Facial Recognition Attendance System for Our Faculty Students

R.A.R. Ranawaka
Index No: 21/ENG/108
Registration No: EN102863
Department of Electrical and Electronics Engineering
University of Sri Jayewardenepura

***Abstract*—This project proposes automated student attendance based on the use of a facial recognition system. The facial recognition system also incorporates a database, image processing, a user interface, and CSV logging. The `face_recognition` saves facial attributes, while the webcam records video streams, allowing for the identification of students and their attendance-taking in real-time.**

SQLite is used for managing the database which is containing students information and the record of their attendances. OpenCV is responsible for image processing and serves for real-time video streaming and face recognition. It uses the Tkinter framework to enable registration of student details. The records of attendance are also kept in the CSV format to make analysis easier and more conclusive.

This concerns Python libraries and packages such as OpenCV, NumPy, `face_recognition`, SQLite3, CSV, Tkinter, and Pillow. Both development and testing processes were conducted using PyCharm software and a webcam. The above system enhances efficiency, accuracy and documentation is done electronically which makes it suitable for universities.

1. Project Problem

The objective of this project is to facilitate and enhance the efficiency of real-time attendance for our faculty students. The traditional process of calling names or using paper sheets is extremely slow and inaccurate. Facial recognition will make the proposed system easier to identify students and record their attendance, which will be faster and more efficient. This saves teachers time, eliminates human intervention and enables accuracy in retrieving and updating attendance records. Furthermore, it also helps in keeping the attendance record which is managed and available in digital format.

2. Short Description of the Solutions

2.1. Face Recognition

It uses the `face_recognition` library built on top of advanced machine learning algorithms for the face recognition process. This morphological feature is an actual scan and digital photograph of each student's face, which creates a set of numbers that correspond to the student. This coding is then kept in a database which is then used for identification processes of the person. When the student is in front of the webcam, the application compares the live facial data with the stored facial codes to identify the student.

2.2. Database Management

It is mainly used for data management and storage and its database is mostly in SQLite. It consists of tables for student data, including name, department, face code, and tables for attendance, index number and date. This well-defined structure enables one to efficiently file all data related to students and attendance, easily access and revise the data in case of any changes required. The system will update its database time as new students register or take attendance.

2.3. Image Processing

Real-time video capture from a webcam is done using OpenCV, an open source tool for computer vision. The library analyzes the frames of the video and searches for the face and extracts some information from the face and checks that the information on the face matches with the coding set. Usage reduces their size and converts them to RGB format to improve face detection and make the system work efficiently under different lighting conditions.

2.4. User Interface

As it consists of various programs, Tkinter, a standard GUI library for Python, is used to make the GUI easier to understand. The specialized interface enables the registration of new students; By adding new records and including details of students and their photos. It also offers choices on how to mark a visit. The GUI works to be visually appealing, and users will be able to interface the system without needing to be computer experts.

2.5. CSV Logging

The register is kept in a csv file as this format is quite simple and universally accessible for review of entries and information. Each record in the CSV file contains the index number, student's name, department and the time when his/her attendance was recorded. They are important for establishing a paper trail for attendance data, especially when such information needs to be exported and analyzed as deemed necessary.

2.6. Real-Time Operation

Using video recording and facial recognition, the system can mark the student's presence as soon as this person is recognized. This cuts down on entry time and minimizes the potential for manual entry of such records. For faculty, the system displays a real-time video feed while updating attendance records in real-time, saving time for the process.

3. List of Resources Used

3.1. Libraries and Modules

a) **OpenCV (`cv2`)**

OpenCV is a library that has been well-developed for computer vision and pretty much any image-related task. In this project, cv2 is used first for capturing the video of the webcam and second, for the process of image identification of a face. The functions of the library allow for frame acquisition and manipulation in real time, which is required for the purpose of attendance checking by recognizing the faces in the frames.

b) **NumPy (`numpy`)**

Namely, NumPy is considered an obligatory package for scientific computations using Python. MATLAB supports the use of array, and matrix as types of data structure plus a set of mathematical functions to manipulate these structures. Working with face encoding in this project is made easier by using NumPy to manage and manipulate data. They make it possible to carry out mathematically based operations on large chunks of data as is necessary for matching faces and recognizing a person.

c) **face_recognition**

The kind of tool used in this work is called `face_recognition` and it is relatively easy to use but very effective. It is based on the 'dlib' and helps to work with the high-quality face detection and recognition tools easily. In this project, it is used to convert faces from images into encodings and use the similar encodings identify people. Regarding facial recognition, the library helps to detect faces in real time from the video frames from the webcam.

d) **SQLite3 (`sqlite3`)**

SQLite3 is a kind of lightweight data management engine that is employed in case of data storing and maintaining in file-based databases. In this project SQLite3 is used to created a database in order to store the details of the users and their attendance record. The 'sqlite3' module is used to carry processes that are associated with the SQL database including creation of tables, entry of record and data retrieval.

e) **CSV (`csv`)**

The 'csv' module in Python is employed for working with CSV files; for reading from specific file and writing into another file. Notably, this module comes with several built-in capabilities, which are most effective when dealing with tabular data; in this particular project, the module is used to handle users and attendance data. The 'csv' is useful for exporting and importing data structure because records can easily be kept and checked. In my project I used two .csv files,

- **Registered_Users.csv:** A record of all registered users, their names, index number, among others.
- **Attendance.csv:** Logs attendance records with the student names, index number, and timestamps.

f) **Tkinter (`tkinter`)**

Tkinter is one of the most used graphic toolkit in python used for the creation of simple GUI applications in python. For the development of graphical user interface in this facial attendance system Tkinter is employed. This encompasses developing windows, frames as well as buttons for some of the interactions that may include registration as well as marking of attendance.

g) **Pillow (`PIL` or `Pillow`)**

Pillow or PIL (Python Imaging Library) is a tool that allows for opening, including, and exporting different sorts of images. In this project, Pillow is employed to carry out image operations within the Tkinter like the display of images of users when registering. It permits size changes, and image display, which is crucial in the user interface's graphical characteristics.

3.2. Additional Tools and Libraries

a) **datetime**: They offer classes that are used to manipulate date and time. Employed when creating timestamps to mark people's attendance especially at official functions.

b) **random**: Develops pseudo-random number generators. Enlarge the GUI and make a starry background which moving.

c) **filedialog and messagebox from Tkinter**: Control file

selection dialogs and announce messages in GUI, in sequence.

3.3. Development Tools

- **Python IDE (PyCharm)**: I wrote and tested all the code in PyCharm. PyCharm is integrated development environment that is designed for Python programming language and it aids with coding by prompting the user with string suggestions and error checking and that way makes coding easier and faster.

- **Webcam**: Facial recognition was enabled by use of a webcam to feed live video from the camera to the application. It enables the identification of users' face against the background and record them in one go, which is key to tracking attendance.

3.4. Image Files

- **Background Image for Registration Window** This image is used within the Tkinter GUI to enhance the registration interface.

Figure. Refer FIGURE 1

Figure 1. Background image for Registration Window

4. Implementation Proof

4.1. Code

```
import cv2
import numpy as np
import face_recognition
import sqlite3
import csv
from tkinter import *
from tkinter import filedialog, messagebox
from tkinter.ttk import Combobox
from datetime import datetime, date
from tkinter.font import Font
import random
from tkinter import Toplevel, Label, Tk, Frame,
    Entry
from PIL import Image, ImageTk

#Database setup
conn = sqlite3.connect('User_Database.db')  #
    Connect to the SQLite database
c = conn.cursor()

#Create table
c.execute('''CREATE TABLE IF NOT EXISTS users
             (id INTEGER PRIMARY KEY, name TEXT,
    department TEXT, batch TEXT,
             birthday TEXT, registration_no TEXT,
    index_no TEXT, encoding BLOB)''')
conn.commit()

def create_main_menu():
    main_menu = Tk()
    main_menu.title("Facial Attendance System")
    main_menu.geometry("800x600")  #Set window
    size

    #Create the background
    main_menu.configure(bg='#000000')  #Setting
    background color to black

    stars = []

    def create_star(x, y, size, color):
        star = Label(main_menu, text='*', font=('
    Arial', size), fg=color, bg='#000000')
        star.place(x=x, y=y)
        stars.append(star)

    def toggle_stars():
        for star in stars:
            star.place_forget()  #Hide all stars

        main_menu.after(500, show_stars)  #Toggle
    visibility after 500 milliseconds

    def show_stars():
        for star in stars:
            star.place(x=random.randint(0, 800), y
    =random.randint(0, 600))  #Randomize star
    positions
            star.config(font=('Arial', random.
    randint(10, 20)))  #Randomize star size
            star.config(fg=random.choice(['white',
     'yellow', 'blue', 'orange','red']))  #
    Randomize star color

        main_menu.after(500, toggle_stars)  #
    Toggle visibility after 500 milliseconds

    #Create stars using labels with different
    colors and sizes
    for _ in range(100):  #Increased number of
    stars
        x = random.randint(0, 800)  #Adjusted for
    the enlarged window
        y = random.randint(0, 600)  #Adjusted for
    the enlarged window
        size = random.randint(10, 20)  #Larger
    size range
        color = random.choice(['white', 'yellow',
    'blue', 'orange'])
        create_star(x, y, size, color)

    label_heading = Label(main_menu, text="Facial
    Attendance System", font=('Rockwell', 36, '
    italic', 'bold'), fg='#FFFFFF', bg='#000000')
     # Increased font size
    label_heading.pack(pady=100)  #Increased
    padding

    button_frame = Frame(main_menu, bg='#000000')
    button_frame.pack(pady=40)  #Increased padding

    button_font = Font(family='Arial', size=18,
    weight='bold')  #Button font

    #Toggle variable to track button state
    button_state = False
```

```
    def on_enter(button):
        nonlocal button_state
        if button_state:
            button.config(bg="#FFA500")  #Orange color
            button_state = False
        else:
            button.config(bg="#4169E1")  #Royal blue color
            button_state = True

    def on_leave(button):
        button.config(bg="#FFA500" if button_state else "#4169E1")  #Adjusted colors

    #Create buttons
    for i, (text, command) in enumerate(zip(["New User Registration", "Mark Attendance"], [open_registration_window, open_attendance_window])):
        button = Button(button_frame, text=text, command=command, font=button_font, bg="#4169E1", fg='white', bd=0, padx=30, pady=15)  #Adjusted button size and colors
        button.grid(row=i, column=0, padx=20, pady=10, sticky="ew")  #Adjusted padding
        button.bind("<Enter>", lambda e, b=button: on_enter(b))
        button.bind("<Leave>", lambda e, b=button: on_leave(b))

    main_menu.after(500, toggle_stars)

    main_menu.mainloop()

def create_user_registration_window():
    registration_window = Toplevel()
    registration_window.title("User Registration")
    registration_window.geometry("600x650")
    #Create a Frame for the image at the top
    image_frame = Frame(registration_window)
    image_frame.pack(side="top", fill="both", expand = True)
    #Load and display the image
    image_path = "C:/Users/ICS/PycharmProjects/pythonProject1/.venv/facer1.png"  #Path to image
    image = Image.open(image_path)
    #Resize the image
    new_size = (600,240)  #Specify the new size
    image = image.resize(new_size, Image.NEAREST)  #Use a basic resizing filter
    photo = ImageTk.PhotoImage(image)


    #Create a Label to hold the image
    image_label = Label(registration_window, image=photo)
    image_label.image = photo
    image_label.pack(side="top", fill="both", expand = True)


    #Create a Frame for the registration form below the image
    form_frame = Frame(registration_window)
    form_frame.pack(side="top", fill="both", expand=True)

    #Add user details to the database.
    def add_user_to_database(name, department, batch, birthday, registration_no, index_no, encoding):
        c.execute(
            "INSERT INTO users (name, department, batch, birthday, registration_no, index_no, encoding) VALUES (?, ?, ?, ?, ?, ?, ?)",
            (name, department, batch, birthday, registration_no, index_no, encoding))
        conn.commit()

   #Encode the face from the given image path
    def encode_faces(image_path):
        img = cv2.imread(image_path)
        if img is None:
            print(f"Error: Unable to read image file '{image_path}'")
            return None
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        face_locations = face_recognition.face_locations(img_rgb)
        if len(face_locations) == 0:
            print("Error: No face detected in the image.")
            return None
        face_encodings = face_recognition.face_encodings(img_rgb, face_locations)
        return face_encodings[0]

    #Open a file dialog to select an image and display it
    def browse_image():
        global selected_image_path
        selected_image_path = filedialog.askopenfilename()
        #Display the selected image
        if selected_image_path:
            img = cv2.imread(selected_image_path)
            img_resized = cv2.resize(img, (400, 300))  #Resize image to 400x300
            cv2.imshow("Selected Image", img_resized)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

    #Register the user by adding their details to the database.
    def register_user(entry_name, entry_department, entry_batch, entry_birthday, entry_registration_no, entry_index_no):
        name = entry_name.get()
        department = entry_department.get()
        batch = entry_batch.get()
        birthday = entry_birthday.get()
        registration_no = entry_registration_no.get()
        index_no = entry_index_no.get()
        #Check if all fields are filled
        if not (name and department and batch and birthday and registration_no and index_no):
            warning_message = "Please fill in all fields."
            messagebox.showwarning("Incomplete Data", warning_message)
            print(warning_message)
            return

        global selected_image_path
        if selected_image_path is None:
            warning_message = "Please select an image."
```

```
            messagebox.showwarning("No Image Selected", warning_message)
            print(warning_message)
            return

        encoding = encode_faces(selected_image_path)
        if encoding is None:
            warning_message = "Registration failed. No face detected."
            messagebox.showwarning("Registration Failed", warning_message)
            print(warning_message)
            return

        add_user_to_database(name, department, batch, birthday, registration_no, index_no, encoding.tobytes())
        success_message = "User registered successfully."
        messagebox.showinfo("Registration Successful", success_message)
        print(success_message)
        #Update the CSV file with the new user data
        write_users_to_csv()

    font_label = ('Helvetica', 14, 'bold')
    font_entry = ('Helvetica', 14)
    color_button = '#4682B4'

    labels = ["Name:", "Department:", "Batch:", "Birthday:", "Registration No:", "Index No:"]
    entries = [StringVar() for _ in range(6)]

    title_label = Label(registration_window, text="New User Registration", font=('Helvetica', 29, 'bold'))

    #Create and pack the title label in a Frame to span the entire width
    title_frame = Frame(image_frame, bg="lightblue")
    title_frame.pack(side="top", fill="x", pady=(0,0))
    title_label = Label(title_frame, text="New User Registration", font=('Helvetica', 29, 'bold'), bg="lightblue")
    title_label.pack(fill="x")

    for i, label_text in enumerate(labels):
        frame = Frame(registration_window)
        frame.pack(pady=5, padx=10, fill=X)
        label = Label(frame, text=label_text, font=font_label, width=12, anchor='w')
        label.pack(side=LEFT, padx=(0, 5))
        if i == 1:
            department_options = ['EE', 'CO', 'ME', 'CE']
            entry = Combobox(frame, textvariable=entries[i], values=department_options, font=font_entry, state="readonly")
        elif i == 2:
            batch_options = ['Batch 4', 'Batch 5', 'Batch 6', 'Batch 7', 'Batch 8']
            entry = Combobox(frame, textvariable=entries[i], values=batch_options, font=font_entry, state="readonly")
        else:
            entry = Entry(frame, textvariable=entries[i], font=font_entry)
        entry.pack(side=LEFT, fill=X, expand=True)

    def write_users_to_csv():
        c.execute("SELECT name, index_no, registration_no, department, birthday, batch FROM users")
        users_data = c.fetchall()

        if users_data:
            with open('Registered_Users.csv', 'w', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(
                    ["Name", "Index No", "Registration No", "Department", "Birthday", "Batch"])  # Header row
                writer.writerows(users_data)
            print("Registered users data has been written to 'Registered_Users.csv'.")
        else:
            print("No user data found to write.")

    button_browse = Button(registration_window, text="Browse Image", command=browse_image, bg=color_button, fg='white', font=font_label)
    button_browse.pack(pady=10)

    #Submit Button
    button_submit = Button(registration_window, text="Submit", command=lambda: register_user(*entries), bg=color_button, fg='white', font=font_label)
    button_submit.pack(pady=10)



def create_attendance_table():
    c.execute('''CREATE TABLE IF NOT EXISTS attendance (
                    id INTEGER PRIMARY KEY,
                    index_no TEXT,
                    date TEXT
                )''')
    conn.commit()

#Create and display the attendance marking window
def create_attendance_marking_window():
    #Mark attendance for the user
    def mark_attendance(name):
        c.execute("SELECT index_no, department FROM users WHERE name = ?", (name,))
        data = c.fetchone()
        if data:
            index_no, department = data
            today = date.today()
            today_str = today.strftime('%Y-%m-%d')
            #Add attendance to the database
            c.execute("SELECT * FROM attendance WHERE index_no = ? AND date = ?", (index_no, today_str))
            if c.fetchone():
                message = f"Attendance for {name} ({index_no}) has already been marked today."
                print(message)

            else:
                now = datetime.now()
                dt_string = now.strftime('%Y-%m-%d %H:%M:%S')
                with open('Attendance.csv', 'a')
```

```
    as f:
                    f.write(f"{index_no},{name},{department},{dt_string}\n")
                print(f"Attendance marked for {name} ({index_no}), {department} at {dt_string}.")
                c.execute("INSERT INTO attendance (index_no, date) VALUES (?, ?)", (index_no, today_str))
                conn.commit()
        else:
            print(f"No user found with the name {name}.")

        def update_status(message):
            status_label.config(text=message)

    def load_known_encodings():
        c.execute("SELECT name, department, batch, encoding FROM users")
        data = c.fetchall()
        names = [row[0] for row in data]
        departments = [row[1] for row in data]
        batches = [row[2] for row in data]
        encodings = [np.frombuffer(row[3]) for row in data]
        return names, departments, batches, encodings

    def main():
        cap = cv2.VideoCapture(0)

        while True:
            success, img = cap.read()
            img_s = cv2.resize(img, (0, 0), None, 0.25, 0.25)
            img_s = cv2.cvtColor(img_s, cv2.COLOR_BGR2RGB)

            faces_cur_frame = face_recognition.face_locations(img_s)
            encode_cur_frame = face_recognition.face_encodings(img_s, faces_cur_frame)

            for encode_face, face_loc in zip(encode_cur_frame, faces_cur_frame):
                matches = face_recognition.compare_faces(encode_list_known, encode_face)
                face_dis = face_recognition.face_distance(encode_list_known, encode_face)
                match_index = np.argmin(face_dis)

                if matches[match_index]:
                    name = names[match_index]
                    department = departments[match_index]
                    batch = batches[match_index]
                    print(name, department, batch)
                    y1, x2, y2, x1 = face_loc
                    y1, x2, y2, x1 = 4 * y1, 4 * x2, 4 * y2, 4 * x1
                    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
                    cv2.rectangle(img, (x1, y2 + 60), (x2, y2), (0, 255, 0), 2, cv2.FILLED)
                    cv2.putText(img, f"Name: {name}", (x1 + 6, y2 + 15), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
                    cv2.putText(img, f"Department: {department}", (x1 + 6, y2 + 30), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
                    cv2.putText(img, f"Batch: {batch}", (x1 + 6, y2 + 50), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
                    mark_attendance(name)

            cv2.imshow('Webcam', img)
            cv2.waitKey(2)

    create_attendance_table()
    names, departments, batches, encode_list_known = load_known_encodings()
    main()

def open_registration_window():
    create_user_registration_window()

def open_attendance_window():
    create_attendance_marking_window()

create_main_menu()
```

Listing 1. Python Code for Face Recognition Attendance System

4.1.1. Brief Overview of the Code. The complete code is about Facial Attendance System created in Python with the uses of OpenCV, SQLite for database, and various libraries for GUI. Here is a general description of major components and their purpose.

- **Database Setup**

  This introduces the script by firstly creating an SQLite database by the name of `User_Database.dB`. It also raises a relation and builds a `users` table for storing details of customers, identification data, and face prints. Another table with the name `attendance` is defined in order to store the attendance information.

- **Main Menu Creation**

  The `create_main_menu` function sets up the main interface of the application, the graphing capability of which is provided by Tkinter. This GUI has the background of a dynamic star animation that makes the interface more attractive. The menu includes two buttons: In addition to the above fundamental features, it was convenient to have one portal for the registration of the users and another portal for attendance.

- **User Registration Window**

  The user registration window is created by using `create_user_registration_window` function. This allows the user to fill in details and load a picture on an input form. The photo is processed with the `face_recognition` library to obtain face encodings, which are saved to the database along with the user details. The CSV file is named as `Registered_Users.csv`. It is used to update csv with the new data of the users hence ease in managing its data.

- **Attendance Marking Window**

  The last function to be mentioned is the `create_attendance_marking_window` which

is in charge of marking of attendance. It employs the webcam to record live video feed and identify faces with the help of the textttface_recognition library. In this case, the bio-metrics confirm to an entry in a database thus tabulating the attendance of the particular user. Attendance records are logged in a CSV file named 'Attendance.csv' and also updated in the database.

- **Encoding and Recognition**

  The system loads face encodings from the database to recognize faces during the attendance marking process. The main function in the `create_attendance_marking_window` captures live video from the webcam, processes it to detect faces, and compares them with stored encoding to identify users. Recognized users are marked as present, and their attendance is recorded.

4.2. Outputs

A) **Main Window**

The main window of the 'Facial Attendance System' developed through Tkinter module is the Home window through which the user gains access to the application. The interface looks somewhat professional by including what can be referred to as a dynamic star animation in the background. The background is black with stars splattered on it in various sizes and colors and the stars blink giving it a dynamic feel. On the top of the form, there is an appropriate heading that has been entered, and it is 'Facial Attendance System'. Below this heading, there are two essential buttons: Two of the buttons on the home page, namely 'New User Registration' and 'Mark Attendance', will take the users directly to the intended functionality of the application.
Figure. Refer FIGURE 2

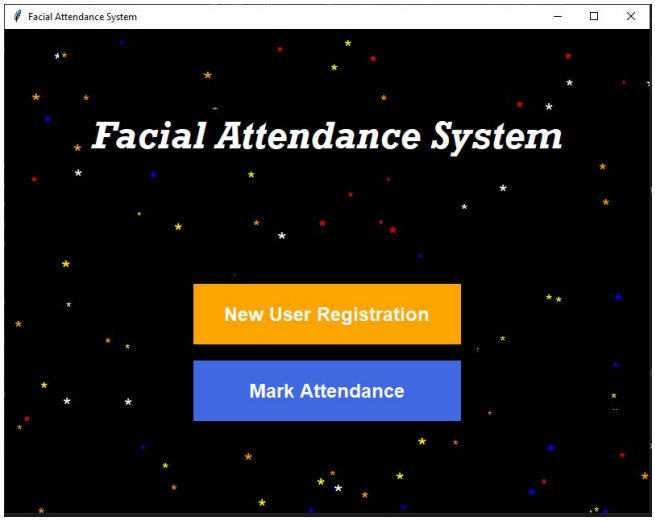

Figure 2. Main Window

B) **Registration Form**

After clicking the 'New User Registration' button in the main window, the registration form can be seen. The registration form is a separate form of data filling for creating a new account. It has text boxes where users can enter information like their name, department, group, date of birth, registration number and index number. Department and batch inputs have selection options. Departmental section has four options ('EE', 'CO', 'ME', 'CE'). The batch section has five options to select the user's batch. Furthermore, there is the user's choice of selecting and encoding an image and its intended use for facial recognition. A form layout is used with a header section that includes an image placeholder; Input fields for user information, 'Browse Image' button to select user image and 'Register' button to create a user. Also, the user can view the selected image in a new window.
Figure. Refer FIGURE 3

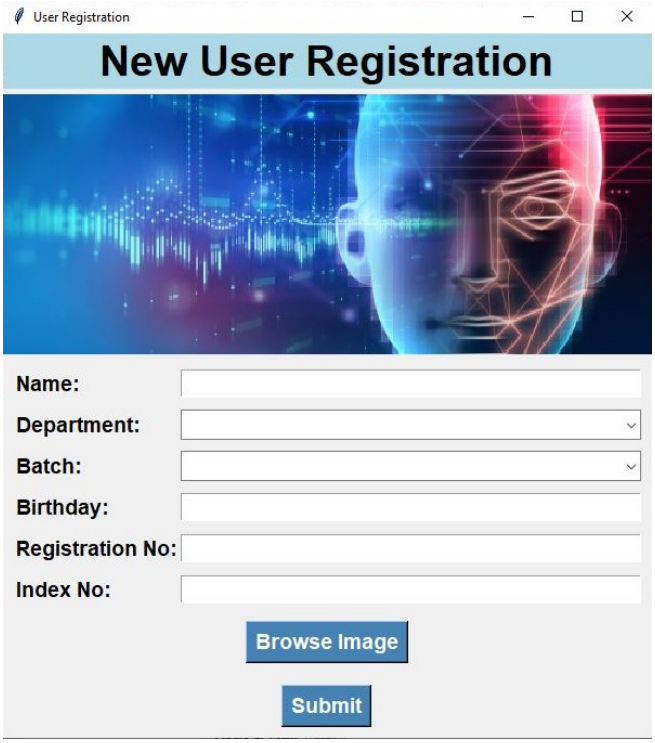

Figure 3. New User Registration Window

Figure. Refer FIGURE 4

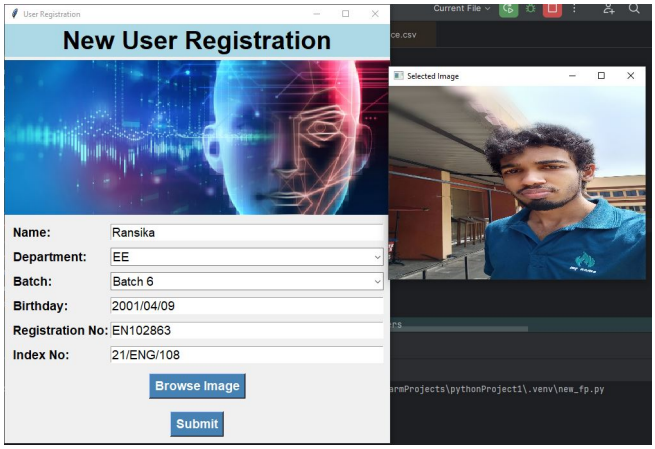

Figure 4. Registering a User

After successful registration, if the message box 'User registered successfully' appears, if the picture fields are left blank, or if a picture is selected but the face is not recognized, this message will appear 'Please fill all fields,' 'Please select a picture,' or 'Registration failed. Faces are not recognized respectively.
Figure. Refer FIGURE 5

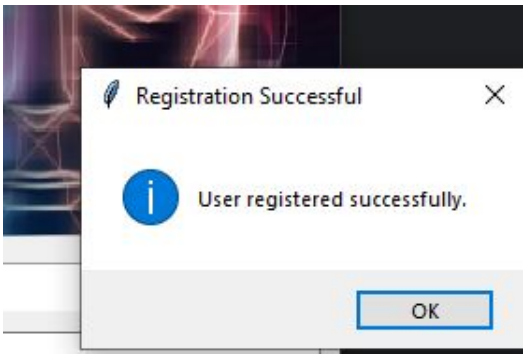

Figure 5. Registration success message

Figure. Refer FIGURE 6

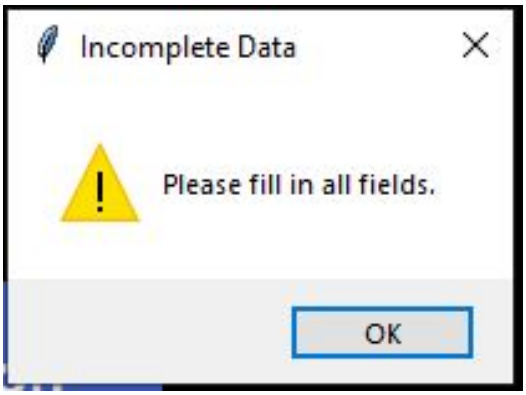

Figure 6. Incomplete data message

Figure. Refer FIGURE 7

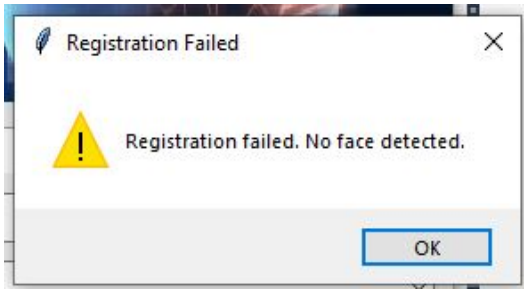

Figure 7. No face detected message

C) **CSV Files** The system is responsible for producing two .csv files `Registered_Users.csv` and `Attendance.csv`. Specifically, `Registered_Users.csv` file has details about registered users like their name, index number, registration number, department they belong to, date of birth and group. In `Attendance.csv`, it tracks index number, name, department and time stamp.
Figure. Refer FIGURE 8

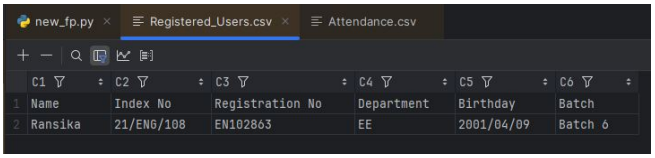

Figure 8. Registered Users csv

Figure. Refer FIGURE 9

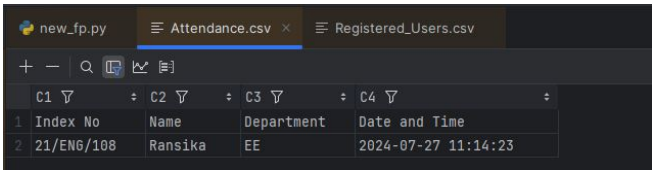

Figure 9. Attendance csv

D) **Webcam Interface**

Live video is captured in real-time by the OpenCV webcam interface while simultaneously recognizing human faces; The system works by comparing live face encodings with stored encodings of registered users. The webcam uses a window to display the feed, meaning that after a face is detected, a rectangle containing text elements such as the person's name, department, or team appears alongside it.
Figure. Refer FIGURE 10

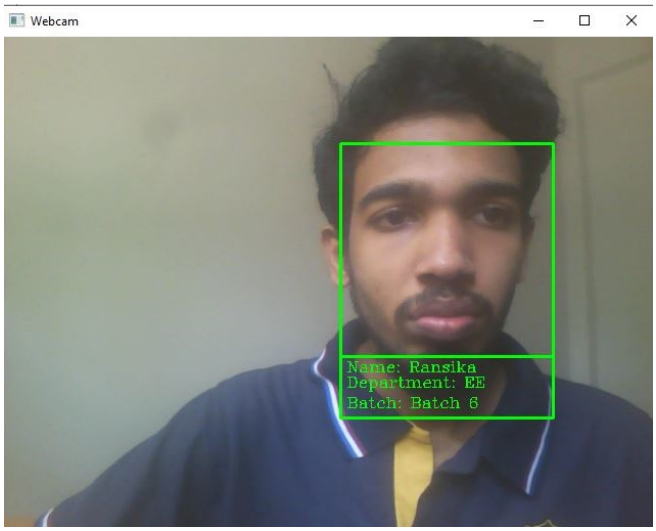

Figure 10. Webcam Interface

If the attendance is marked successfully, a message will be displayed in the command window "Attendance marked for [name] ([index number]), [department] at [date time]". If a user's presence is already marked for the date, the message box "Attendance for [Name] ([Index Number]) is already marked today.". In cases where no user matching the live face encryption was found, the system returned the message "No user found with name [name]" in the command window.
Figure. Refer FIGURE 11

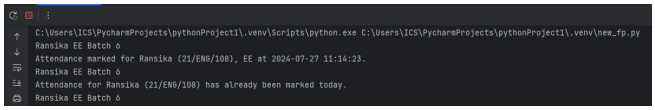

Figure 11. Attendance marked message

4.3. Evidence of Implementation

This code operates several servers in a facial attendance system, mainly the extraction of face encodings from images and face recognition based on such encodings.

4.3.1. Getting Face Encodings from an Image. To obtain face encodings, the `encode_faces` function performs the following steps:

I. Loading the Image

The image is loaded from a specified file path using OpenCV's *cv2.imread()* function, which loads the image into an array in the programming language for numerical computations, numpy.

```
def encode_faces(image_path):
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error: Unable to read image
    file '{image_path}'")
        return None
```

II. Converting the Image Format

The loaded image, initially in BGR format, is first converted to RGB using the command *cv2.cvtColor()* that will suit the `face_recognition` library.

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

III. Detecting Faces

The *face_recognition.face_locations()* function detects the faces in the RGB image and returns their coordinates.

```
face_locations = face_recognition.
    face_locations(img_rgb)
    if len(face_locations) == 0:
        print("Error: No face detected in the
    image.")
        return None
```

IV. Encoding Faces

Once the facial locations are recognized, *face_recognition.face_encodings()* encodes these faces into a numeric representation. This encoding is a 128-dimensional feature vector that uniquely represents the face.

```
face_encodings = face_recognition.
    face_encodings(img_rgb, face_locations)
return face_encodings[0]
```

4.3.2. Identifying Faces Using Encodings. The process of identifying faces involves comparing the encoded face from a live video feed with known face encodings stored in a database:The process of identifying faces involves comparing the encoded face from a live video feed with known face encodings stored in a database.

I. Loading Known Encodings

The `load_known_encodings` function retrieves face encodings and associated user information from the database, converting the binary encoding data back into NumPy arrays.

```
def load_known_encodings():
    c.execute("SELECT name, department, batch,
     encoding FROM users")
    data = c.fetchall()
    names = [row[0] for row in data]
    departments = [row[1] for row in data]
    batches = [row[2] for row in data]
    encodings = [np.frombuffer(row[3]) for row
     in data]
    return names, departments, batches,
    encodings
```

II. Capturing Video

A live video feed can be captured by utilizing OpenCV in python through the *cv2.VideoCapture(0)* which access the default camera.

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

III. Processing Each Frame

Every frame from the video feed is then resized and then converted to an RGB format. Moving facial features in the current frame are detected and encoded in a manner similar to the one in the static image processing.

```
while True:
    success, img = cap.read()
    img_s = cv2.resize(img, (0, 0), None, 0.25, 0.25)
    img_s = cv2.cvtColor(img_s, cv2.COLOR_BGR2RGB)

    faces_cur_frame = face_recognition.face_locations(img_s)
    encode_cur_frame = face_recognition.face_encodings(img_s, faces_cur_frame)
```

IV. Comparing Encodings

Then, the current face encoding is compared with the known encodings using the *face_recognition.compare_faces()* and *face_recognition.face_distance()*. There is an attempt to match the face of the unknown person to one of the registered ones, the closest match is identified based on the smallest face distance.

```
for encode_face, face_loc in zip(encode_cur_frame, faces_cur_frame):
    matches = face_recognition.compare_faces(encode_list_known, encode_face)
    face_dis = face_recognition.face_distance(encode_list_known, encode_face)
    match_index = np.argmin(face_dis)
```

V. Displaying Results

If a match is found, the name, department, and batch of the person are displayed on the video feed using OpenCV's *cv2.putText()*. The frame is obtained and then a rectangle drawn around the face that was recognized using *cv2. rectangle()*.

```
if matches[match_index]:
    name = names[match_index]
    department = departments[match_index]
    batch = batches[match_index]
    print(name, department, batch)
    y1, x2, y2, x1 = face_loc
    y1, x2, y2, x1 = 4 * y1, 4 * x2, 4 * y2, 4 * x1
    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
    cv2.rectangle(img, (x1, y2 + 60), (x2, y2), (0, 255, 0), 2, cv2.FILLED)
    cv2.putText(img, f"Name: {name}", (x1 + 6, y2 + 15), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
    cv2.putText(img, f"Department: {department}", (x1 + 6, y2 + 30), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
    cv2.putText(img, f"Batch: {batch}", (x1 + 6, y2 + 50), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 0), 1)
    mark_attendance(name)
```

References

[1] J. Minichino and J. Howse, *Learning OpenCV 3 computer vision with python*. Packt Publishing Ltd, 2015.

[2] J. Howse, P. Joshi, and M. Beyeler, *Opencv: computer vision projects with python*. Packt Publishing Ltd, 2016.

[3] P. Joshi, *OpenCV with Python by example*. Packt Publishing Ltd, 2015.

[4] A. D. Moore, *Python GUI Programming with Tkinter: Design and build functional and user-friendly GUI applications*. Packt Publishing Ltd, 2021.

[5] S. U. Bukhari. (2022) Face recognition in python: A beginner's guide. Accessed: 2024-07-27. [Online]. Available: https://sesamedisk.com/face-recognition-in-python-a-beginners-guide/

[6] A. Geitgey. (2016) Machine learning is fun! part 4: Modern face recognition with deep learning. Accessed: 2024-07-27. [Online]. Available: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc12

[1] [2] [3] [4] [5] [6]

5. Viewer Link for Overleaf Document

You can view the Overleaf document using this link: https://www.overleaf.com/read/snfqfcrsdyhh#3c5c4e