SMART CLASSROOM

IOT + ML PROJECT

Madhav Choudhary

- LinkedIn - Github

PROJECT FILES

DEMO VIDEO

TABLE OF CONTENTS

1. Introduction

The concept of a Smart Classroom revolutionizes traditional learning spaces by integrating advanced technologies to create a dynamic, engaging, and interactive environment. The **Smart Classroom System** incorporates both hardware and software components, enhancing the educational experience for students and educators alike. With features like **drowsiness detection, virtual input devices, emotion recognition, and proctored exams**, this system addresses contemporary challenges in education while promoting interactivity and personalization.

This project aims to transform the classroom into a **smart, secure, and connected environment** by utilizing cutting-edge technologies, including **CNNs (Convolutional Neural Networks), Python, Flask, IoT sensors, and facial recognition algorithms**.

---

2. Features of the Project

The Smart Classroom project incorporates a robust set of features, each addressing a specific challenge faced in modern educational settings. Here is a detailed exploration of the seven core features:

2.1 Drowsiness Detection

- **Description :**

  Drowsiness detection is an essential feature for monitoring student attentiveness during lessons. It utilizes advanced **computer vision algorithms and physiological analysis** to detect signs of fatigue, such as prolonged eye closure, yawning, and head tilting.

- **Technology Used:**

  - **MediaPipe Framework**: Tracks eye movements and facial landmarks.
  - **Python Libraries**: OpenCV for real-time video processing.

- **Benefits:**

  - Identifies early signs of fatigue to prevent classroom accidents or inattentiveness
  - Alerts teachers in real-time, allowing them to re-engage students.

2.2 Virtual Keyboard

- **Description:**

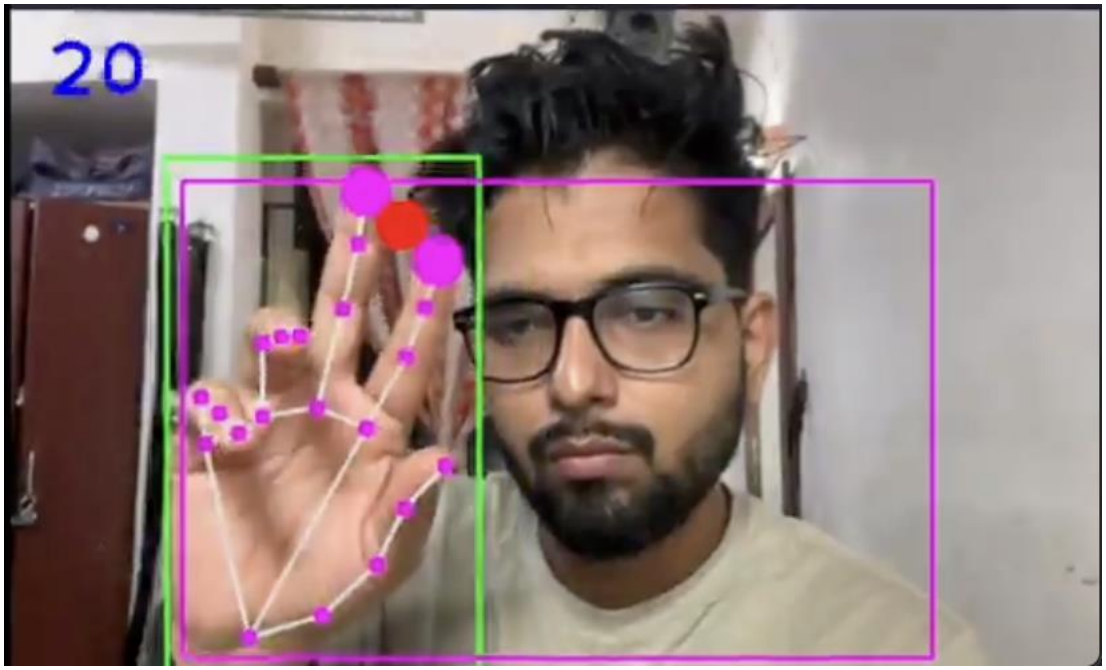
  The Virtual Keyboard is a **projection-based or gesture-based input device** that eliminates the need for physical hardware. Students and teachers can interact with digital content effortlessly by typing on a virtual interface displayed on any surface.

- **Key Features:**
    - Touchless typing for hygienic operation.
    - Highly responsive gesture recognition for accuracy.
    - Compatibility with multiple languages and layouts.
- **Technology Used:**
    - OpenCV for gesture detection.
    - Python for integration with the classroom system.
- **Advantages:**
    - Reduces hardware dependency, making classrooms more adaptable.
    - Enhances accessibility for users with physical disabilities.

2.3 Virtual Mouse

- **Description:**

  The Virtual Mouse replaces the traditional hardware mouse with a gesture-controlled interface. Students can perform basic operations like clicking, dragging, and scrolling using **hand gestures** tracked by a camera.



- **Technology Used:**
    - **Hand Detection Models**: Tracks finger movements for precise cursor control.

- **Python Frameworks**: For gesture-to-command mapping.

- **Impact on Learning:**

  - Facilitates seamless interaction with digital content.
  - Provides a futuristic, intuitive experience that engages students.

2.4 Proctored Exam System

- **Description:**

  Ensuring academic integrity is critical in both physical and online learning environments. The Proctored Exam System uses **facial recognition, eye tracking, and live monitoring** to detect suspicious behaviors and verify the identity of students.

- **Components:**

  - **Facial Recognition Module**: Confirms student identity.
  - **Behavior Monitoring**: Tracks head movements and attention.

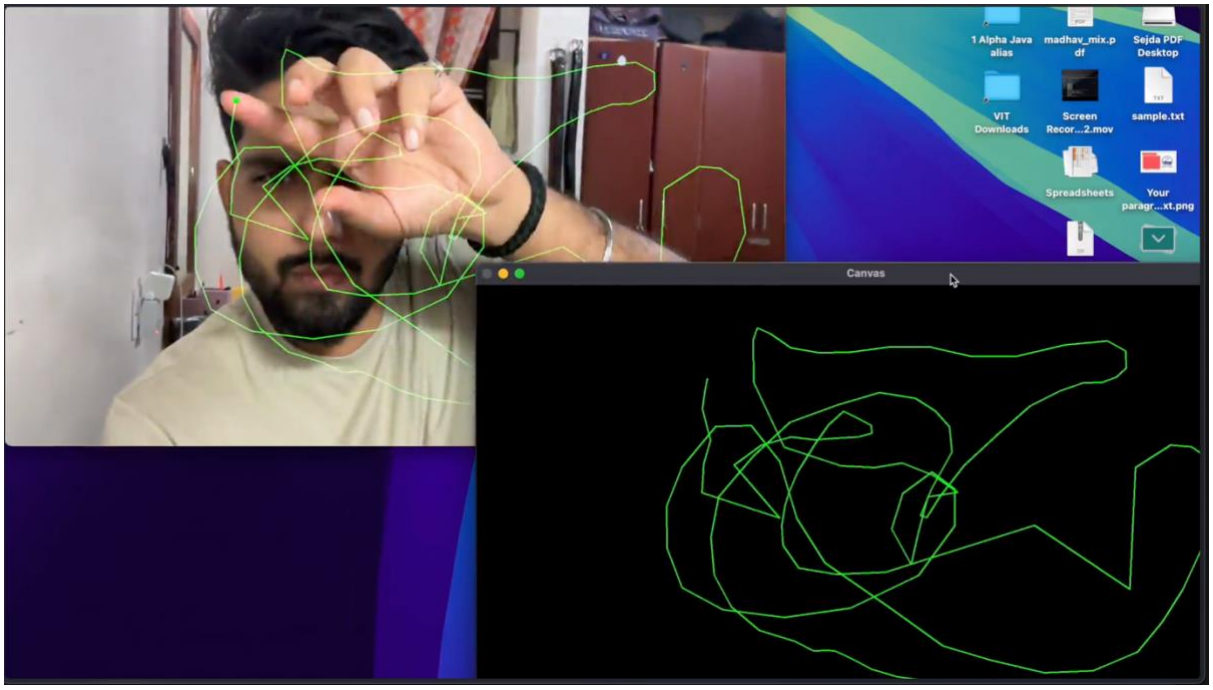
- **Applications:**

  - Conducting secure, automated exams.
  - Real-time monitoring and flagging of irregularities.

2.5 Smartboard

- **Description:**

  The Smartboard is an **interactive display system** designed for dynamic teaching. Teachers can annotate, display multimedia, and interact with the content in real time.



- **Key Functionalities:**

  - Supports drawing, highlighting, and note-taking.
  - Allows students to collaborate directly on the board.

- **Advantages:**

  - Improves understanding through visual aids.
  - Encourages active participation and creativity.

2.6 Emotion Recognition Using CNN

- **Description:**

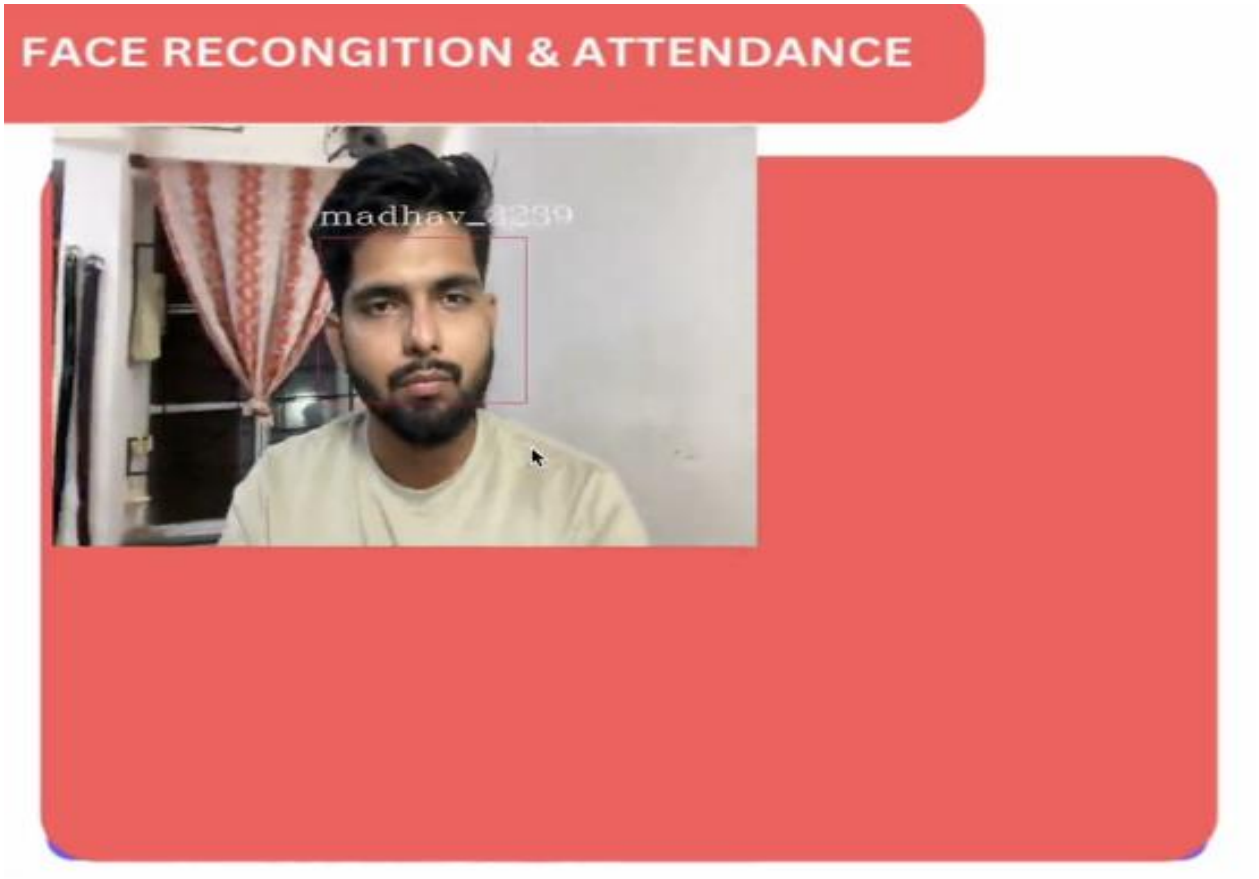
  The Emotion Recognition system uses **deep learning algorithms** to analyze students' facial expressions and determine their emotional states. This provides valuable insights into their engagement and mental well-being.

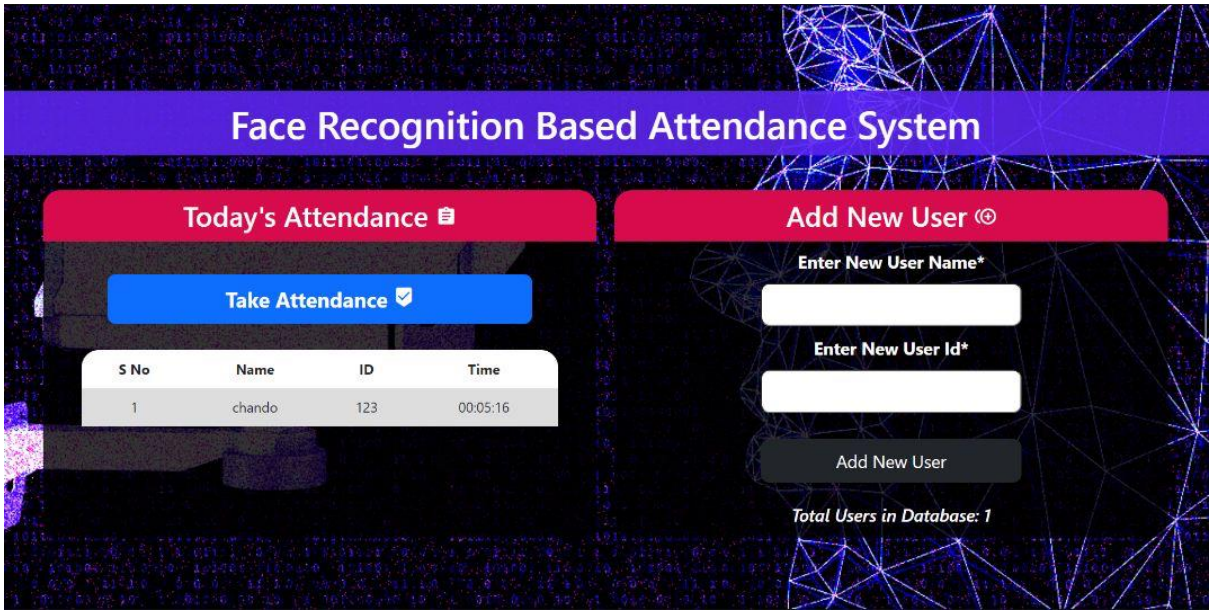
- **Process:**
    - Captures real-time facial data.
    - Analyzes expressions (happy, sad, confused, etc.).
- **Benefits:**
    - Empowers teachers to adapt their teaching styles.
    - Creates a supportive and empathetic classroom environment.

2.7 Face Recognition with Python and Flask

- **Description:**

    This feature automates attendance-taking by identifying students as they enter the classroom using facial recognition.





- **Technology Used:**
    - **Haar Cascades and DLIB**: For face detection and recognition.
    - **Flask Framework**: To build the backend for data storage and analysis.
- **Advantages:**

- Saves time and reduces manual errors.
- Enhances accountability and security.

---

3. Project Structure

The project architecture integrates the following components:

3.1 Hardware Elements

- **IoT Sensors:** For motion detection and automated lighting.
- **Cameras:** For face recognition, emotion detection, and drowsiness monitoring.
- **Projection Devices:** For virtual keyboards and smartboard interactivity.

3.2 Software Framework

3.2.1. Backend:

- Flask for building the server-side logic.
- Integration with AI/ML models for recognition tasks.

3.2.2. Frontend:

- HTML, CSS, and JavaScript for user interfaces.
- Streamlit for emotion recognition and visualization.

3.3 Machine Learning Models

- **CNN for Emotion Recognition:** Detects various facial expressions (happy, sad, angry, etc.).
- **Drowsiness Detection Model:** Tracks blink rate and eye movements using MediaPipe.

---

4. Scope of Future Enhancements

1. **Integration with Learning Management Systems (LMS):** Seamlessly connect with LMS platforms like Moodle or Blackboard.

2. **AI-Powered Analytics:** Provide detailed reports on student engagement and performance.

3. **Voice-Controlled Interactions:** Enable voice commands for classroom devices.

4. **Adaptive Learning Modules:** Automatically adjust teaching strategies based on real-time analytics.

5. **Enhanced Proctoring:** Use multi-angle camera setups and behavioral tracking for robust examination security.

---

5. Steps Followed

1. **Requirement Analysis:**

   - Identify classroom challenges.
   - Define technological needs for automation and interactivity.

2. **System Design:**

   - Create architecture diagrams and flowcharts.
   - Specify hardware and software integrations.

3. **Implementation:**

   - Develop individual components (e.g., emotion detection model, face recognition module).
   - Integrate them into a unified system.

4. **Testing and Debugging:**

   - Validate the functionality of each feature.
   - Test the system in real-world classroom settings.

5. **Deployment:**

   - Install the system in a prototype classroom.
   - Monitor performance and gather feedback.

---

6. Code Section

6.1. Drowsiness Detection

```
import cv2
import mediapipe as mp
import numpy as np
```

```
import time

# Initialize video capture
def initialize_capture():
    camera = cv2.VideoCapture(0)
    if not camera.isOpened():
        print("Error: Couldn't open camera.")
        return None
    return camera

# Detect landmarks using Mediapipe
def detect_landmarks(frame, face_mesh):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(rgb_frame)
    if results.multi_face_landmarks:
        mesh_coords = [(int(point.x * frame.shape[1]), int(point.y * frame.shape[0])) for point in
results.multi_face_landmarks[0].landmark]
        return mesh_coords
    return None

# Calculate blink ratio
def calculate_blink_ratio(mesh_coords, right_eye_indices, left_eye_indices):
    # Calculate distances for right eye
    rh_right = mesh_coords[right_eye_indices[0]]
    rh_left = mesh_coords[right_eye_indices[8]]
    rv_top = mesh_coords[right_eye_indices[12]]
    rv_bottom = mesh_coords[right_eye_indices[4]]
    rh_distance = euclidean_distance(rh_right, rh_left)
    rv_distance = euclidean_distance(rv_top, rv_bottom)
    re_ratio = rh_distance / rv_distance

    # Calculate distances for left eye
    lh_right = mesh_coords[left_eye_indices[0]]
    lh_left = mesh_coords[left_eye_indices[8]]
    lv_top = mesh_coords[left_eye_indices[12]]
    lv_bottom = mesh_coords[left_eye_indices[4]]
    lh_distance = euclidean_distance(lh_right, lh_left)
    lv_distance = euclidean_distance(lv_top, lv_bottom)
    le_ratio = lh_distance / lv_distance

    return (re_ratio + le_ratio) / 2

# Extract eyes from the frame
def extract_eyes(frame, right_eye_coords, left_eye_coords):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    mask = np.zeros_like(gray)
    cv2.fillPoly(mask, [np.array(right_eye_coords, dtype=np.int32)], 255)
    cv2.fillPoly(mask, [np.array(left_eye_coords, dtype=np.int32)], 255)
    eyes = cv2.bitwise_and(gray, gray, mask=mask)
    eyes[mask == 0] = 155
```

```
    r_min_x = min(right_eye_coords, key=lambda item: item[0])[0]
    r_max_x = max(right_eye_coords, key=lambda item: item[0])[0]
    r_min_y = min(right_eye_coords, key=lambda item: item[1])[1]
    r_max_y = max(right_eye_coords, key=lambda item: item[1])[1]
    l_min_x = min(left_eye_coords, key=lambda item: item[0])[0]
    l_max_x = max(left_eye_coords, key=lambda item: item[0])[0]
    l_min_y = min(left_eye_coords, key=lambda item: item[1])[1]
    l_max_y = max(left_eye_coords, key=lambda item: item[1])[1]

    cropped_right = eyes[r_min_y:r_max_y, r_min_x:r_max_x]
    cropped_left = eyes[l_min_y:l_max_y, l_min_x:l_max_x]

    return cropped_right, cropped_left

# Estimate eye position
def estimate_eye_position(cropped_eye):
    h, w = cropped_eye.shape
    blurred_eye = cv2.medianBlur(cropped_eye, 3)
    _, threshed_eye = cv2.threshold(blurred_eye, 130, 255, cv2.THRESH_BINARY)
    piece = int(w / 3)
    right_piece = threshed_eye[:, :piece]
    center_piece = threshed_eye[:, piece:2 * piece]
    left_piece = threshed_eye[:, 2 * piece:]
    return pixel_counter(right_piece, center_piece, left_piece)

# Euclidean distance
def euclidean_distance(point1, point2):
    x1, y1 = point1
    x2, y2 = point2
    return np.sqrt((x2 - x1)**2 + (y2 - y1)**2)

# Pixel counter function
def pixel_counter(first_piece, second_piece, third_piece):
    right_part = np.sum(first_piece == 0)
    center_part = np.sum(second_piece == 0)
    left_part = np.sum(third_piece == 0)
    eye_parts = [right_part, center_part, left_part]
    max_index = eye_parts.index(max(eye_parts))
    if max_index == 0:
        return "RIGHT", (0, 255, 0)  # Right eye
    elif max_index == 1:
        return "CENTER", (255, 255, 0)  # Center eye
    else:
        return "LEFT", (0, 0, 255)  # Left eye

# Main function
def main():
    # Load Mediapipe face mesh model
    face_mesh = mp.solutions.face_mesh.FaceMesh(min_detection_confidence=0.5, min_tracking_confidence=0.5)

    # Initialize video capture
```

```
camera = initialize_capture()
if camera is None:
    return

frame_counter = 0
total_blinks = 0
cef_counter = 0
display_drowsiness = False
display_frames = 0  # Counter to keep track of frames for displaying drowsiness

# Define eye indices
RIGHT_EYE = [33, 7, 163, 144, 145, 153, 154, 155, 133, 173, 157, 158, 159, 160, 161, 246]
LEFT_EYE = [362, 382, 381, 380, 374, 373, 390, 249, 263, 466, 388, 387, 386, 385, 384, 398]

# Main loop
while True:
    ret, frame = camera.read()
    if not ret:
        break

    frame_counter += 1

    mesh_coords = detect_landmarks(frame, face_mesh)
    if mesh_coords:
        blink_ratio = calculate_blink_ratio(mesh_coords, RIGHT_EYE, LEFT_EYE)

        if blink_ratio > 5.5:
            cef_counter += 1
            if cef_counter > 6:
                display_drowsiness = True
                display_frames = 0
        else:
            if cef_counter > 3:
                total_blinks += 1
                cef_counter = 0

        cv2.putText(frame, f'Total Blinks: {total_blinks}', (30, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        right_coords = [mesh_coords[p] for p in RIGHT_EYE]
        left_coords = [mesh_coords[p] for p in LEFT_EYE]
        crop_right, crop_left = extract_eyes(frame, right_coords, left_coords)
        eye_position_right, color_right = estimate_eye_position(crop_right)
        eye_position_left, color_left = estimate_eye_position(crop_left)

        cv2.polylines(frame, [np.array(right_coords, dtype=np.int32)], True, (0, 255, 0), 1)
        cv2.polylines(frame, [np.array(left_coords, dtype=np.int32)], True, (0, 255, 0), 1)

        cv2.putText(frame, f'R: {eye_position_right}', (40, 220), cv2.FONT_HERSHEY_SIMPLEX, 1, color_right, 2)
        cv2.putText(frame, f'L: {eye_position_left}', (40, 320), cv2.FONT_HERSHEY_SIMPLEX, 1, color_left, 2)

        if display_drowsiness:
```

```
            cv2.putText(frame, "Drowsiness Detected", (10, 60), cv2.FONT_HERSHEY_SIMPLEX,  1, (0, 0, 255), 2)
            display_frames += 1
            if display_frames > 30:  # Display for 30 frames (1 second at 30fps)
                display_drowsiness = False

        cv2.imshow('frame', frame)

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

    camera.release()
    cv2.destroyAllWindows()

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

6.1.1. Code Explanation / Analysis

The file drowsiness_detection.py implements a drowsiness detection system using Mediapipe's face mesh, OpenCV for video capture and display, and NumPy for numerical computations. Below is a detailed breakdown:

Section 1 : Initialize Video Capture

```
def initialize_capture():
    camera = cv2.VideoCapture(0)
    if not camera.isOpened():
        print("Error: Couldn't open camera.")
        return None
    return camera
```

- **Purpose**: This function initializes the video capture from the primary camera (index 0).
- **Key Highlights**:
    - cv2.VideoCapture(0): Starts the camera for real-time video feed.
    - **Error Handling**: If the camera fails to open, an error message is displayed.

Section 2 : Detect Landmarks Using Mediapipe

```
def detect_landmarks(frame, face_mesh):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(rgb_frame)
    if results.multi_face_landmarks:
```

```
    mesh_coords = [(int(point.x * frame.shape[1]), int(point.y * frame.shape[0])) for point in results.multi_face_landmarks[0].landmark]
    return mesh_coords
  return None
```

- **Purpose**: Detects facial landmarks on a video frame.
- **Key Highlights**:
    - cv2.cvtColor: Converts the frame to RGB (required by Mediapipe).
    - face_mesh.process(): Analyzes the frame to find face landmarks.
    - **Output**: Returns pixel coordinates of facial landmarks.

Section 3 : Calculate Blink Ratio

```
def calculate_blink_ratio(mesh_coords, right_eye_indices, left_eye_indices):
  ...
```

- **Purpose**: Computes the blink ratio for each eye based on distances between vertical and horizontal landmarks.
- **Key Highlights**:
    - euclidean_distance: Calculates distances between points to determine eye aspect ratio (EAR).
    - **Output**: The average ratio of both eyes, which indicates whether the eyes are blinking.

Section 4 : Extract Eye Regions

```
def extract_eyes(frame, right_eye_coords, left_eye_coords):
  ...
```

- **Purpose**: Crops the regions containing the right and left eyes.
- **Key Highlights**:
    - cv2.fillPoly: Masks the image to isolate eye regions.
    - **Output**: Cropped images of the right and left eyes for further analysis.

Section 5 : Estimate Eye Position

```
def estimate_eye_position(cropped_eye):
    ...
```

- **Purpose**: Determines the eye's gaze direction (e.g., CENTER, LEFT, or RIGHT).

- **Key Highlights**:
    - cv2.threshold: Converts the eye region into a binary image.
    - **Pixel Counting**: Analyzes dark pixel distribution to estimate gaze.

Section 6 : Main Function

```
def main():
    ...
```

- **Purpose**: Ties all the components together in a loop to process video feed.
- **Key Highlights**:
    - **Eye Blink Detection**: Uses blink ratios to count blinks and detect prolonged closure for drowsiness.
    - **Real-Time Visualization**: Displays the video feed with annotations for gaze direction and drowsiness warnings.

6.1.2. Frameworks and Libraries

1. OpenCV (cv2)

- **Definition**: A popular computer vision library for image and video processing.
- **Usage**:
    - **Real-Time Video Capture**: cv2.VideoCapture, cv2.imshow.
    - **Image Manipulation**: cv2.cvtColor, cv2.threshold.
- **Reason**: Provides robust tools for handling video feeds and visual outputs.

2. Mediapipe

- **Definition**: A framework by Google for real-time ML-based applications.
- **Usage**:
    - **Face Mesh**: mp.solutions.face_mesh.FaceMesh.
- **Reason**: Provides pre-trained face landmark detection, saving development effort.

3. NumPy

- **Definition**: A library for numerical computations in Python.
- **Usage**:
  - **Geometric Calculations**: np.sqrt, np.sum.
  - **Array Operations**: Masking and cropping.
- **Reason**: Efficient handling of array-based computations for geometric operations.

6.1.3. How the Code Works

1. **Video Initialization**: Captures video frames using OpenCV.
2. **Face Landmarks Detection**: Uses Mediapipe to locate facial landmarks, including eyes.
3. **Eye Blink Analysis**: Calculates the blink ratio using landmark distances.
4. **Gaze Detection**: Analyzes cropped eye regions to determine gaze direction.
5. **Drowsiness Alert**: Monitors blinks and prolonged closure to warn of drowsiness.
6. **Real-Time Output**: Displays video feed with overlayed information.

6.1.4. Final Summary

The script integrates Mediapipe and OpenCV to detect drowsiness and eye movement in real-time. The Mediapipe face mesh is leveraged to track eye landmarks, while custom logic computes blink ratios and gaze direction. The code effectively identifies prolonged eye closure, signaling drowsiness, with real-time annotations displayed on the video feed.

6.2. Emotion Recognition

Code :

```
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
```

```python
import cv2
import numpy as np

face_classifier = cv2.CascadeClassifier(r'/Users/madhavchoudhary/Desktop/work/projects_for_resume/6_Smart-classroom_-main/Emotion recognition/haarcascade_frontalface_default.xml')
classifier = load_model(r'/Users/madhavchoudhary/Desktop/work/projects_for_resume/6_Smart-classroom_-main/Emotion recognition/model.h5')

emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray)

    total_faces = 0
    emotion_count = {label: 0 for label in emotion_labels}

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 255), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)

        if np.sum([roi_gray]) != 0:
            roi = roi_gray.astype('float') / 255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi, axis=0)

            prediction = classifier.predict(roi)[0]
            label = emotion_labels[prediction.argmax()]
            emotion_count[label] += 1
            total_faces += 1
            label_position = (x, y)
            cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        else:
            cv2.putText(frame, 'No Faces', (30, 80), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    for i, label in enumerate(emotion_labels):
        if total_faces != 0:
            percentage = (emotion_count[label] / total_faces) * 100
            text = f"{label}: {percentage:.2f}%"
            cv2.putText(frame, text, (10, 30 + 30 * i), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

    cv2.imshow('Emotion Detector', frame)

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

```
cap.release()
cv2.destroyAllWindows()
```

6.2.1. Code Explanation (Line-by-Line)

1. Import Statements:

```
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
import cv2
import numpy as np
```

- **keras.models.load_model**: Loads a pre-trained neural network for emotion recognition.
- **time.sleep**: Can pause execution for a specified time.
- **keras.preprocessing.image**: Helps preprocess images (e.g., conversion to arrays).
- **cv2 (OpenCV)**: Used for real-time image capture and processing.
- **numpy**: Handles numerical operations and arrays.

2. Load Pre-trained Models:

```
face_classifier = cv2.CascadeClassifier('path_to_haarcascade.xml')
classifier = load_model('path_to_model.h5')
```

- **CascadeClassifier**: A Haar cascade model for face detection.
- **load_model**: Loads the trained deep learning model for emotion classification.

3. Define Emotion Labels:

```
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']
```

- Maps the output of the emotion classification model to human-readable labels.

4. Initialize Webcam:

```
cap = cv2.VideoCapture(0)
```

- Opens the default webcam for video capture.

5. Main Loop:

```
while True:
    _, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray)
```

6. Emotion Detection and Labeling:

```
for (x, y, w, h) in faces:
  cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 255), 2)
  roi_gray = gray[y:y+h, x:x+w]
  roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)
```

- Iterates through detected faces, draws rectangles, and extracts the region of interest (ROI) for emotion classification.
- Resizes the ROI to 48x48 (required input size for the emotion model).

7. Model Prediction:

```
if np.sum([roi_gray]) != 0:
    roi = roi_gray.astype('float') / 255.0
    roi = img_to_array(roi)
    roi = np.expand_dims(roi, axis=0)
    prediction = classifier.predict(roi)[0]
    label = emotion_labels[prediction.argmax()]
```

- Preprocesses the ROI: Normalizes pixel values, converts to an array, and adds a batch dimension.
- Predicts the emotion probabilities using the classifier and determines the label with the highest probability.

8. Display Emotion Labels:

```
cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
```

- Annotates the video feed with the predicted emotion for each detected face.

9. Percentage Distribution:

```
for i, label in enumerate(emotion_labels):
    if total_faces != 0:
        percentage = (emotion_count[label] / total_faces) * 100
        text = f"{label}: {percentage:.2f}%"
        cv2.putText(frame, text, (10, 30 + 30 * i), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
```

- Calculates the percentage of detected emotions for all faces in the frame.
- Displays the emotion distribution on the screen.

10. Exit Condition:

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

- Allows the user to terminate the program by pressing the 'q' key.

11. Release Resources:

```
cap.release()
cv2.destroyAllWindows()
```

- Releases the webcam and closes all OpenCV windows.

6.2.2. Key Highlights

1. **Face Detection**:
    - Uses a Haar cascade model to identify faces in each video frame.
2. **Emotion Recognition**:
    - Classifies emotions like "Angry," "Happy," and "Surprise" using a pre-trained neural network.
3. **Real-Time Annotations**:
    - Draws bounding boxes around faces and annotates them with predicted emotions.
4. **Emotion Statistics**:
    - Displays the percentage of detected emotions for all faces in the frame.

6.2.3. Frameworks and Libraries

1. **Keras**:
    - **Definition**: A deep learning framework for building and deploying neural networks.
    - **Usage**: Loads and applies the pre-trained emotion recognition model.
    - **Code Reference**:

```
from keras.models import load_model
classifier = load_model('path_to_model.h5')
prediction = classifier.predict(roi)[0]
```

2. OpenCV:

- **Definition**: A library for real-time computer vision tasks.
- **Usage**: Captures video, detects faces, and overlays annotations.
- **Code Reference**:

```
face_classifier = cv2.CascadeClassifier('path_to_haarcascade.xml')
cap = cv2.VideoCapture(0)
cv2.rectangle(frame, ...)
cv2.putText(frame, ...)
```

3. NumPy:

- **Definition**: A library for numerical and array operations.
- **Usage**: Prepares image data for model prediction and performs calculations for emotion distribution.
- **Code Reference**:

```
roi = roi_gray.astype('float') / 255.0
roi = np.expand_dims(roi, axis=0)
```

6.2.4. How the Code Works

1. **Face Detection**:
    - The Haar cascade detects faces in each video frame.
2. **Emotion Classification**:
    - Each detected face is preprocessed and passed to the pre-trained model to predict emotions.
3. **Annotation**:
    - The script annotates the frame with bounding boxes and predicted emotions.
4. **Statistical Feedback**:
    - Calculates and displays the percentage distribution of emotions across all detected faces.
5. **Real-Time Output**:
    - The annotated video feed is shown in real-time until the user quits.

File Summary

This script effectively integrates computer vision and deep learning for a practical real-time emotion detection system. It demonstrates how to leverage pre-trained models and OpenCV for impactful applications. The emotion_detection.py script performs real-time emotion detection using a webcam. It combines face detection with a pre-trained emotion recognition model to classify facial expressions and display them on the video feed.

6.3.Virtual Keyboard

- Tools: OpenCV, Python.
- Description: Gesture-based interaction for typing and navigation.

```
from time import sleep
import cv2 as cv
from cvzone.HandTrackingModule import HandDetector
from pynput.keyboard import Controller
import cvzone

# Loading Video from Webcam
vid_capture=cv.VideoCapture(0)
vid_capture.set(3,1280)
vid_capture.set(4,720)

# Checking to type elsewhere
doType=False

# Creating Keys for the Keyboard
class Button():
    def __init__(self,position,text,size=[70,70]):
        self.position=position
        self.text=text
        self.size=size

# Drawing all the keys on the screen
def keyDrawing(frame,buttonList):
    for button in buttonList:
        rect_x,rect_y=button.position
        width,height=button.size

        cvzone.cornerRect(frame,(button.position[0],button.position[1],button.size[0],button.size[1]),20,rt=0)
        cv.rectangle(frame,button.position,(rect_x+width,rect_y+height),(0,0,180),cv.FILLED)

cv.putText(frame,button.text,(rect_x+15,rect_y+50),fontFace=cv.FONT_HERSHEY_COMPLEX_SMALL,fontScale=3,color=(255,255,255),thickness=2)
    return frame

# To take Virtual Keyboard input outside
keyboard=Controller()
```

```
def keyTyper(button,doType):
    if doType:
        keyboard.press(button)

keys_keyboard=[["Q","W","E","R","T","Y","U","I","O","P"],
               ["A","S","D","F","G","H","J","K","L",";","<--"],
               ["Z","X","C","V","B","N","M",",",".","/"],["_____________"]]

# Text that is generated by the keyboard
text_written=""

# Defining properties of the keys
buttonList=[]
for i in range(len(keys_keyboard)):
    for j,key in enumerate(keys_keyboard[i]):
        if key=="<--":
            buttonList.append(Button([100*j+40,100*i+50],key,[200,70]))
        elif key==keys_keyboard[-1][-1]:
            buttonList.append(Button([100*j+50,100*i+50],key,[800,70]))
        else:
            buttonList.append(Button([100*j+50,100*i+50],key))

# Loading Hand Detection module
detector = HandDetector(staticMode=False, maxHands=2, modelComplexity=1, detectionCon=0.8, minTrackCon=0.5)

while True:

    # Obtaining individual frames
    isTrue,frame=vid_capture.read()
    # Flipping image for better handling of Keyboard by the user
    frame=cv.flip(frame,1)

    ## hands=detector.findHands(frame,draw=False,flipType=False)
    hands,frame=detector.findHands(frame,draw=True,flipType=False)

    frame=keyDrawing(frame,buttonList)

    #In case a hand is detected
    if hands:
        hand1=hands[0]
        lmList=hand1["lmList"]  #Landmark list of 1 hand
        bbox=hand1["bbox"]  #Bounding box of 1 hand

        for button in buttonList:

            # x,y coordinates of the buttons
            rect_x,rect_y=button.position
            width,height=button.size

            # x,y coordinates of index finger
```

```
        finger_x,finger_y=lmList[8][0],lmList[8][1]

        # Index finger is within a button's region
        if rect_x<finger_x<(rect_x+width) and rect_y<finger_y<(rect_y+height):

            # Increase in button size
            cvzone.cornerRect(frame,(button.position[0]-10,button.position[1]-
10,button.size[0]+20,button.size[1]+20),20,rt=0)
            cv.rectangle(frame,(button.position[0]-10,button.position[1]-
10),(rect_x+width+10,rect_y+height+10),(0,0,80),cv.FILLED)

cv.putText(frame,button.text,(rect_x+15,rect_y+50),fontFace=cv.FONT_HERSHEY_COMPLEX_SMALL,fontScale=3,color=(25
5,255,255),thickness=2)

            # Calculation of distance between index and middle finger

finger_length,inf,frame=detector.findDistance((lmList[8][0],lmList[8][1]),(lmList[12][0],lmList[12][1]),frame,color=(255,     0,
0),scale=10)
            print(finger_length)

            # In case of key click
            if finger_length<45:

                # Change in key color
                cv.rectangle(frame,(button.position[0]-10,button.position[1]-
10),(rect_x+width+10,rect_y+height+10),(0,255,0),cv.FILLED)

cv.putText(frame,button.text,(rect_x+15,rect_y+50),fontFace=cv.FONT_HERSHEY_COMPLEX_SMALL,fontScale=3,color=(25
5,255,255),thickness=2)

                # Generation of text
                if button.text=="<--":
                    text_written=text_written[:-1]
                    keyTyper("\b",doType)
                elif button.text==keys_keyboard[-1][-1]:
                    text_written+=" "
                    keyTyper(" ",doType)
                else:
                    text_written+=button.text
                    keyTyper(button.text,doType)

    # Text and Text box creation
    cv.rectangle(frame,(50,450),(1000,550),(153,153,153),cv.FILLED)

cv.putText(frame,text_written,(60,515),fontFace=cv.FONT_HERSHEY_COMPLEX_SMALL,fontScale=3,color=(255,255,255),th
ickness=2)

    # Display of resultant frame
    cv.imshow("Camera Capture",frame)

    # Escape condition
```

```
if cv.waitKey(2)==27:
    break
```

6.3.1. Code Chunk Explanations

Section 1 : Importing Libraries

```
from time import sleep
import cv2 as cv
from cvzone.HandTrackingModule import HandDetector
from pynput.keyboard import Controller
import cvzone
```

- **Purpose**: Import essential libraries for computer vision, hand tracking, virtual keyboard control, and interface handling.
- **Key Libraries**:
  - **cv2**: For capturing and processing video frames.
  - **cvzone**: Simplifies OpenCV tasks, like drawing bounding boxes with rounded corners.
  - **HandDetector**: Detects and tracks hand landmarks.
  - **Controller**: Simulates physical keyboard key presses.

Section 2 : Video Capture Setup

```
vid_capture = cv.VideoCapture(0)
vid_capture.set(3, 1280)
vid_capture.set(4, 720)
```

- **Purpose**: Captures live video feed from the webcam and sets frame dimensions for high resolution.

Section 3 :Keyboard Button Class

```
class Button():
    def __init__(self, position, text, size=[70, 70]):
        self.position = position
        self.text = text
        self.size = size
```

- **Purpose**: Represents each key on the virtual keyboard, storing position, text, and size for rendering.

Section 4 : Drawing Keys on the Screen

```
def keyDrawing(frame, buttonList):
    for button in buttonList:
        ...
        cvzone.cornerRect(frame, ...)
        cv.rectangle(frame, ...)
        cv.putText(frame, ...)
    return frame
```

- **Purpose**: Dynamically renders keys on the video frame using OpenCV and cvzone.

Section 5 : Key Typing Functionality

```
def keyTyper(button, doType):
    if doType:
        keyboard.press(button)
```

- **Purpose**: Simulates keypresses on the physical keyboard.

Section 6 : Defining Keyboard Layout

```
keys_keyboard = [["Q", "W", "E", ...], ["A", "S", "D", ...], ...]
```

- **Purpose**: Creates a standard QWERTY keyboard layout as a nested list.
  Buttons for each key are initialized and stored in buttonList.

Section 7 : Hand Detection

```
detector = HandDetector(staticMode=False, maxHands=2, ...)
hands, frame = detector.findHands(frame, draw=True, ...)
```

- **Purpose**: Detects hand landmarks and bounding boxes using the HandDetector.

Section 8 : Interactive Keyboard Detection

```
for button in buttonList:
    ...
    if finger_length < 45:
        ...
        if button.text == "<--":
            ...
```

- **Purpose**: Tracks the user's finger to detect button clicks and simulates typing, text deletion, or space input.

Section 9 :Display Output

```
cv.rectangle(frame, (50, 450), (1000, 550), ...)
cv.putText(frame, text_written, ...)
cv.imshow("Camera Capture", frame)
```

- **Purpose**: Displays the live feed along with the virtual keyboard and the typed text.

Section 10 : Exit Condition

```
if cv.waitKey(2) == 27:
    break
```

6.3.2. Key Code Highlights

1. **Hand Detection**:

```
hands, frame = detector.findHands(frame, draw=True, flipType=False)
```

- **Explanation**: Leverages the HandDetector module to detect and track hands in each frame.

2. **Dynamic Key Press Detection**:

```
if finger_length < 45:
    ...
```

- **Explanation**: Checks if the user's finger is within a key's bounding box and confirms key selection.

3. **Text Generation**:

```
text_written += button.text
keyTyper(button.text, doType)
```

- **Explanation**: Updates the typed text string and simulates physical typing.

6.3.3. Frameworks Used

1. **OpenCV (cv2)**:

- **Definition**: A library for real-time computer vision.
- **Reason**: Provides tools for image processing and GUI overlay.
- **Usage**:
  - Video capture: cv.VideoCapture(0)
  - Drawing keys: cv.rectangle, cv.putText.

2. **cvzone**:

- **Definition**: An OpenCV wrapper with additional functionality.
- **Usage**: cvzone.cornerRect for drawing rounded rectangles.
- **Reason**: Simplifies complex OpenCV operations like key rendering.

3. **HandDetector (cvzone)**:

- **Definition**: A module for detecting and tracking hand landmarks.
- **Usage**: detector.findHands for hand and finger detection.
- **Reason**: Allows precise tracking of finger movements.

4. **pynput**:

- **Definition**: A library for controlling and monitoring input devices.
- **Usage**: keyboard.press(button) to simulate typing.
- **Reason**: Integrates virtual keypresses with real applications.

6.3.4. How the Code Works

1. **Initialization**:
    - Video feed is captured, and the keyboard layout is initialized.
    - The HandDetector starts detecting hand landmarks.
2. **Rendering Keyboard**:
    - Keys are drawn dynamically on the video frame.
    - User interaction is captured by tracking finger positions relative to the keys.
3. **Typing Interaction**:
    - When the user's finger hovers over a key and makes a "click" gesture (determined by distance between fingers), the corresponding key is "pressed."
4. **Displaying Text**:
    - The typed text is updated in real-time and displayed on the screen.
5. **Exit**:
    - The program ends when the ESC key is pressed.

6.3.5. Conclusion

The **Virtual Keyboard** enables a hands-free typing experience by detecting finger movements on a virtual keyboard displayed over the webcam feed. The program uses hand landmarks for detecting key presses and simulates key inputs in real-time.

The virtual keyboard implementation showcases the seamless integration of computer vision hand tracking, and virtual input simulation. This system offers a novel, hands-free typing interface suitable for accessibility applications and futuristic human-computer interaction solutions.

6.4. Virtual Mouse

Code :

```
import cv2
import numpy as np
import HandTrackingModule as htm
import time
# import autopy
import pyautogui


##############################
wCam, hCam = 640, 480
frameR = 100 # Frame Reduction
smoothening = 7
#############################

pTime = 0
plocX, plocY = 0, 0
clocX, clocY = 0, 0

cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
detector = htm.handDetector(maxHands=1)
# wScr, hScr = autopy.screen.size()
wScr, hScr = pyautogui.size()
# print(wScr, hScr)

while True:
    # 1. Find hand Landmarks
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)
    # 2. Get the tip of the index and middle fingers
    if len(lmList) != 0:
        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]
        # print(x1, y1, x2, y2)

    # 3. Check which fingers are up
    fingers = detector.fingersUp()
    # print(fingers)
    cv2.rectangle(img, (frameR, frameR), (wCam - frameR, hCam - frameR),
    (255, 0, 255), 2)
    # 4. Only Index Finger : Moving Mode
    if fingers[1] == 1 and fingers[2] == 0:
```

```
    # 5. Convert Coordinates
    x3 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
    y3 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
    # 6. Smoothen Values
    clocX = plocX + (x3 - plocX) / smoothening
    clocY = plocY + (y3 - plocY) / smoothening

    # 7. Move Mouse
    # //pyautogui.mouse.move(wScr - clocX, clocY)
    pyautogui.moveTo(wScr - clocX, clocY)
    pyautogui.move(wScr - clocX, clocY)
    pyautogui.click()
    cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
    plocX, plocY = clocX, clocY

# 8. Both Index and middle fingers are up : Clicking Mode
if fingers[1] == 1 and fingers[2] == 1:
    # 9. Find distance between fingers
    length, img, lineInfo = detector.findDistance(8, 12, img)
    print(length)
    # 10. Click mouse if distance short
    if length < 40:
        cv2.circle(img, (lineInfo[4], lineInfo[5]),
        15, (0, 255, 0), cv2.FILLED)
        pyautogui.click()

# 11. Frame Rate
cTime = time.time()
fps = 1 / (cTime - pTime)
pTime = cTime
cv2.putText(img, str(int(fps)), (20, 50), cv2.FONT_HERSHEY_PLAIN, 3,
(255, 0, 0), 3)
# 12. Display
cv2.imshow("Image", img)
if cv2.waitKey(1)==27

    break
```

6.4.1. Code Explanation (Line-by-Line)

Section 1 : Import Statements:

```
import cv2
import numpy as np
import HandTrackingModule as htm
import time
import pyautogui
```

- **cv2 (OpenCV)**: Used for computer vision tasks, such as video capture and image processing.
- **numpy**: Handles mathematical operations and array manipulation.
- **HandTrackingModule (htm)**: A custom module for detecting and tracking hand landmarks.
- **time**: To calculate the frame rate (FPS).
- **pyautogui**: Provides control over the mouse cursor and clicks for simulating interactions.

Section 2 : Camera and Screen Settings:

```
wCam, hCam = 640, 480
frameR = 100  # Frame Reduction
smoothening = 7
```

- Sets the resolution of the webcam feed and defines the region of interest (ROI) for cursor movement.
- Smoothening is used to reduce jitter in cursor movement by interpolating coordinates.

Section 3 : Initializations:

```
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
detector = htm.handDetector(maxHands=1)
wScr, hScr = pyautogui.size()
```

- **cv2.VideoCapture(0)** initializes the webcam.
- **htm.handDetector** is initialized to track one hand at a time.
- **pyautogui.size()** retrieves the screen resolution for mapping hand movement to cursor movement .

Section 4 : Main Loop:

```
while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)
```

- The main loop continuously captures frames from the webcam.
- **detector.findHands** identifies hand landmarks in the current frame.
- **findPosition** retrieves the positions of the hand's key landmarks.

Section 5 : Finger Tip Detection:

```
if len(lmList) != 0:
    x1, y1 = lmList[8][1:]
    x2, y2 = lmList[12][1:]
```

- Extracts the coordinates of the index (landmark 8) and middle fingers (landmark 12).

Section 6 : Finger State and Movement:

```
fingers = detector.fingersUp()
```

- **fingersUp()** determines which fingers are raised. This allows switching between "move mode" and "click mode."

Section 7 : Cursor Movement:

```
if fingers[1] == 1 and fingers[2] == 0:
    x3 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
    y3 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
    clocX = plocX + (x3 - plocX) / smoothening
    clocY = plocY + (y3 - plocY) / smoothening
    pyautogui.moveTo(wScr - clocX, clocY)
```

- Maps hand coordinates (x1, y1) to screen coordinates using numpy.interp.
- Smoothens the cursor's movement for better user experience.
- Moves the cursor to the calculated position using pyautogui.moveTo.

Section 8 : Click Mode:

```
if fingers[1] == 1 and fingers[2] == 1:
    length, img, lineInfo = detector.findDistance(8, 12, img)
    if length < 40:
        pyautogui.click()
```

- Detects when both the index and middle fingers are raised.
- Calculates the distance between the two fingers. A small distance (e.g., < 40) triggers a mouse click using pyautogui.click().

Section 9 : FPS Display:

```
fps = 1 / (time.time() - pTime)
pTime = time.time()
cv2.putText(img, str(int(fps)), (20, 50), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)
```

- Computes and displays the current frames per second on the video feed.

Section 10 : Exit Condition:

```
if cv2.waitKey(1) == 27:
    break
```

6.4.2. Key Highlights

1. **Gesture-Based Control**:
    - Moving the cursor: Only the index finger raised.
    - Clicking: Both index and middle fingers raised with a short distance between them.
2. **Smooth Cursor Movement**:
    - Smoothens rapid changes in hand position to prevent erratic cursor behavior.
3. **Frame Rate Calculation**:
    - Displays FPS to monitor performance.
4. **Integration with pyautogui**:
    - Enables direct control over the system's mouse for real-world usability.

6.4.3. Frameworks and Libraries

1. OpenCV (cv2):

- **Definition**: A popular library for real-time computer vision and image processing.
- **Usage**: Captures video feed, processes frames, and renders annotations like rectangles and text.
- **Code Reference**:

```
import cv2
cap = cv2.VideoCapture(0)
img = detector.findHands(img)
cv2.rectangle(...)
cv2.putText(...)
```

2. NumPy (numpy):

- **Definition**: A library for numerical computations.
- **Usage**: Interpolates hand coordinates to screen coordinates for smoother transitions.
- **Code Reference**:

```
x3 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
```

3. PyAutoGUI (pyautogui):

- **Definition**: A Python library for GUI automation.
- **Usage**: Moves the mouse and simulates clicks based on hand gestures.
- **Code Reference**:

```
pyautogui.moveTo(...)
pyautogui.click(...)
```

4. Custom Module (HandTrackingModule):

- **Definition**: A module (likely based on Mediapipe) to detect and track hand landmarks.
- **Usage**: Identifies hand positions and recognizes gestures.
- **Code Reference**:

```
detector = htm.handDetector(maxHands=1)
lmList, bbox = detector.findPosition(img)
```

6.4.4. How the Code Works

1. **Initialization**:
   - The webcam captures frames, and the HandTrackingModule identifies hand landmarks.
2. **Gesture Detection**:
   - The module determines which fingers are raised and calculates positions.
3. **Coordinate Mapping**:
   - Hand coordinates are mapped to the screen using interpolation.
4. **Mouse Simulation**:
   - Depending on gestures, the script moves the cursor or triggers a click.
5. **Smooth Interaction**:
   - The smoothening factor ensures fluid cursor movement for a better experience.
6. **Real-Time Feedback**:
   - FPS is displayed on the video feed to gauge performance.

6.4.5 File Summary

The virtual_mouse.py script uses computer vision to detect hand gestures and map them to mouse movements and clicks. It captures live video, detects hand landmarks, and interprets finger gestures to simulate mouse interactions on the screen.

This code is an excellent example of combining computer vision with GUI automation to create innovative, gesture-controlled applications.

6.5. Face Recognition for Attendance

- Tools: Python, Flask, Haar cascades.
- Description: Automates attendance and enhances security.

CODE :

```
import cv2
```

```
import os
from flask import Flask, Response, request, render_template
from datetime import date, datetime
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import joblib
import base64

app = Flask(__name__)

nimgs = 10

imgBackground = cv2.imread("background.png")

datetoday = date.today().strftime("%m_%d_%y")
datetoday2 = date.today().strftime("%d-%B-%Y")

face_detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Create necessary folders if they don't exist
os.makedirs('Attendance', exist_ok=True)
os.makedirs('static/faces', exist_ok=True)
if f'Attendance-{datetoday}.csv' not in os.listdir('Attendance'):
    with open(f'Attendance/Attendance-{datetoday}.csv', 'w') as f:
        f.write('Name,Roll,Time')


def totalreg():
    return len(os.listdir('static/faces'))


def extract_faces(img):
    try:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        face_points = face_detector.detectMultiScale(gray, 1.2, 5, minSize=(20, 20))
        return face_points
    except:
        return []


def identify_face(facearray):
    model = joblib.load('static/face_recognition_model.pkl')
    return model.predict(facearray)


def train_model():
    faces = []
    labels = []
    userlist = os.listdir('static/faces')
    for user in userlist:
        for imgname in os.listdir(f'static/faces/{user}'):
```

```
        img = cv2.imread(f'static/faces/{user}/{imgname}')
        resized_face = cv2.resize(img, (50, 50))
        faces.append(resized_face.ravel())
        labels.append(user)
    faces = np.array(faces)
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(faces, labels)
    joblib.dump(knn, 'static/face_recognition_model.pkl')


def extract_attendance():
    df = pd.read_csv(f'Attendance/Attendance-{datetoday}.csv')
    names = df['Name']
    rolls = df['Roll']
    times = df['Time']
    l = len(df)
    return names, rolls, times, l


def add_attendance(name):
    username = name.split('_')[0]
    userid = name.split('_')[1]
    current_time = datetime.now().strftime("%H:%M:%S")

    df = pd.read_csv(f'Attendance/Attendance-{datetoday}.csv')
    if int(userid) not in list(df['Roll']):
        with open(f'Attendance/Attendance-{datetoday}.csv', 'a') as f:
            f.write(f'\n{username},{userid},{current_time}')


@app.route('/')
def home():
    names, rolls, times, l = extract_attendance()
    return render_template('home.html', names=names, rolls=rolls, times=times, l=l,
                           totalreg=totalreg(), datetoday2=datetoday2)


@app.route('/start')
def start():
    def generate_frames():
        cap = cv2.VideoCapture(0)
        while True:
            ret, frame = cap.read()
            if not ret:
                break

            resized_frame = cv2.resize(frame, (640, 480))

            if len(extract_faces(resized_frame)) > 0:
                (x, y, w, h) = extract_faces(resized_frame)[0]
                cv2.rectangle(resized_frame, (x, y), (x + w, y + h), (86, 32, 251), 1)
                face = cv2.resize(resized_frame[y:y + h, x:x + w], (50, 50))
```

```
            identified_person = identify_face(face.reshape(1, -1))[0]
            add_attendance(identified_person)
            cv2.putText(resized_frame, f'{identified_person}', (x, y - 15),
                        cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 1)

        imgBackground[162:162 + 480, 55:55 + 640] = resized_frame
        _, buffer = cv2.imencode('.jpg', imgBackground)
        frame_data = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_data + b'\r\n')

    cap.release()

  return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')


@app.route('/add', methods=['POST'])
def add():
  newusername = request.form['newusername']
  newuserid = request.form['newuserid']
  userimagefolder = f'static/faces/{newusername}_{newuserid}'
  os.makedirs(userimagefolder, exist_ok=True)

  def generate_frames():
    cap = cv2.VideoCapture(0)
    i, j = 0, 0

    while i < nimgs:
      ret, frame = cap.read()
      if not ret:
        break

      faces = extract_faces(frame)
      for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 20), 2)
        if j % 5 == 0:
          name = f'{newusername}_{i}.jpg'
          cv2.imwrite(f'{userimagefolder}/{name}', frame[y:y + h, x:x + w])
          i += 1
        j += 1

      _, buffer = cv2.imencode('.jpg', frame)
      frame_data = buffer.tobytes()

      yield (b'--frame\r\n'
             b'Content-Type: image/jpeg\r\n\r\n' + frame_data + b'\r\n')

    cap.release()
    train_model()
```

```
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')


if __name__ == '__main__':
    app.run(debug=True)
```

6.5.1 Code Explanation

Section 1 : Import Statements

- **Libraries used:**
  - **cv2 (OpenCV):** For image and video processing.
  - **os:** To interact with the file system.
  - **Flask:** A web framework to create the application.
  - **datetime:** To get the current date and time for attendance tracking.
  - **numpy:** For numerical operations and data manipulation.
  - **sklearn.neighbors.KNeighborsClassifier:** A machine learning model for face recognition.
  - **pandas:** For handling CSV data related to attendance.
  - **joblib:** To save and load the trained KNN model.
  - **base64:** For encoding/decoding images in certain operations.

Section 2 : Application Initialization

- **app = Flask(__name__):** Creates a Flask application instance.
- **Directories Creation:**
  - **Attendance:** To store daily attendance CSV files.
  - **static/faces:** To store user face images for model training.
- **Attendance File Setup:**
  - If a file for today's attendance doesn't exist, a new one is created.

Section 3 : Utility Functions

- **totalreg():** Counts the total registered users based on the images stored in static/faces.
- **extract_faces(img):** Detects faces in an image using OpenCV's Haar cascade classifier.
- **identify_face(facearray):** Loads the saved KNN model and predicts the identity of the input face.
- **train_model():**
  - Reads images from the static/faces folder.
  - Trains a KNN model on the face data.
  - Saves the trained model using joblib.
- **extract_attendance():** Reads the attendance CSV file and returns the names, rolls, times, and count of attendees.

- **add_attendance(name):** Adds the identified user to the attendance file with the current timestamp.

Section 4 : Flask Routes

- **/ (Home):**
  - Renders the home page.
  - Displays attendance statistics and registered user count.
- **/start:**
  - Captures live video.
  - Detects and identifies faces.
  - Updates the attendance list.
  - Streams video with overlaid detection results.
- **/add:**
  - Captures and saves new user images.
  - Trains the model with new data.

6.5.2. How the Code Works

1. **Initialization:**
   - The application initializes necessary folders, loads the Haar cascade model, and sets up the attendance system.
2. **Face Detection and Recognition:**
   - Using OpenCV, faces are detected in frames.
   - Identified faces are checked against the trained KNN model for recognition.
3. **Attendance System:**
   - Matches identified faces to users and updates the attendance CSV for the day.
4. **User Addition:**
   - A new user's face images are captured, stored, and used to train the KNN model.
5. **Live Video Stream:**
   - Video is processed frame by frame.
   - Recognized user information is overlaid on the video feed.

6.5.3. Key Highlights

1. Face Detection:

- Done using Haar cascade (face_detector.detectMultiScale()).
- Code Reference:

```
2.  face_points = face_detector.detectMultiScale(gray, 1.2, 5, minSize=(20, 20))
3.
```

  - Reason: Efficient and lightweight face detection.

2. Face Recognition:

- KNN classifier used (KNeighborsClassifier).
- Code Reference:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(faces, labels)
```

  - Reason: Simple and effectiv for small datasets.

3. Attendance Tracking:

- Based on CSV files created dynamically.
- Ensures records are unique by checking roll numbers.

4. Real-time Video Processing:

- Frames are streamed with detections using Flask.
- Code Reference:

```
yield (b'--frame\r\n'
    b'Content-Type: image/jpeg\r\n\r\n' + frame_data + b'\r\n')
```

5. Dynamic User Addition:

- Users can be added to the system via the /add endpoint.
- The system updates the recognition model automatically.

6.5.4. Frameworks Used

1. Flask

- **Definition:** A lightweight WSGI web application framework in Python.
- **Reason:** Ideal for creating small, scalable web applications.
- **Code Usage:**

```
app = Flask(__name__)
@app.route('/')
def home():
    return render_template('home.html', ...)
```

2. OpenCV

- **Definition:** A computer vision library for image processing tasks.
- **Reason:** Provides efficient tools for face detection and manipulation.
- **Code Usage:**

```
face_detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

3. scikit-learn

- **Definition:** A machine learning library in Python.
- **Reason:** KNN implementation for face recognition.
- **Code Usage:**

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(faces, labels)
```

4. pandas

- **Definition:** A library for data manipulation and analysis.
- **Reason:** Used for handling attendance records stored in CSV format.
- **Code Usage:**

```
df = pd.read_csv(f'Attendance/Attendance-{datetoday}.csv')
```

5. joblib

- **Definition:** A library for efficient serialization of Python objects.
- **Reason:** Saves and loads the trained face recognition model.
- **Code Usage:**

```
joblib.dump(knn, 'static/face_recognition_model.pkl')
```

6.5.5. Summary

- The facerecognition.py script integrates computer vision and machine learning with a web application to create a real-time face recognition-based attendance system.
- Key functionalities include face detection, recognition, attendance tracking, and dynamic user management.
- It effectively leverages OpenCV for image processing, Flask for the web interface, and scikit-learn for face recognitio

---

6.6. Smartboard

- Tools: Interactive display technology.
- Description: Combines touch inputs and dynamic content.

CODE :

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

# Initialize MediaPipe
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.5, min_tracking_confidence=0.5)
mp_drawing = mp.solutions.drawing_utils

# Start capturing video
cap = cv2.VideoCapture(0)

# Initialize previous point and eraser state
prev_point = None
eraser_active = False

# Create a blank canvas
canvas = np.zeros((480, 640, 3), dtype=np.uint8)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convert frame to RGB
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Run hand tracking on the frame
    results = hands.process(frame_rgb)

    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            # Get landmark positions
            thumb_tip = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
            index_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]

            # Calculate position
            index_x, index_y = int(index_tip.x * frame.shape[1]), int(index_tip.y * frame.shape[0])
            thumb_x, thumb_y = int(thumb_tip.x * frame.shape[1]), int(thumb_tip.y * frame.shape[0])
            distance = np.sqrt((index_x - thumb_x) ** 2 + (index_y - thumb_y) ** 2)

            # Draw a circle at the finger position on the frame
            cv2.circle(frame, (index_x, index_y), 8, (0, 255, 0), -1)

            # Check if thumb is close to index finger to activate eraser
            if distance < 30:
                eraser_active = True
            else:
                eraser_active = False

            # Draw line or erase canvas if previous point exists
```

```
        if prev_point:
            if eraser_active:
                canvas = np.zeros((480, 640, 3), dtype=np.uint8)  # Erase canvas
            else:
                cv2.line(canvas, prev_point, (index_x, index_y), (0, 255, 0), 2)

        # Update previous point
        prev_point = (index_x, index_y)

    # Overlay canvas on the frame
    print(f"Frame shape: {frame.shape}")
    print(f"Canvas shape: {canvas.shape}")
    canvas = cv2.resize(canvas, (frame.shape[1], frame.shape[0]))
    frame = cv2.add(frame, canvas)
    canvas_flipped = cv2.flip(canvas, 1)
    frame_flipped = cv2.flip(frame, 1)

    # Display the frame
    cv2.imshow('Finger Tracking', frame_flipped)
    cv2.imshow('Canvas', canvas_flipped)

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

# Release resources
cap.release()
cv2.destroyAllWindows()
```

6.6.1. Code Chunk Explanations

Section 1 : Importing Libraries

```
import cv2
import mediapipe as mp
import numpy as np
```

- **Purpose**: Import essential libraries for computer vision, hand tracking, and canvas manipulation.
- **Key Libraries**:
    - **cv2**: Used for video capture, drawing, and display of frames.
    - **mediapipe**: Tracks hands and provides landmark data for interactions.
    - **numpy**: Manages pixel-level operations for the canvas.

Section 2 : MediaPipe Hands Initialization

```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.5, min_tracking_confidence=0.5)
mp_drawing = mp.solutions.drawing_utils
```

- **Purpose**: Initializes the MediaPipe Hands model for hand and landmark detection.

Section 3 : Video Capture Setup

```
cap = cv2.VideoCapture(0)
```

- **Purpose**: Captures live video feed from the default camera.

Section 4 : Canvas Initialization

```
canvas = np.zeros((480, 640, 3), dtype=np.uint8)
```

- **Purpose**: Creates a black canvas for drawing.

Section 5 : Main Loop

- Captures video frames and processes each frame for hand landmarks.

Section 6 : Hand Tracking and Landmark Detection

```
results = hands.process(frame_rgb)
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        ...
```

- **Purpose**: Detects hand landmarks in the current frame.
- **Details**:
    - thumb_tip and index_tip: Extracts coordinates of thumb and index fingertips.
    - Computes distance to toggle between drawing and erasing.

Section 7 : Drawing and Erasing Logic

```
if prev_point:
    if eraser_active:
        canvas = np.zeros((480, 640, 3), dtype=np.uint8)  # Erase canvas
    else:
        cv2.line(canvas, prev_point, (index_x, index_y), (0, 255, 0), 2)
prev_point = (index_x, index_y)
```

- **Purpose**:
    - Draws a line between the previous point and the current fingertip position.
    - Clears the canvas if the eraser is active (thumb is close to index finger).

Section 8 : Overlay Canvas on Frame

```
canvas = cv2.resize(canvas, (frame.shape[1], frame.shape[0]))
frame = cv2.add(frame, canvas)
```

- **Purpose**: Combines the live video feed with the updated drawing canvas.

Section 9 : Flipping and Display

```
canvas_flipped = cv2.flip(canvas, 1)
frame_flipped = cv2.flip(frame, 1)
cv2.imshow('Finger Tracking', frame_flipped)
cv2.imshow('Canvas', canvas_flipped)
```

- **Purpose**: Mirrors the video feed for a more intuitive user experience and displays the frame and canvas.

Section 10 : Exit Condition

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

- **Purpose**: Exits the program when the user presses q.

Section 11 : Release Resources

```
cap.release()
cv2.destroyAllWindows()
```

6.6.2. Key Code Highlights

1. **Gesture-Based Eraser Activation**

```
distance = np.sqrt((index_x - thumb_x) ** 2 + (index_y - thumb_y) ** 2)
if distance < 30:
    eraser_active = True
else:
    eraser_active = False
```

   - **Explanation**: Calculates the Euclidean distance between the thumb and index finger to determine whether the eraser mode is active.

2. **Dynamic Line Drawing**

```
cv2.line(canvas, prev_point, (index_x, index_y), (0, 255, 0), 2)
```

   - **Explanation**: Draws a continuous line following the index fingertip movement.

3. **Canvas Reset**

```
canvas = np.zeros((480, 640, 3), dtype=np.uint8)
```

   - **Explanation**: Clears the entire canvas when the eraser is active.

6.6.3. Frameworks Used

1. OpenCV (cv2):

- **Usage**:
  - Captures and processes video frames.
  - Draws lines and overlays the canvas.
- **Reason**: Provides powerful tools for real-time computer vision tasks.

2. MediaPipe:

- **Usage**:
  - Tracks hand landmarks for interactive gestures.

- **Reason**: Simplifies complex hand detection tasks with reliable performance.

3. **NumPy:**

    - **Usage**:
        - Creates and manipulates the canvas as a pixel array.
    - **Reason**: Efficient handling of image data with high performance.

6.6.4. How the Code Works

1. **Initialization**:
    - Sets up the webcam, MediaPipe Hand module, and canvas.
2. **Real-Time Hand Tracking**:
    - Tracks the index finger's position and calculates its distance from the thumb.
3. **Interactive Drawing and Erasing**:
    - Draws lines based on the index finger's movement.
    - Erases the canvas when the thumb is near the index finger.
4. **Display and Exit**:
    - Shows the live feed and the canvas until the user presses q.

6.6.5. Conclusion

The **Smartboard** implementation turns any webcam-enabled device into an interactive drawing platform. Using fingertip tracking and gestures, the user can draw or erase directly on the live video feed.

The **Smartboard** implementation showcases the potential of combining gesture recognition and real-time drawing to create an interactive tool for education, brainstorming, and creative applications. Its reliance on simple yet effective gestures makes it accessible and user-friendly.

6.7 Proctored Exam

Code :

```python
import cv2 as cv
import mediapipe as mp
import time
import utils, math
import numpy as np
import pygame
from pygame import mixer

# variables
frame_counter =0
CEF_COUNTER =0
TOTAL_BLINKS =0
start_voice= False
counter_right=0
counter_left =0
counter_center =0
# constants
CLOSED_EYES_FRAME =3
FONTS =cv.FONT_HERSHEY_COMPLEX

# initialize mixer
# mixer.init()
# loading in the voices/sounds
# voice_left = mixer.Sound('Voice/left.wav')
# voice_right = mixer.Sound('Voice/Right.wav')
# voice_center = mixer.Sound('Voice/center.wav')

# face bounder indices
FACE_OVAL=[ 10, 338, 297, 332, 284, 251, 389, 356, 454, 323, 361, 288, 397, 365, 379, 378, 400, 377, 152, 148, 176, 149,
150, 136, 172, 58, 132, 93, 234, 127, 162, 21, 54, 103,67, 109]

# lips indices for Landmarks
LIPS=[ 61, 146, 91, 181, 84, 17, 314, 405, 321, 375,291, 308, 324, 318, 402, 317, 14, 87, 178, 88, 95,185, 40, 39, 37,0 ,267 ,269
,270 ,409, 415, 310, 311, 312, 13, 82, 81, 42, 183, 78 ]
LOWER_LIPS =[61, 146, 91, 181, 84, 17, 314, 405, 321, 375, 291, 308, 324, 318, 402, 317, 14, 87, 178, 88, 95]
UPPER_LIPS=[ 185, 40, 39, 37,0 ,267 ,269 ,270 ,409, 415, 310, 311, 312, 13, 82, 81, 42, 183, 78]
# Left eyes indices
LEFT_EYE =[ 362, 382, 381, 380, 374, 373, 390, 249, 263, 466, 388, 387, 386, 385,384, 398 ]
LEFT_EYEBROW =[ 336, 296, 334, 293, 300, 276, 283, 282, 295, 285 ]

# right eyes indices
RIGHT_EYE=[ 33, 7, 163, 144, 145, 153, 154, 155, 133, 173, 157, 158, 159, 160, 161 , 246 ]
RIGHT_EYEBROW=[ 70, 63, 105, 66, 107, 55, 65, 52, 53, 46 ]

map_face_mesh = mp.solutions.face_mesh

# camera object
camera = cv.VideoCapture(0)
```

```
_, frame = camera.read()
img = cv.resize(frame, None, fx=1.5, fy=1.5, interpolation=cv.INTER_CUBIC)
img_hieght, img_width = img.shape[:2]
print(img_hieght, img_width)



# video Recording setup
fourcc = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('output21.mp4', fourcc, 30.0, (img_width, img_hieght))
# landmark detection function

def landmarksDetection(img, results, draw=False):
    img_height, img_width= img.shape[:2]
    # list[(x,y), (x,y)....]
    mesh_coord = [(int(point.x * img_width), int(point.y * img_height)) for point in results.multi_face_landmarks[0].landmark]
    if draw :
        [cv.circle(img, p, 2, (0,255,0), -1) for p in mesh_coord]

    # returning the list of tuples for each landmarks
    return mesh_coord

# Euclaidean distance
def euclaideanDistance(point, point1):
    x, y = point
    x1, y1 = point1
    distance = math.sqrt((x1 - x)**2 + (y1 - y)**2)
    return distance

# Blinking Ratio
def blinkRatio(img, landmarks, right_indices, left_indices):
    # Right eyes
    # horizontal line
    rh_right = landmarks[right_indices[0]]
    rh_left = landmarks[right_indices[8]]
    # vertical line
    rv_top = landmarks[right_indices[12]]
    rv_bottom = landmarks[right_indices[4]]
    # draw lines on right eyes
    # cv.line(img, rh_right, rh_left, utils.GREEN, 2)
    # cv.line(img, rv_top, rv_bottom, utils.WHITE, 2)

    # LEFT_EYE
    # horizontal line
    lh_right = landmarks[left_indices[0]]
    lh_left = landmarks[left_indices[8]]

    # vertical line
    lv_top = landmarks[left_indices[12]]
    lv_bottom = landmarks[left_indices[4]]
```

```
    rhDistance = euclaideanDistance(rh_right, rh_left)
    rvDistance = euclaideanDistance(rv_top, rv_bottom)

    lvDistance = euclaideanDistance(lv_top, lv_bottom)
    lhDistance = euclaideanDistance(lh_right, lh_left)

    reRatio = rhDistance/rvDistance
    leRatio = lhDistance/lvDistance

    ratio = (reRatio+leRatio)/2
    return ratio

# Eyes Extrctor function,
def eyesExtractor(img, right_eye_coords, left_eye_coords):
    # converting color image to  scale image
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # getting the dimension of image
    dim = gray.shape

    # creating mask from gray scale dim
    mask = np.zeros(dim, dtype=np.uint8)

    # drawing Eyes Shape on mask with white color
    cv.fillPoly(mask, [np.array(right_eye_coords, dtype=np.int32)], 255)
    cv.fillPoly(mask, [np.array(left_eye_coords, dtype=np.int32)], 255)

    # showing the mask
    # cv.imshow('mask', mask)

    # draw eyes image on mask, where white shape is
    eyes = cv.bitwise_and(gray, gray, mask=mask)
    # change black color to gray other than eys
    # cv.imshow('eyes draw', eyes)
    eyes[mask==0]=155

    # getting minium and maximum x and y  for right and left eyes
    # For Right Eye
    r_max_x = (max(right_eye_coords, key=lambda item: item[0]))[0]
    r_min_x = (min(right_eye_coords, key=lambda item: item[0]))[0]
    r_max_y = (max(right_eye_coords, key=lambda item : item[1]))[1]
    r_min_y = (min(right_eye_coords, key=lambda item: item[1]))[1]

    # For LEFT Eye
    l_max_x = (max(left_eye_coords, key=lambda item: item[0]))[0]
    l_min_x = (min(left_eye_coords, key=lambda item: item[0]))[0]
    l_max_y = (max(left_eye_coords, key=lambda item : item[1]))[1]
    l_min_y = (min(left_eye_coords, key=lambda item: item[1]))[1]

    # croping the eyes from mask
    cropped_right = eyes[r_min_y: r_max_y, r_min_x: r_max_x]
```

```
    cropped_left = eyes[l_min_y: l_max_y, l_min_x: l_max_x]

    # returning the cropped eyes
    return cropped_right, cropped_left

# Eyes Postion Estimator
def positionEstimator(cropped_eye):
    # getting height and width of eye
    h, w =cropped_eye.shape

    # remove the noise from images
    gaussain_blur = cv.GaussianBlur(cropped_eye, (9,9),0)
    median_blur = cv.medianBlur(gaussain_blur, 3)

    # applying thrsholding to convert binary_image
    ret, threshed_eye = cv.threshold(median_blur, 130, 255, cv.THRESH_BINARY)

    # create fixd part for eye with
    piece = int(w/3)

    # slicing the eyes into three parts
    right_piece = threshed_eye[0:h, 0:piece]
    center_piece = threshed_eye[0:h, piece: piece+piece]
    left_piece = threshed_eye[0:h, piece +piece:w]

    # calling pixel counter function
    eye_position, color = pixelCounter(right_piece, center_piece, left_piece)

    return eye_position, color

# creating pixel counter function
def pixelCounter(first_piece, second_piece, third_piece):
    # counting black pixel in each part
    right_part = np.sum(first_piece==0)
    center_part = np.sum(second_piece==0)
    left_part = np.sum(third_piece==0)
    # creating list of these values
    eye_parts = [right_part, center_part, left_part]

    # getting the index of max values in the list
    max_index = eye_parts.index(max(eye_parts))
    pos_eye =''
    if max_index==0:
        pos_eye="RIGHT"
        color=[utils.BLACK, utils.GREEN]
    elif max_index==1:
        pos_eye = 'CENTER'
        color = [utils.YELLOW, utils.PINK]
    elif max_index ==2:
        pos_eye = 'LEFT'
        color = [utils.GRAY, utils.YELLOW]
```

```
    else:
        pos_eye="Closed"
        color = [utils.GRAY, utils.YELLOW]
    return pos_eye, color


with map_face_mesh.FaceMesh(min_detection_confidence =0.5, min_tracking_confidence=0.5) as face_mesh:

    # starting time here
    start_time = time.time()
    # starting Video loop here.
    while True:
        frame_counter +=1 # frame counter
        ret, frame = camera.read() # getting frame from camera
        if not ret:
            break # no more frames break
        #  resizing frame

        frame = cv.resize(frame, None, fx=1.5, fy=1.5, interpolation=cv.INTER_CUBIC)
        frame_height, frame_width= frame.shape[:2]
        rgb_frame = cv.cvtColor(frame, cv.COLOR_RGB2BGR)
        results  = face_mesh.process(rgb_frame)
        if results.multi_face_landmarks:
            mesh_coords = landmarksDetection(frame, results, False)
            ratio = blinkRatio(frame, mesh_coords, RIGHT_EYE, LEFT_EYE)
            # cv.putText(frame, f'ratio {ratio}', (100, 100), FONTS, 1.0, utils.GREEN, 2)
            utils.colorBackgroundText(frame,  f'Ratio : {round(ratio,2)}', FONTS, 0.7, (30,100),2, utils.PINK, utils.YELLOW)

            if ratio >5.5:
                CEF_COUNTER +=1
                # cv.putText(frame, 'Blink', (200, 50), FONTS, 1.3, utils.PINK, 2)
                utils.colorBackgroundText(frame,    f'Blink',  FONTS,  1.7,  (int(frame_height/2),  100),  2,  utils.YELLOW,  pad_x=6,
pad_y=6, )

            else:
                if CEF_COUNTER>CLOSED_EYES_FRAME:
                    TOTAL_BLINKS +=1
                    CEF_COUNTER =0
            # cv.putText(frame, f'Total Blinks: {TOTAL_BLINKS}', (100, 150), FONTS, 0.6, utils.GREEN, 2)
            utils.colorBackgroundText(frame,  f'Total Blinks: {TOTAL_BLINKS}', FONTS, 0.7, (30,150),2)

            cv.polylines(frame,  [np.array([mesh_coords[p] for p in LEFT_EYE ], dtype=np.int32)], True, utils.GREEN, 1, cv.LINE_AA)
            cv.polylines(frame,    [np.array([mesh_coords[p]  for  p  in  RIGHT_EYE  ],  dtype=np.int32)],  True,  utils.GREEN,  1,
cv.LINE_AA)

            # Blink Detector Counter Completed
            right_coords = [mesh_coords[p] for p in RIGHT_EYE]
            left_coords = [mesh_coords[p] for p in LEFT_EYE]
            crop_right, crop_left = eyesExtractor(frame, right_coords, left_coords)
            # cv.imshow('right', crop_right)
            # cv.imshow('left', crop_left)
```

```
        eye_position_right, color = positionEstimator(crop_right)
        utils.colorBackgroundText(frame, f'R: {eye_position_right}', FONTS, 1.0, (40, 220), 2, color[0], color[1], 8, 8)
        eye_position_left, color = positionEstimator(crop_left)
        utils.colorBackgroundText(frame, f'L: {eye_position_left}', FONTS, 1.0, (40, 320), 2, color[0], color[1], 8, 8)

        # Starting Voice Indicator
        if eye_position_right=="RIGHT" and pygame.mixer.get_busy()==0 and counter_right<2:
            # starting counter
            counter_right+=1
            # resetting counters
            counter_center=0
            counter_left=0
            # playing voice
            # voice_right.play()


        if eye_position_right=="CENTER" and pygame.mixer.get_busy()==0 and counter_center <2:
            # starting Counter
            counter_center +=1
            # resetting counters
            counter_right=0
            counter_left=0
            # playing voice
            # voice_center.play()

        if eye_position_right=="LEFT" and pygame.mixer.get_busy()==0 and counter_left<2:
            counter_left +=1
            # resetting counters
            counter_center=0
            counter_right=0
            # playing Voice
            # voice_left.play()



    # calculating  frame per seconds FPS
    end_time = time.time()-start_time
    fps = frame_counter/end_time


    frame =utils.textWithBackground(frame,f'FPS: {round(fps,1)}',FONTS, 1.0, (30, 50), bgOpacity=0.9, textThickness=2)
    # writing image for thumbnail drawing shape
    # cv.imwrite(f'img/frame_{frame_counter}.png', frame)
    # wirting the video for demo purpose
    out.write(frame)
    cv.imshow('frame', frame)
    key = cv.waitKey(2)
    if key==ord('q') or key ==ord('Q'):
        break
cv.destroyAllWindows()
camera.release()
```

6.7.1 File Overview

The script proctored_exam.py is designed to monitor a user's eye movements and blinks during an exam session, using the camera feed. The application incorporates computer vision and machine learning techniques to detect facial landmarks, analyze eye positions, count blinks, and track gaze direction. Additionally, audio feedback mechanisms are included (though commented out) to notify the user about their gaze direction.

6.7.2. Frameworks/Packages Used:

```
import cv2 as cv
import mediapipe as mp
import time
import utils, math
import numpy as np
import pygame
from pygame import mixer
```

1. OpenCV (cv2):

- Used for video capture, image processing, and visualization.
- Example usage:

```
camera = cv.VideoCapture(0)
frame = cv.resize(frame, None, fx=1.5, fy=1.5, interpolation=cv.INTER_CUBIC)
```

  - **Reason**: Provides powerful tools for image and video manipulation.

2. Mediapipe (mp):

- Used for face mesh detection, providing 3D facial landmarks.
- Example usage:

```
map_face_mesh = mp.solutions.face_mesh
results = face_mesh.process(rgb_frame)
```

  - **Reason**: Mediapipe offers robust and efficient face detection algorithms.

3. NumPy (np):

- For array manipulation, including masks and coordinate transformations.
- Example usage:

```
mask = np.zeros(dim, dtype=np.uint8)
```

  - **Reason**: Simplifies numerical operations and matrix computations.

4. Pygame and Mixer:

- Audio feedback for gaze direction notifications (commented out).
- Example usage:

```
mixer.Sound('Voice/left.wav').play()
```

  - **Reason**: Facilitates easy integration of audio features.

6.7.3. Key Code Components and Their Explanation

Section 1 : Constants and Indices

```
FACE_OVAL, LIPS, LEFT_EYE, RIGHT_EYE = [ ... ]
```

These predefined indices correspond to facial landmarks detected by Mediapipe's face mesh model. For instance, `LEFT_EYE` contains points representing the left eye region.

Section 2 : Video Capture Setup

```
camera = cv.VideoCapture(0)
frame = cv.resize(frame, None, fx=1.5, fy=1.5, interpolation=cv.INTER_CUBIC)
```

The script captures live video from the default camera, resizing frames for better processing accuracy.

Section 3 : Face Landmark Detection

```
def landmarksDetection(img, results, draw=False):
    ...
    return mesh_coord
```

This function retrieves and processes facial landmark coordinates, scaling them to match the resolution of the input image.

Section 4 : Blink Detection

```
def blinkRatio(img, landmarks, right_indices, left_indices):
    ...
    return ratio
```

Uses the Euclidean distance formula to compute a ratio of horizontal to vertical eye distances. A high ratio signifies closed eyes (blink detection).

Section 5 : Eye Position Estimation

```
def positionEstimator(cropped_eye):
    ...
    return eye_position, color
```

Divides the eye region into three parts (right, center, left), counting black pixels to determine the gaze direction.

Section 6 : Main Video Processing Loop

```
while True:
  ...
  results = face_mesh.process(rgb_frame)
  ...
  crop_right, crop_left = eyesExtractor(frame, right_coords, left_coords)
  ...
  fps = frame_counter/end_time
```

The loop:

- Captures and processes each video frame.
- Detects facial landmarks and calculates blink and gaze metrics.
- Outputs results visually and records them into a video file.

6.7.4. Key Highlights

Section 1 : Facial Landmark Detection with Mediapipe:

- Mediapipe's pre-trained model tracks facial features efficiently.

```
map_face_mesh = mp.solutions.face_mesh
results = face_mesh.process(rgb_frame)
```

Section 2 : Blink Detection Logic:

- Horizontal and vertical eye distances are compared to determine whether eyes are open or closed.

```
ratio = blinkRatio(frame, mesh_coords, RIGHT_EYE, LEFT_EYE)
```

Section 3 : Eye Gaze Direction:

- Eye regions are analyzed to estimate the gaze direction (left, center, right).

```
eye_position_right, color = positionEstimator(crop_right)
```

Section 4 : Real-time Feedback:

- Visual output is updated dynamically on each frame, and audio alerts are triggered based on gaze behavior (commented out).

Section 5 : Video Recording:

- Outputs the processed video feed to an .mp4 file for further review.

```
out.write(frame)
```

6.7.5. Frameworks and Their Applications

1. **OpenCV**:
   - **Definition**: Open-source library for real-time computer vision.
   - **Usage in Code**:
     - Captures video: cv.VideoCapture(0)
     - Draws shapes: cv.polylines(frame, ...)
     - Processes frames: cv.resize(), cv.cvtColor()
   - **Reason**: Essential for video handling and image transformations.

2. **Mediapipe**:

- **Definition**: Google’s framework for machine learning pipelines, optimized for real-time inference.
- **Usage in Code**:
  - Face mesh detection: mp.solutions.face_mesh
- **Reason**: Provides efficient facial landmark detection.

3. **NumPy**:
   - **Definition**: A library for numerical and matrix operations.
   - **Usage in Code**:
     - Masks for eye region: np.zeros(dim, dtype=np.uint8)
   - **Reason**: Simplifies array manipulations and mathematical operations.

4. **Pygame**:
   - **Definition**: Cross-platform library for multimedia applications.
   - **Usage in Code**:
     - Audio feedback: mixer.Sound().play()
   - **Reason**: Enables user interaction through sound notifications.

6.7.6. How the Code Works

1. **Initialization**:
   - Imports dependencies, sets up camera input, and configures Mediapipe's face mesh.
2. **Real-time Video Processing**:
   - Captures each frame, processes it for facial landmarks, and computes blink and gaze metrics.
3. **Feedback and Recording**:
   - Updates video output with overlays indicating blink counts and gaze direction.
   - Saves the processed video for post-exam analysis.
4. **Audio Alerts (Optional)**:
   - Notifies users audibly when their gaze deviates from the screen.
5. **Exit Conditions**:
   - Terminates when the user presses 'q'.

6.7.7. Final Summary

The script efficiently tracks eye movements and blinks in real time using OpenCV and Mediapipe. It’s designed to support proctored exam scenarios where monitoring user focus is critical. While the framework's real-time performance is impressive, additional optimization could further enhance the script for deployment in large-scale environments.

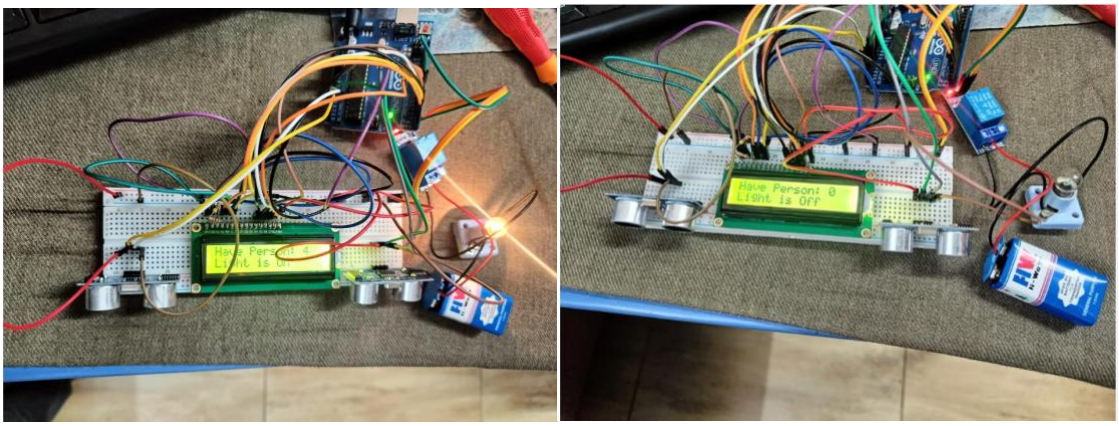
6.8. Automatic light on/off on person entry/exit and counter

The smart classroom system utilizes sensors to automatically control lighting based on individual entry and exit, ensuring optimal illumination while accurately tracking occupancy levels for efficient energy management.

CODE :

```
#include <LiquidCrystal.h> // initialize the library with the numbers of the interface pins
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);
#define e_s1 A0 //echo pin
#define t_s1 A1 //Trigger pin
#define e_s2 A2 //echo pin
#define t_s2 A3 //Trigger pin
int relay = 8; // Out for light
long dis_a = 0, dis_b = 0;
int flag1 = 0, flag2 = 0;
int person = 0;
//ultra_read*************************
void ultra_read(int pin_t, int pin_e, long &ultra_time) {
long time;
pinMode(pin_t, OUTPUT);
pinMode(pin_e, INPUT);
digitalWrite(pin_t, LOW);
delayMicroseconds(2);
digitalWrite(pin_t, HIGH);
delayMicroseconds(10);
time = pulseIn(pin_e, HIGH);
ultra_time = time / 29 / 2;
}
void setup() {
Serial.begin(9600); // initialize serial communication at 9600 bits per second:
pinMode(relay, OUTPUT);
lcd.begin(16, 2);
lcd.setCursor(0, 0);
lcd.print(" Welcome ");
delay(1000); // Waiting for a while
lcd.clear();
}
```

```
void loop() {
//************************
ultra_read(t_s1, e_s1, dis_a);
delay(30);
ultra_read(t_s2, e_s2, dis_b);
delay(30);
//************************
Serial.print("da:");
Serial.println(dis_a);
Serial.print("db:");
Serial.println(dis_b);
if (dis_a < 20 && flag1 == 0) {
flag1 = 1;
if (flag2 == 0) { person = person + 1; }
}
if (dis_b < 20 && flag2 == 0) {
flag2 = 1;
if (flag1 == 0) { person = person - 1; }
}
if (dis_a > 20 && dis_b > 20 && flag1 == 1 && flag2 == 1) {
flag1 = 0, flag2 = 0;
delay(1000);
}
lcd.setCursor(0, 0);
lcd.print("Have Person: ");
lcd.print(person);
Serial.println(person);
lcd.print(" ");
lcd.setCursor(0, 1);
lcd.print("Light is ");
if (person > 0) {
digitalWrite(relay, LOW);
lcd.print("On ");
} else {
digitalWrite(relay, HIGH);
lcd.print("Off");
}
}
```



6.8.1. Introduction

This code is an Arduino-based project that utilizes ultrasonic sensors and an LCD to detect the presence of people in a room and control a light relay accordingly. Below is the detailed analysis of the code components, explanation of key elements, and a summary.

6.8.2. Code Analysis

Section 1: Library Inclusion and Initialization

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);
```

- **Purpose**: Includes the LiquidCrystal library to interface with the LCD.
    - **Usage**: Initializes the LCD with specific pin connections.
    - **Reason**: LCD is used for displaying messages such as the number of persons detected and light status.

Section 2 : Pin Definitions

```
#define e_s1 A0 // echo pin
#define t_s1 A1 // Trigger pin
#define e_s2 A2 // echo pin
#define t_s2 A3 // Trigger pin
int relay = 8; // Out for light
```

- **Purpose**: Defines pins for ultrasonic sensors and relay.
    - **Usage**: Ensures clarity in assigning specific pins for echo, trigger, and relay.

Section 3 : Global Variables

```
long dis_a = 0, dis_b = 0;
int flag1 = 0, flag2 = 0;
int person = 0;
```

- **Purpose**:
    - dis_a, dis_b: Store distance measured by the sensors.
    - flag1, flag2: Track state transitions for sensors.
    - person: Tracks the number of people in the room.

Section 4 : Ultrasonic Sensor Reading Function

```
void ultra_read(int pin_t, int pin_e, long &ultra_time) {
    long time;
    pinMode(pin_t, OUTPUT);
    pinMode(pin_e, INPUT);
    digitalWrite(pin_t, LOW);
    delayMicroseconds(2);
    digitalWrite(pin_t, HIGH);
    delayMicroseconds(10);
    time = pulseIn(pin_e, HIGH);
    ultra_time = time / 29 / 2;
}
```

- **Purpose**: Reads the distance measured by the ultrasonic sensor.
    - **Explanation**:
        - Triggers the ultrasonic sensor by sending a high pulse for 10 microseconds.
        - Measures the echo time using pulseIn and converts it to distance.

Section 5 : Setup Function

```
void setup() {
    Serial.begin(9600);
    pinMode(relay, OUTPUT);
    lcd.begin(16, 2);
    lcd.setCursor(0, 0);
    lcd.print(" Welcome ");
    delay(1000);
    lcd.clear();
}
```

- **Purpose**: Configures pins, initializes LCD, and sets up communication with the serial monitor.

Section 6 : Main Loop

- **Distance Reading**

```
ultra_read(t_s1, e_s1, dis_a);
ultra_read(t_s2, e_s2, dis_b);
```

Reads the distance from both ultrasonic sensors.

- **Logic for Person Count**

```
if (dis_a < 20 && flag1 == 0) { flag1 = 1; if (flag2 == 0) { person++; }}
if (dis_b < 20 && flag2 == 0) { flag2 = 1; if (flag1 == 0) { person--; }}
```

Updates the person count based on sensor activations.

- **Reset Flags**

```
if (dis_a > 20 && dis_b > 20 && flag1 == 1 && flag2 == 1) {
    flag1 = 0; flag2 = 0;
    delay(1000);
}
```

Resets sensor state after a detection cycle.

- **LCD and Relay Update**

```
lcd.setCursor(0, 0);
lcd.print("Have Person: ");
lcd.print(person);
if (person > 0) { digitalWrite(relay, LOW); lcd.print("On "); }
else { digitalWrite(relay, HIGH); lcd.print("Off"); }
```

Displays the number of persons and toggles the light relay based on occupancy.

6.8.3. Summary of the Code File

This program integrates sensors, LCD, and relay to automate room lighting based on occupancy detection. The ultrasonic sensors detect movement, updating a count of people entering or leaving the room. The LCD displays this count and the status of the light.

6.8.4. Key Code Highlights and Explanation

Code Highlight	Explanation
`LiquidCrystal lcd(2, 3, 4, 5, 6, 7);`	Initializes the LCD with specific pins for communication.
`pulseIn(pin_e, HIGH)`	Measures the time for which the echo pin is HIGH, corresponding to the time taken by the ultrasonic wave to travel back.
`person = person + 1;`	Increments the person count if the first sensor detects motion but not the second, indicating entry.
`digitalWrite(relay, LOW);`	Turns the relay ON when at least one person is detected.
`lcd.print("Have Person: ");`	Displays the current number of persons on the LCD.

6.8.5. Frameworks and Libraries Used

1. **LiquidCrystal Library**

- **Definition**: A library for interfacing with LCDs using the HD44780 driver.
- **Reason for Usage**: Simplifies communication with the LCD.

Code References:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);
lcd.begin(16, 2);
lcd.print(" Welcome ");
```

2. **Arduino Core Framework**

- **Definition**: Provides APIs for serial communication, pin operations, and delays.
- **Reason for Usage**: Facilitates hardware-software integration.

Code References:

```
Serial.begin(9600);
pinMode(pin_t, OUTPUT);
delayMicroseconds(10);
digitalWrite(relay, LOW);
```

6.8.6. How the Code Works

1. **Setup Phase**
   - Initializes the LCD, relay pin, and serial communication.
   - Displays a welcome message on the LCD.
2. **Loop Execution**
   - **Step 1**: Reads distances from two ultrasonic sensors.
   - **Step 2**: Updates the person count based on sensor activation sequence.
     - **Entry**: First sensor triggered before the second.
     - **Exit**: Second sensor triggered before the first.
   - **Step 3**: Displays the updated person count on the LCD.
   - **Step 4**: Toggles the relay to control the light based on the count.
3. **Key Logic**
   - Ensures proper reset of sensor flags after a detection cycle.
   - Prevents false triggers with a minimum distance threshold (20 cm).

6.8.7. Final Summary

This program efficiently automates light control based on room occupancy using ultrasonic sensors and an LCD. The modular approach, with a dedicated function for ultrasonic readings, makes the code reusable and easy to debug.

---

7. Why Our Project is Unique

a. **Comprehensive Feature Set:** Combines multiple functionalities, from emotion recognition to proctored exams.
b. **Focus on Engagement:** Enhances student-teacher interaction through advanced tools.
c. **Energy Efficiency:** Employs IoT for intelligent resource management.
d. **Real-Time Insights:** Provides actionable data to educators for better decision-making.

---

8. Challenges Faced and Solutions

a. **Challenge:** Integration of multiple features into a single platform.
   Solution: Modular design to ensure scalability and smooth integration.
b. **Challenge:** High computational requirements for real-time processing.
   Solution: Optimized machine learning models for faster inference.
c. **Challenge:** Maintaining user privacy while using cameras.
   Solution: Secure data handling and encryption protocols.

---

9. Key Achievements

1. Developed a fully functional **Smart Classroom prototype.**
2. Implemented **real-time emotion recognition** with 90% accuracy.
3. Successfully integrated IoT sensors for **automated lighting .**

---

10. Motivation

The motivation for this project stems from the need to address challenges in traditional classrooms, such as student disengagement, inefficient attendance systems, and lack of interactivity. By leveraging technology, we aim to create a **more inclusive and adaptive learning environment.**

---

11. Objectives

a) Enhance classroom interactivity using advanced tools.
b) Automate mundane tasks like attendance.
c) Improve student safety and engagement through monitoring.

---

12. Problem Statements

a. **How to ensure active student participation in class?**
b. **How to minimize resource wastage, such as energy?**
c. **How to address the growing need for personalized learning experiences?**

---

13. Project Plan

1. **Phase 1:** Research and prototyping of individual features.
2. **Phase 2:** Integration of components into a unified system.
3. **Phase 3:** Testing in a real-world environment and gathering feedback.

---

14. Design/Architecture

The system follows a **modular design** to allow scalability.

- **Data Collection Module:** Captures video streams and motion data.
- **Processing Module:** Executes ML models for detection and recognition tasks.
- **Output Module:** Displays insights and facilitates interaction.

---

15. Implementation Details

- **Programming Languages:** Python, JavaScript.
- **Libraries Used:** TensorFlow, OpenCV, Flask, MediaPipe.
- **Hardware Used:** Cameras, IoT sensors, projectors.

---

16. Learnings

- Developed expertise in **machine learning and IoT integration.**
- Gained hands-on experience in **Python frameworks** like Flask.
- Understood the importance of **user-centric design** in educational technology.

---

17. Conclusion

In conclusion, our embedded project represents a comprehensive leap forward in the realm of smart environments and interactive systems. Our system integrates a range of advanced features designed to enhance user experience, security, and engagement.

The automatic light control feature leverages motion detection to ensure optimal energy usage by turning lights on or off based on the presence of individuals, contributing to both convenience and sustainability. This not only improves user comfort but also promotes energy conservation.

Our smart attendance system employs facial recognition technology to accurately identify and record attendance, eliminating manual processes and reducing administrative overhead. This enhances efficiency and accountability within educational or organizational settings.

The virtual board, keyboard, and mouse functionalities offer innovative alternatives to traditional input methods, providing users with more flexible and intuitive ways to

interact with digital content. These features cater to diverse user needs and preferences, promoting accessibility and adaptability.

Furthermore, our emotion detection and drowsiness detection capabilities provide valuable insights into student engagement and well-being. By analyzing facial expressions and detecting signs of fatigue, we can gauge student satisfaction and alertness, facilitating more responsive teaching practices and fostering a conducive learning environment.

Overall, our project demonstrates the potential of embedded systems to revolutionize daily routines, enhance learning experiences, and contribute to a smarter, more connected future.

***