in different lighting
   - Limited gestures
   - Laggy and unsmooth drawing

2. MediaPipe Hand Tracking
   - Tracks hands/fingers with higher precision

   Limitations:
   - Struggles with fast movements
   - High CPU usage on low-end devices
   - Limited gesture recognition

EXISTING WORK WITH LIMITATION

3. AR-based Drawing Applications
   - Use mobile AR (ARCore/ARKit) to draw in 3D space

   Limitations:
   - Requires specific hardware
   - Less desktop-friendly
   - Not open-source/flexible

4. ML-based Gesture Recognition
   - Machine learning models classify gestures for drawing actions

   Limitations:
   - Needs large datasets
   - More complex implementation
   - Slower processing

PROPOSED WORK AND METHODOLOGY

- Video Capture: Use OpenCV to access the webcam and mirror the feed.
- Hand Tracking: Apply MediaPipe to detect and track hand landmarks in real-time.
- Gesture Recognition: Identify gestures (e.g., index finger up for drawing) based on landmark positions.

PROPOSED WORK AND METHODOLOGY

- Drawing Logic: Use NumPy and OpenCV to draw on a virtual canvas by tracking finger movement.
- UI Controls: Integrate PyAutoGUI for gesture-based screen actions (e.g., clear or save).
- Optimization: Smooth drawings and enhance tracking stability.
- Output: Display the final canvas with options to save, clear, or exit.

NOVELTY OF PROJECT

- Enables contactless drawing using just hand gestures, promoting hygiene and accessibility.
- Utilizes real-time hand tracking with MediaPipe for precise and responsive interaction.
- Combines gesture recognition with virtual canvas, offering a unique human-computer interaction experience.
- Does not require special hardware- runs on any device with a webcam.
- Offers potential for creative applications like digital art, education, and interactive presentations.

HARDWARE & SOFTWARE REQUIREMENTS

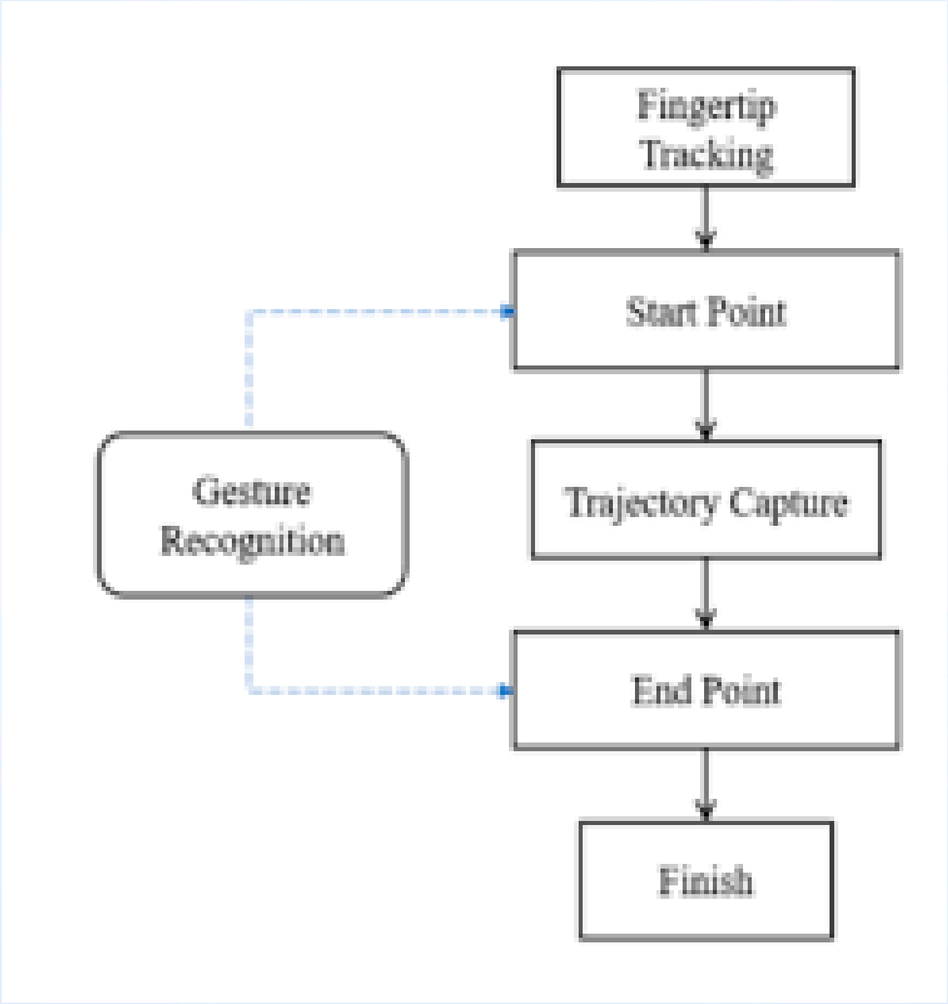
Hardware:

- Camera (Laptop/Webcam)
- Computer with a minimum of 4GB RAM
- Decent processing power (Core i5 and above)

Software:

- Python
- OpenCV, MediaPipe, NumPy, PyAutoGUI
- Jupyter Notebook/PyCharm/VS Code

SYSTEM ARCHITECTURE DIAGRAM



LIST OF MODULES & EXPLANATION

1. Hand Detection Module
   - Uses MediaPipe Hand Tracking to identify finger positions.
   - Detects index finger for drawing and gestures for other actions.

2. Drawing Module
   - Translates finger movements into strokes on the canvas.
   - Enables different colors, thickness, and eraser mode.

3. Interface Interaction Module
   - Uses PyAutoGUI for interactive controls if needed.

4. Save & Clear Module
   - Saves drawings and allows users to clear the screen.

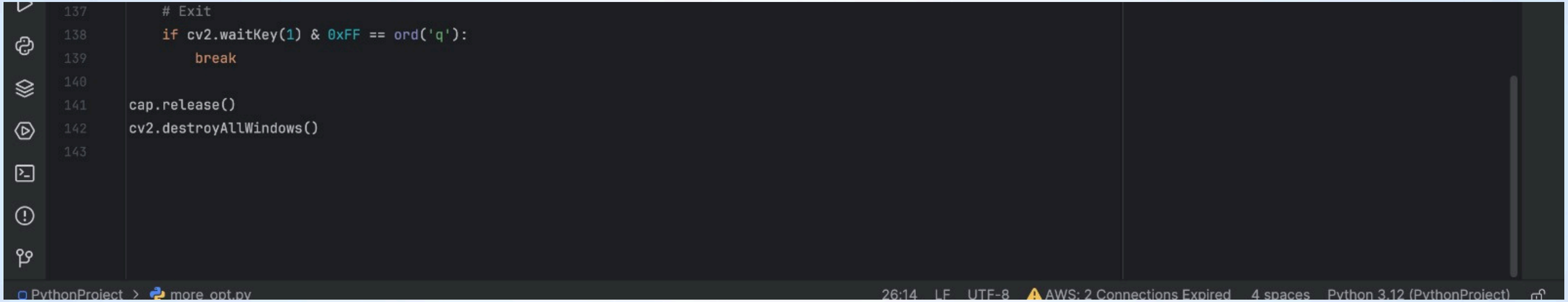
CODE:-

```python
import cv2
import numpy as np
import mediapipe as mp
import pytesseract
import os

# Suppress TensorFlow warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.5)

# Initialize OCR (Tesseract for handwriting recognition)
pytesseract.pytesseract.tesseract_cmd = '/opt/homebrew/bin/tesseract'  # Adjust for your installation

# Colors
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (255, 165, 0), (255, 255, 255)]
current_color = (255, 0, 0)  # Default: Red

# Initialize Canvas
canvas = None
prev_x, prev_y = None, None
expression = ""

# Open Camera
cap = cv2.VideoCapture(0)


def is_index_finger_extended(landmarks, w, h):  1 usage
    """
    Optimized check to ensure only the index finger is extended.
    The index finger should be significantly higher than its DIP joint,
    and all other fingers should be sufficiently bent.
    """
    index_tip = landmarks.landmark[8]
    index_dip = landmarks.landmark[6]
```

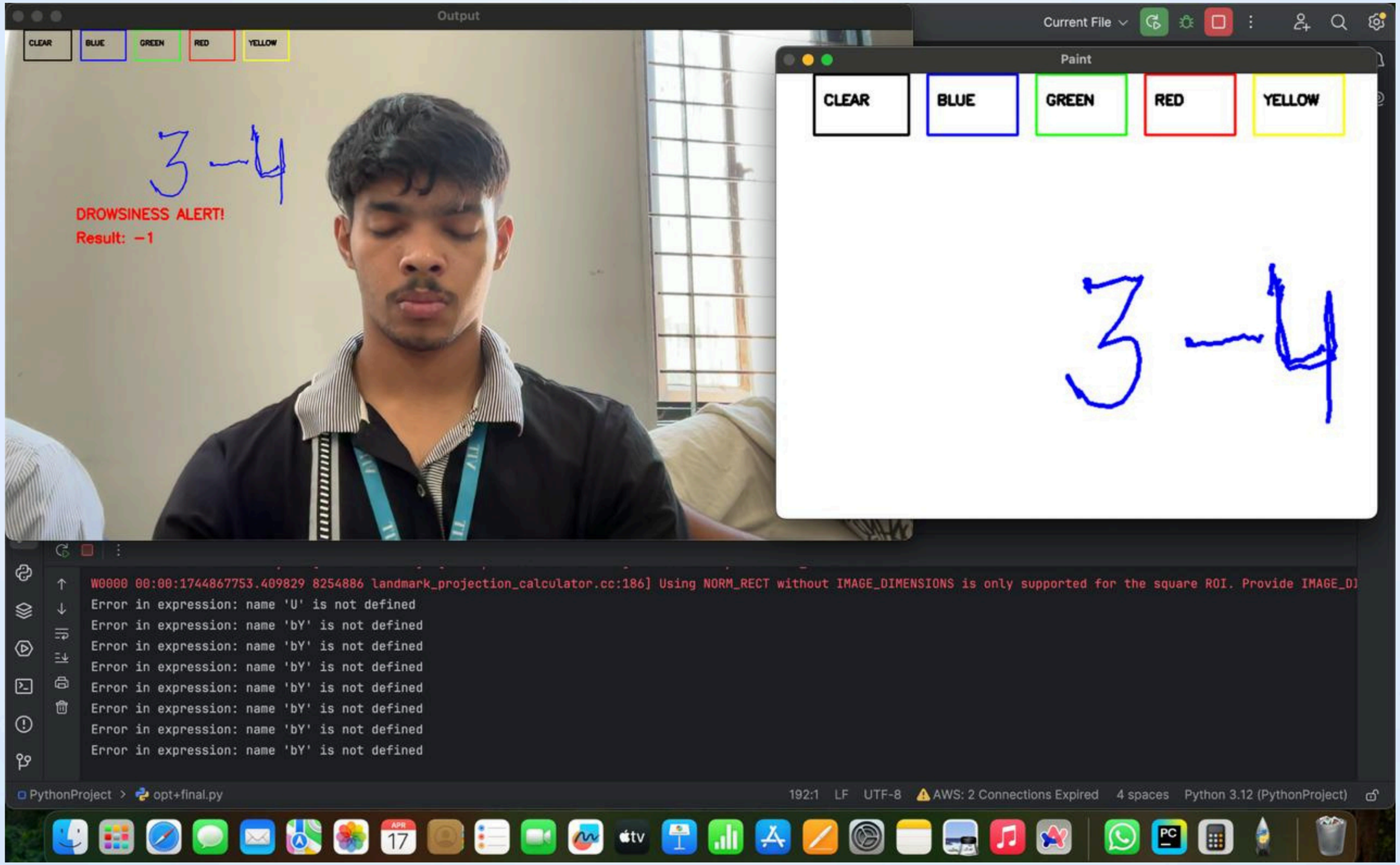
```python
    index_tip = landmarks.landmark[8]
    index_dip = landmarks.landmark[6]
    middle_tip = landmarks.landmark[12]
    middle_pip = landmarks.landmark[10]
    ring_tip = landmarks.landmark[16]
    ring_pip = landmarks.landmark[14]
    pinky_tip = landmarks.landmark[20]
    pinky_pip = landmarks.landmark[18]

    # Convert to pixel values
    index_y = int(index_tip.y * h)
    index_dip_y = int(index_dip.y * h)
    middle_y = int(middle_tip.y * h)
    middle_pip_y = int(middle_pip.y * h)
    ring_y = int(ring_tip.y * h)
    ring_pip_y = int(ring_pip.y * h)
    pinky_y = int(pinky_tip.y * h)
    pinky_pip_y = int(pinky_pip.y * h)

    # Index finger should be extended with a good margin
    index_extended = index_y < index_dip_y - 10

    # Other fingers should be bent (tip should be below the PIP joint)
    others_bent = (
        middle_y > middle_pip_y + 10 and
        ring_y > ring_pip_y + 10 and
        pinky_y > pinky_pip_y + 10
    )

    return index_extended and others_bent


while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
```

```python
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, flipCode: 1)
    h, w, _ = frame.shape
    if canvas is None:
        canvas = np.zeros( shape: (h, w, 3), dtype=np.uint8)

    # Convert to RGB for MediaPipe
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    hand_results = hands.process(rgb_frame)

    # Detect Hand for Drawing
    if hand_results.multi_hand_landmarks:
        for hand_landmarks in hand_results.multi_hand_landmarks:
            index_x, index_y = int(hand_landmarks.landmark[8].x * w), int(hand_landmarks.landmark[8].y * h)

            if is_index_finger_extended(hand_landmarks, w, h):
                # Check if selecting color (top region)
                if index_y < 50:
                    for i, color in enumerate(colors):
                        if index_x in range(i * 50, (i + 1) * 50):
                            current_color = color

                    # Check if touching "CLEAR" button (Last rectangle)
                    if index_x in range(len(colors) * 50, (len(colors) + 1) * 50):
                        canvas = np.zeros( shape: (h, w, 3), dtype=np.uint8)  # Reset Canvas
                else:
                    if prev_x is None or prev_y is None:
                        prev_x, prev_y = index_x, index_y
                    cv2.line(canvas, pt1: (prev_x, prev_y), pt2: (index_x, index_y), current_color, thickness: 5)
                    prev_x, prev_y = index_x, index_y
            else:
                prev_x, prev_y = None, None
```

```python

    # Overlay Drawing on Frame
    frame = cv2.addWeighted(frame, alpha: 1, canvas, beta: 0.5, gamma: 0)

    # Draw Color Palette
    for i, color in enumerate(colors):
        cv2.rectangle(frame, (i * 50, 0), ((i + 1) * 50, 50), color, -1)

    # Draw CLEAR button
    clear_x_start = len(colors) * 50
    clear_x_end = (len(colors) + 1) * 50
    cv2.rectangle(frame, (clear_x_start, 0), (clear_x_end, 50), (255, 255, 255), -1)  # White box
    cv2.putText(frame, text: "CLEAR", org: (clear_x_start + 5, 30), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.5, color: (0, 0, 0), thickness: 2)

    # Convert Canvas to Grayscale and Extract Expression
    if cv2.waitKey(1) & 0xFF == ord('e'):  # Press 'E' to evaluate expression
        gray_canvas = cv2.cvtColor(canvas, cv2.COLOR_BGR2GRAY)
        _, binary_canvas = cv2.threshold(gray_canvas, thresh: 127, maxval: 255, cv2.THRESH_BINARY_INV)
        expression = pytesseract.image_to_string(binary_canvas, config='--psm 7 digits')

        try:
            result = str(eval(expression.strip()))  # Evaluate the extracted expression
        except:
            result = "Invalid Expression"

        expression = f"{expression.strip()} = {result}"

    # Show Expression Result
    cv2.putText(frame, expression, org: (50, h - 50), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 255, 255), thickness: 2)

    # Display Output
    cv2.imshow( winname: "AI-Powered AirCanvas - Write & Solve", frame)

    # Exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```


137 # Exit
138 if cv2.waitKey(1) & 0xFF == ord('q'):
139 break
140
141 cap.release()
142 cv2.destroyAllWindows()
143
PythonProject > more_opt.py
26:14 LF UTF-8 AWS: 2 Connections Expired 4 spaces Python 3.12 (PythonProject)

RESULT AND DISCUSSION



RESULT AND DISCUSSION

• Hand Tracking: Achieved accurate real-time tracking using MediaPipe (~20-30 FPS).

• Drawing via Gestures: Users drew on a virtual canvas using finger gestures (e.g., index up = draw, index+middle =

select).

• Tool & Color Selection:

Gesture-based switching between pen, eraser, and colors worked effectively.

• Canvas Persistence: Drawings remained intact across frames using layered mask logic.

RESULT AND DISCUSSION

- Performance: Smooth operation in well-lit environments; accuracy drops in low-light or cluttered backgrounds. User Feedback: Interactive and intuitive, though a brief learning curve was noted.

- Limitations: Single-user only, needs stable lighting and camera position.

- Future Scope: Add image saving, improve gesture recognition, and include on-screen UI elements.

CONCLUSION

The Air Canvas project successfully demonstrated real-time, touchless drawing using hand gestures. It offers an intuitive and interactive interface, showcasing the potential of computer vision in human-computer interaction. With further improvements, it can be developed into a practical tool for creative and educational applications.

THANK YOU