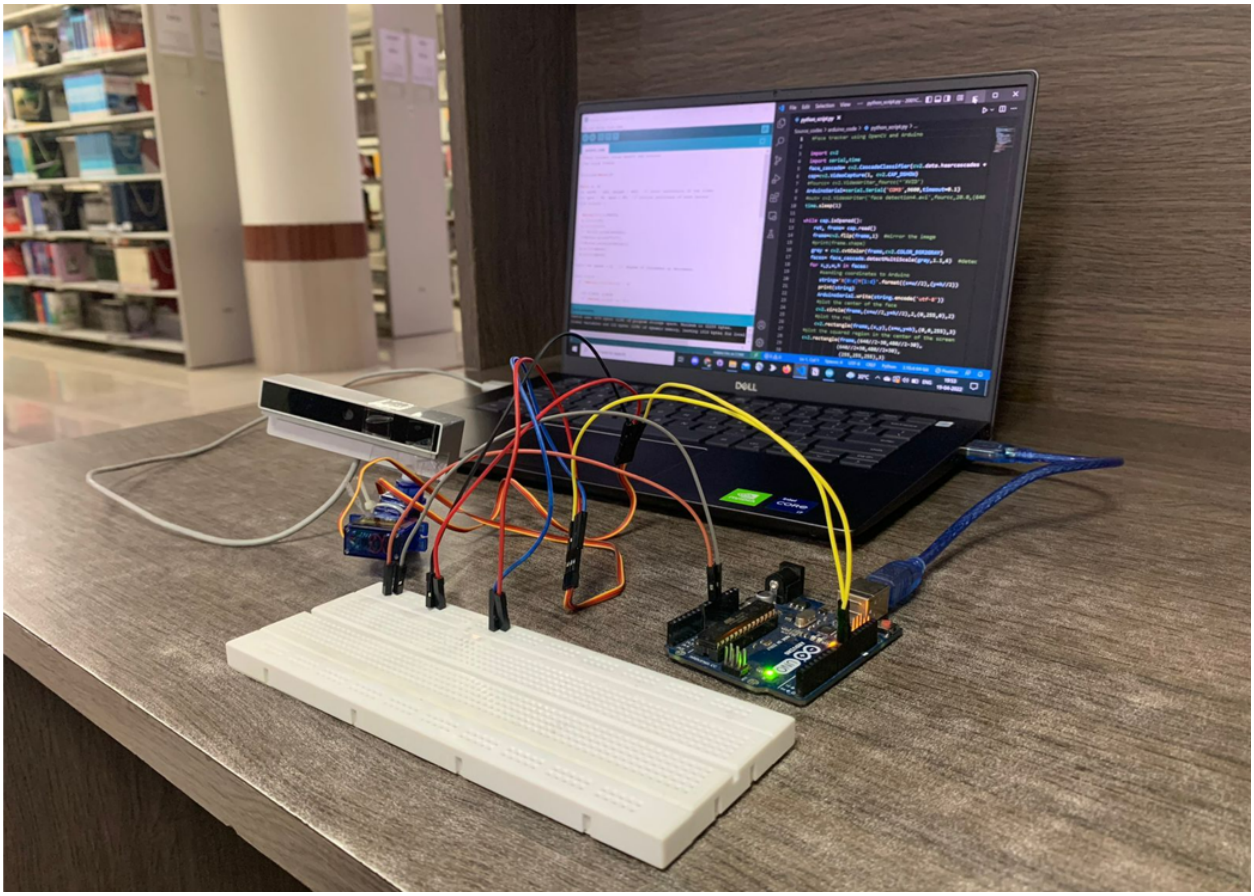


CS225/226 Project Report

TOPIC: FACE DETECTION AND TRACKING USING ARDUINO UNO + FACE RECOGNITION BASED ATTENDANCE SYSTEM

Name: Palak Totala

Roll No: 2001CS84

Date: 20-04-2022

Overview

Face detection, often known as facial recognition, is a computer technology that uses artificial intelligence (AI) to recognize and identify human faces in digital images. Face detection technology can be used in a variety of industries to enable real-time surveillance and tracking of people, including security, biometrics, law enforcement, entertainment, and personal safety.

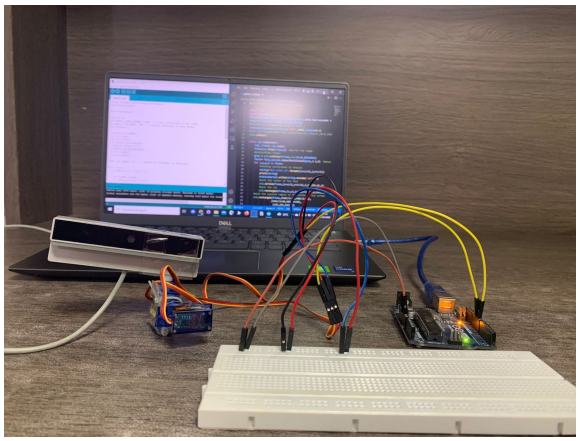
Face identification has developed from basic computer vision approaches to innovations in machine learning (ML) and associated technologies, resulting in ongoing performance gains. It now serves as a crucial first step in a variety of critical applications, including face tracking, face analysis, and facial recognition. Face detection has a substantial impact on the application's ability to perform consecutive activities.

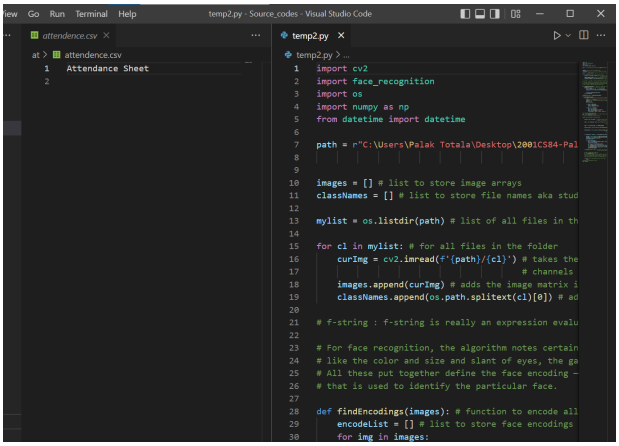
In face analysis, face detection helps identify which parts of an image or video should be focused on to determine age, gender, and emotions using facial expressions. In a facial recognition system -- which maps an individual's facial features mathematically and stores the data as a faceprint -- face detection data is required for the algorithms that discern which parts of an image or video are needed to generate a faceprint. Once identified, the new faceprint can be compared with stored faceprints to determine if there is a match, which can be called face recognition.

Goals

In this project, I have worked towards the following two targets:

1. To create an Arduino model that can detect and track faces.
2. To build an attendance system based on face recognition technology.



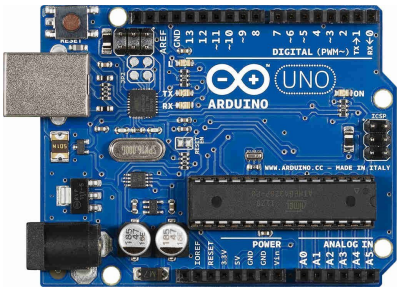


1. Face detector and tracker using Arduino UNO

Background

Facial recognition is a very helpful tool that is integrated into many modern gadgets to detect human faces for tracking, biometric, and activity recognition. I utilized OpenCV's Harr cascade classifiers to recognize human faces and a dual servo mechanism to track the user's face using an Arduino UNO in this project.

Components Used

COMPONENT NAME	IMAGE	DESCRIPTION
ARDUINO UNO		Arduino UNO is a low-cost, flexible, and easy-to-use open-source programmable microcontroller board that can be used in a wide range of electronic applications. This board can operate relays, LEDs, servos, and motors as output and can be interfaced with other Arduino boards, Arduino shields, and Raspberry Pi boards. The Arduino UNO has an Atmega328 AVR microprocessor, six analog input pins, and 14 digital I/O pins, six of which are utilized for PWM output.
USB CAMERA		A USB webcam is a camera that connects to a computer by inserting it into one of the machine's USB ports. The video is transmitted to the computer, where a software program allows you to view and transfer the images to the Internet.

SERVO MOTORS (SG90) (2)		Servo motors are high-torque motors that are extensively employed in robotics because their rotation is simple to control. A geared output shaft on a servo motor can be electrically controlled to turn one (1) degree at a time. Servo motors, unlike typical DC motors, usually feature a third pin (the signal pin) in addition to the two power pins (Vcc and GND) for control. The signal pin is utilized to turn the shaft of the servo motor to any specified angle.
BREADBOARD		A breadboard is a solderless device used to prototype electronics and test circuit designs on a temporary basis. Most electronic components in electronic circuits can be connected by fitting their leads or terminals into the holes and connecting them with wires where necessary.
JUMPER WIRES (Male-to-Male)		Male-to-male, male-to-female, and female-to-female jumper wires are the most common types of jumper wires. The distinction between them lies in the wire's terminating point. Male ends feature a protruding pin that can be plugged into items, whereas female ends don't have one and are used to plug things onto.

Technologies Used

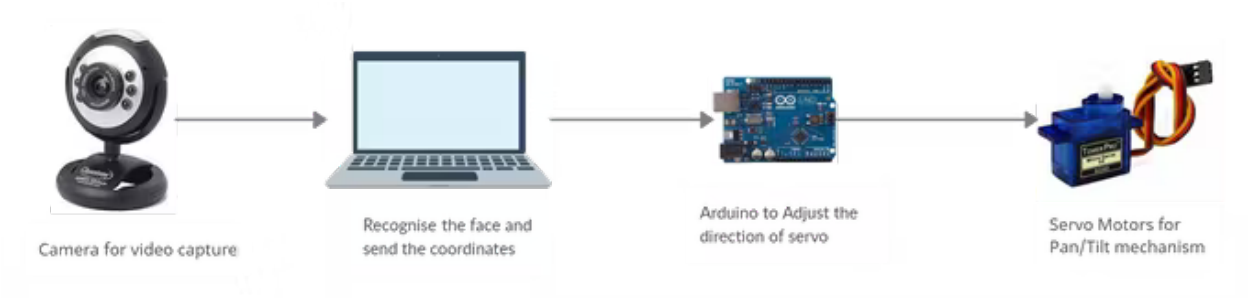
python™	OpenCV	ARDUINO

Outline of the Code

Human faces are identified and localized using facial detection, which ignores any backdrop objects such as curtains, windows, trees, and so on. OpenCV employs a Harr cascade of classifiers, in which each frame of video is sent through a series of classifiers; if the frame passes through all of them, the face is present; otherwise, the frame is thrown from the classifier, indicating that the face was not detected.

When an image is detected, OpenCV returns the cartesian coordinates, as well as the height and width. x+width/2 and y+height/2 can be used to calculate the image's center coordinates from these coordinates.

When the face is detected, these coordinates are sent to the Arduino UNO using the pyserial library. The camera is attached to one of the servos connected to the Arduino, which provides a pan/tilt mechanism. When the face's coordinates are off from the center, the servo will align by 2 degrees (increment or decrement) to bring it closer to the screen's center.

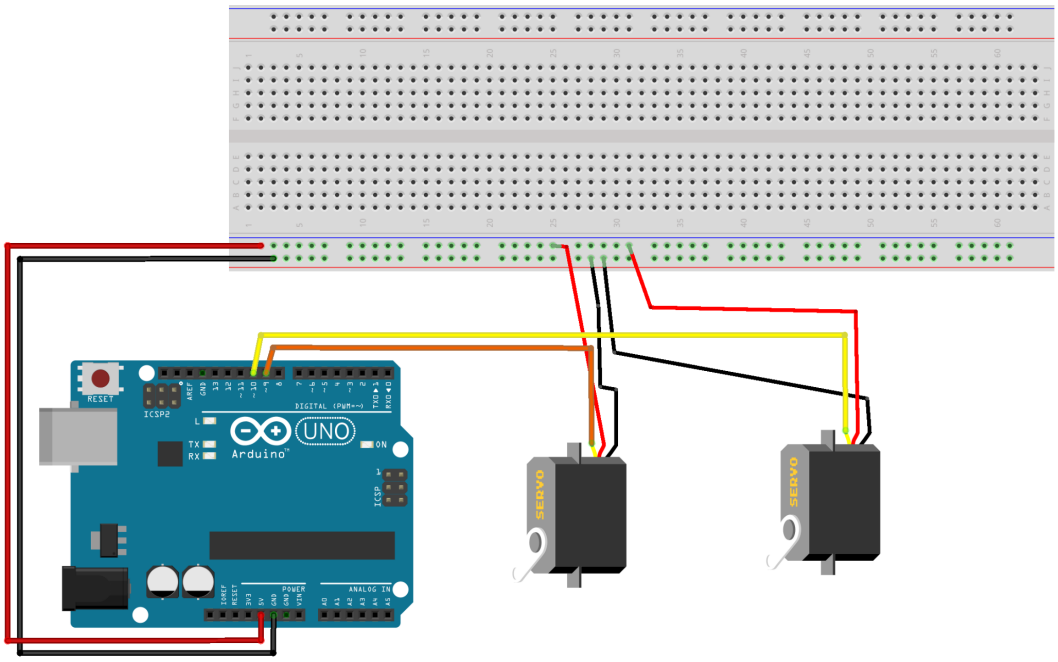


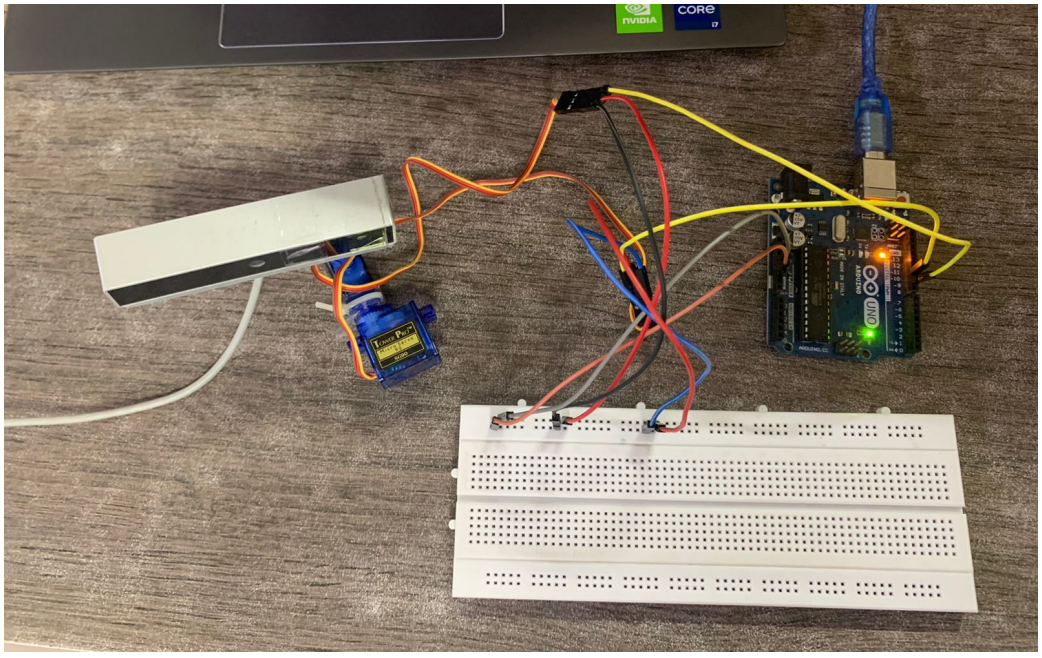
Installations and Libraries

This project includes the use of the pyserial and OpenCV libraries, which can be downloaded using pip. I employed 'haarcascade frontalface default.xml,' a pre-trained model for recognizing human faces that can be downloaded as an XML file from Github. Also, make sure that the face detection XML file is saved in the same directory as the python script.

Before executing the python script, a minor modification (in line 9) must be made by entering the correct COM port of the Arduino.

Circuit Diagram and Arrangement





Attach two servos to Arduino.

- Vertical to Pin 9
- Horizontal to Pin 10
- Power to +5V
- Ground to GND

Code Analysis

1. ARDUINO CODE

(a) Void setup

Technically, void setup is a function that you construct at the start of each program. The code that you wish to run once as soon as the program starts is enclosed in curly brackets. This is where you set things like pinMode.

arduino_code

```
#include<Servo.h>

Servo x, y;
int width = 640, height = 480;  // total resolution of the video
int xpos = 90, ypos = 90;  // initial positions of both Servos
void setup() {

  Serial.begin(9600);
  x.attach(9); // servo x is attached to digital pin 9
  y.attach(10); // servo y is attached to digital pin 10
  x.write(xpos); // sets the angle of the shaft of servo x (90 degrees), moving the shaft to that orientation.
  y.write(ypos); // sets the angle of the shaft of servo y (90 degrees), moving the shaft to that orientation.
}
const int angle = 2;    // degree of increment or decrement
```

We define the two servos as x and y. We initialize their position at 90 degrees. Servo x is attached to digital pin 9 while Servo y is attached to digital pin 10. We define a constant angle of 2 degrees which is the degree of rotation of servo when the center of the face is not within a certain boundary.

(b) Void loop

The loop is yet more feature that Arduino incorporates into its design. As long as the Maker Board is powered on, the code inside the loop function executes over and over.

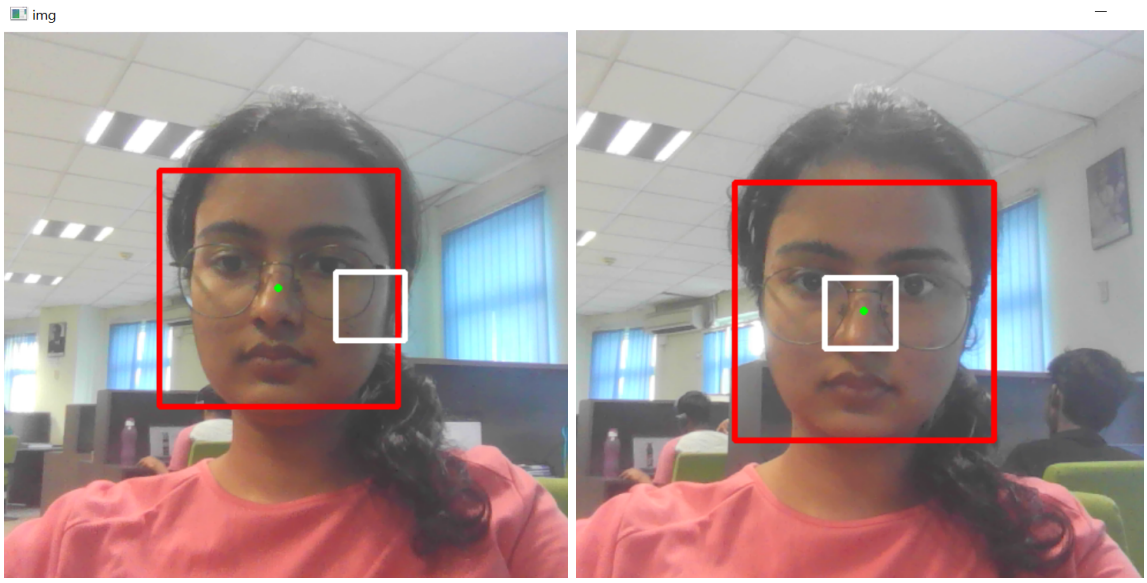
```
void loop() // loop function that executes repeatedly
{
  if (Serial.available() > 0) // It will only send data when the received data is greater than 0.
  {
    int x_mid, y_mid; // initialising variables to store center coordinates
    if (Serial.read() == 'X') // reads out the first available byte from the serial receive buffer. After it reads it out, it removes that byte from the buffer.
    {
      x_mid = Serial.parseInt();  // read center x-coordinate (scans down the serial receive buffer one byte at a time in search of the first valid numerical digit)
      if (Serial.read() == 'Y')
        y_mid = Serial.parseInt(); // read center y-coordinate
    }
```

Python sends the center coordinates in a single string.

To understand this part, let us take an example: Let the incoming value of the coordinates of the center of the face be "X150Y300".

The value 150 after X represents the center x-coordinate and 300 represents the center y-coordinate.
Serial.read() will read the first byte in the string (i.e. X), and the string that remains will be “150Y300”. Since the condition holds true, Serial.parseInt() will find the first valid numerical digit in the string, which is 150 and x_mid will hold this value. The string that remains is “Y300”. Now, Serial.read() will read Y and the remaining string (“300”) is stored in y_mid using parseInt.

This way, we receive the center coordinates in the form of a string from python and read the string to store the coordinates respectively in x_mid and y_mid.



After receiving the center coordinates, we begin detecting and tracking the face.

The white square in the middle of the frame denotes the area inside which the green dot, which is the center of the face, must be. When the face is moved outside the squared zone, the servo will align the camera to bring it inside the region.

This is done using the subsequent code:

```
  // adjust the servo within the squared region if the coordinates is outside it

  if (x_mid > width / 2 + 30)
    xpos += angle;
  if (x_mid < width / 2 - 30)
    xpos -= angle;
  if (y_mid < height / 2 + 30)
    ypos -= angle;
  if (y_mid > height / 2 - 30)
    ypos += angle;


  // if the servo degree is outside its range

  if (xpos >= 180)
    xpos = 180;
  else if (xpos <= 0)
    xpos = 0;
  if (ypos >= 180)
    ypos = 180;
  else if (ypos <= 0)
    ypos = 0;

  x.write(xpos);
  y.write(ypos);
}
```

Here,

If the x-coordinate of the center is greater than the center of the width which is the midpoint of the horizontal axis, we add a 2-degree rotation (anticlockwise movement) to reposition the camera. If it is lesser than the center of the width of the video, then we subtract a 2-degree rotation (clockwise movement).

Similarly with the y-coordinates.

If during the above process of addition and subtraction, the position of the servo calculated exceeds 180 degrees or is less than 0 degrees (i.e. servo is expected to go out of range), then the servo position by default is set to 180 and 0 respectively.

2. PYTHON CODE

arduino_code > python_script.py > ...

```python
#Face tracker using OpenCV and Arduino

import cv2
import serial,time
face_cascade= cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
cap=cv2.VideoCapture(1, cv2.CAP_DSHOW)
ArduinoSerial=serial.Serial('COM3',9600,timeout=0.1)
time.sleep(1)

while cap.isOpened():
    ret, frame= cap.read()
    frame=cv2.flip(frame,1)  #mirror the image
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces= face_cascade.detectMultiScale(gray,1.1,6)  #detect the face
    for x,y,w,h in faces:
        #sending coordinates to Arduino
        string='X{0:d}Y{1:d}'.format((x+w//2),(y+h//2))
        print(string)
        ArduinoSerial.write(string.encode('utf-8'))
        #plot the center of the face
        cv2.circle(frame,(x+w//2,y+h//2),2,(0,255,0),2)
        #plot the roi
        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),3)
    #plot the squared region in the center of the screen
    cv2.rectangle(frame,(640//2-30,480//2-30),
                  (640//2+30,480//2+30),
                   (255,255,255),3)
    cv2.imshow('img',frame)
    # press q to Quit
    if cv2.waitKey(10)&0xFF== ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

First, I have imported the OpenCV module, the pyserial module, and the time module. Then I store the haarcascade classifier in the face_cascade variable.

We capture the video from an external camera, hence port 1 and the video source is cv2.CAP_DSHOW in windows. Then we set up Python to send data to the Arduino over the serial port.

We convert the frame color scheme to grayscale for processing.

cv2.CascadeClassifier.detectMultiScale() is the function used to detect faces. With the scale factor value of 1.1 (default) and the 'minNeighbour' value of 6, it is called. This returns the image's cartesian coordinates, as well as its height and width. Increasing the 'minNeighbour' value improves facial detection but reduces execution rates, resulting in a servo response delay. As a result, the number 6 appeared to be the best option.

The face coordinates are returned by OpenCV as pixel values. The video resolution is set to 640*480 by default. The top-left pixel values (x and y), as well as the height and width, are described by the coordinates. For reference, I utilized the face's center coordinates, which can be computed using x+width/2 and y+height/2 and can be seen as a green dot. The Arduino receives these coordinates and uses them to change the camera angle.

2. Attendance monitoring system via face recognition

Background

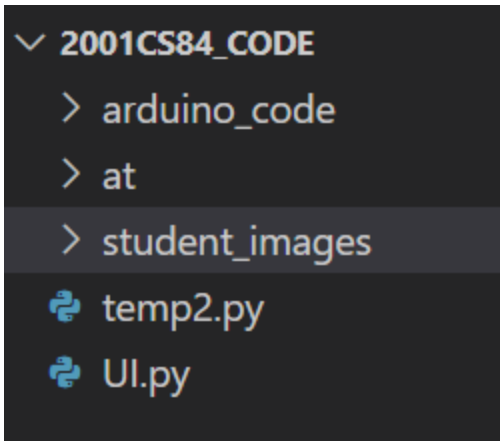
A facial recognition software captures and compares patterns on a person's face, as well as analyses the details, in order to identify and verify the person. Despite the complexity of the underlying mechanism, the entire technique may be boiled down into three steps:

1. **Face Detection**: Locating human faces in real-time is an important step.
2. **Data Transformation**: The analogue facial data is converted into a set of data or vectors based on a person's facial traits once it is taken.
3. **Face Match**: The system verifies the data above by comparing it to the data in the database.

Program Flow

There are **four** components of this module :

1. **“at” folder** - Contains a .csv file which is an attendance sheet. It contains the entries in form of Name, Time of detection, Date of detection.
2. **“student_images” folde**r - Acts as the database for student images. Contains pictures of students with their full name as the file name.
3. **temp2.py file** - Contains the face recognition and attendance marking code.
4. **UI.py file** - Contains the code to a tkinter based simple GUI with two functionalities : Capturing attendance and viewing the attendance sheet.



Code Analysis

1. temp2.py

```python
import cv2
import face_recognition
import os
import numpy as np
from datetime import datetime

path = r"C:\Users\Palak Totala\Desktop\2001CS84-Palak_Project\2001CS84_Code\student_images"    # Path to folder that
                                                                                              # File name is studen

images = [] # list to store image arrays
classNames = [] # list to store file names aka student names

mylist = os.listdir(path) # list of all files in the path folder

for cl in mylist: # for all files in the folder
    curImg = cv2.imread(f'{path}/{cl}') # takes the image as input and decodes into a matrix with the color
                                        # channels stored in the order of Blue, Green, Red, and A respectively
    images.append(curImg) # adds the image matrix in the images list
    classNames.append(os.path.splitext(cl)[0]) # adds the filename in the classNames list

# f-string : f-string is really an expression evaluated at run time, not a constant value (doubt)
```

Here we create two lists, one to store images as arrays and another to store names of the people. We get the names from the corresponding filenames in the path.

For face recognition, the algorithm notes certain important measurements on the face — like the color and size and slant of eyes, the gap between eyebrows, etc. All these put together define the face encoding — the information obtained out of the image — that is used to identify the particular face.

```python
def findEncodings(images): # function to encode all the images and store them in a variable encoded_face_train.
    encodeList = [] # list to store face encodings
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encoded_face = face_recognition.face_encodings(img)[0]  # face_recognition.face_encodings(known_image) basica
                                                                # a list of 128-dimensional face encodings (one for e
                                                                # Now, you are using the index [0] to get the first f
        encodeList.append(encoded_face)
    return encodeList

encoded_face_train = findEncodings(images)
```

This function is used to encode all the images and store them in a variable encoded_face_train. face_recognition.face_encodings(known_image) basically returns a list of 128-dimensional face encodings (one for each face in the image). Now, you are using the index [0] to get the first found face.

```python
def markAttendance(name):
    with open('C:\\Users\\Palak Totala\\Desktop\\2001CS84-Palak_Project\\2001CS84_Code\\at\\attendence.csv','r+') as
                                                                                    # inside the with sta
        myDataList = f.readlines() # returns a list containing each line in the file as a list item
        nameList = [] # A list to store names of students whose attendence is registered in the file
        datelist = []
        #print(nameList)

        for line in myDataList:
            entry = line.split(',')
            nameList.append(entry[0])

        if name not in nameList:
            now = datetime.now()
            time = now.strftime('%I:%M:%S:%p')
            date = now.strftime('%d-%B-%Y')
            f.writelines(f'{name}, {time}, {date}\n')

```

This function is used to update attendance in the csv file. It first reads the file and stores the names of people who have already marked their attendance in a list (nameList). If the name of the currently detected person is not present in the list, i.e. he/she is not re-marking their attendance, then their name, along with time and datestamp is added to the csv file. This ensures that each student gives their attendance only once on a given day.

```python
# To capture video from webcam.
cap  = cv2.VideoCapture(0) #This will return video from the first webcam on your computer.
# To use a video file as input
# cap = cv2.VideoCapture('filename.mp4')

while True: #initiates an infinite loop

    success, img = cap.read()   # Returns a bool (True/False). If the frame is read correctly, it will be True.
                                # Reads the frame and captures an image img.

    imgS = cv2.resize(img, (0,0), None, 0.25,0.25)  # Resize the image by 1/4 only for the recognition part.
                                                    # output frame will be of the original size.
                                                    # Resizing improves the Frame per Second.

    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

    faces_in_frame = face_recognition.face_locations(imgS)  # returns a list of tuples of found face locations in css
                                                            # (top, right, bottom, left) order

    encoded_faces = face_recognition.face_encodings(imgS, faces_in_frame)
                                                    # Parameter 1 : face_image - The image that contains one or more
                                                    # Parameter 2 : known_face_locations - Optional - the bounding bo
                                                    #               each face if you already know them.

```

In this snippet, the video is captured live from the webcam and each frame is read, resized by 1/4th (to make recognition easier), and converted to RGB colour scale. All the faces in the frame are detected and stored in a list (faces_in_frame). Their encodings are stores in the list “encoded_faces”.

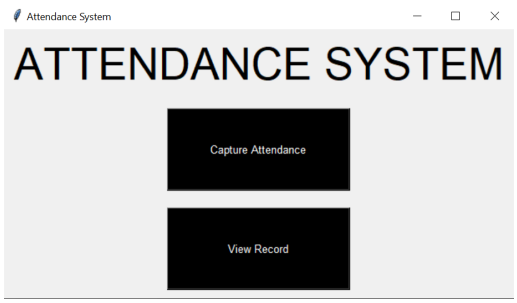
```
    for encode_face, faceloc in zip(encoded_faces,faces_in_frame): # for each encoded face in the frame

        matches = face_recognition.compare_faces(encoded_face_train, encode_face)
        faceDist = face_recognition.face_distance(encoded_face_train, encode_face)
        matchIndex = np.argmin(faceDist)
        print(matchIndex)

        if matches[matchIndex]:
            name = classNames[matchIndex].upper().lower()
            y1,x2,y2,x1 = faceloc
            # since we scaled down by 4 times
            y1, x2,y2,x1 = y1*4,x2*4,y2*4,x1*4
            cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
            cv2.rectangle(img, (x1,y2-35),(x2,y2), (0,255,0), cv2.FILLED)
            cv2.putText(img,name, (x1+6,y2-5), cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
            markAttendance(name)
```

Now for each encoded face in the frame, the face encodings are compared with the encoding data stored from the student database, and if matches, the mark attendance function is called and a ROI (region of interest) is constructed around the face with the identified name displayed in the bottom corner.

```
    cv2.imshow('webcam', img)   # displays the image on a window named webcam, and since there
                                # is an infinite loop of images it is a live video

    # Stop if escape key is pressed
    k = cv2.waitKey(30) & 0xff
    # cv2.waitKey() returns a 32-bit integer that is the ASCII value of the key pressed.
    # 0xFF is a hexadecimal constant which is 11111111 in binary.
    # By using bitwise AND (&) with this constant, it leaves only the last 8 bits of the original
    # We only need last 8 digits since ASCII value of the character can be maximum 255.

    if k==27: #27 is the ASCII value of the esc key.
        break
# Release the VideoCapture object
cap.release()
```

To stop the code after face is recognised, one can click the esc key.

```
from tkinter import *
import tkinter.font as font
import os


root=Tk()
root.title("Attendance System")

def click1():
    os.system("python C:\\Users\\PALAKT~1\\Desktop\\2001CS84-Palak_Project\\2001CS84_Code\\temp2.py")

def click2():
    os.system("C:\\Users\\PALAKT~1\\Desktop\\2001CS84-Palak_Project\\2001CS84_Code\\at\\attendence.csv")


head=Label(root, text="ATTENDANCE SYSTEM", anchor=CENTER, padx=10, pady=10, font=("Arial", 40))
head.grid(row=1, column=1, columnspan=5, sticky="nsew")

op1=Button(root,font=("Arial", 10), text="Capture Attendance", fg="white", bg="black", height="3", pady=20, command=l
op1.grid( row="2", column="3", sticky="nsew",padx=10, pady=10)

op2=Button(root,font=("Arial", 10), text="View Record", fg="white", bg="black", height="3", pady=20, command=lambda:
op2.grid( row="3", column="3", sticky="nsew",padx=10, pady=10)

root.mainloop()
```



(UI)

Scope

Face detection is employed in all facial recognition systems, although not all face detection systems are used for facial recognition. Face detection can also be used for facial motion capture, which is the process of using cameras or laser scanners to turn a human's face motions into a digital database. This database can be used to create lifelike computer animation for films, games, and avatars.

Face detection can also be used to auto-focus cameras or count the number of persons that have entered a room. The technology can also be used for marketing purposes, such as showing targeted adverts when a specific face is detected.

Another application would be a software implementation of emotional inference, which can be used to help persons with autism understand the emotions of those around them. Using powerful image processing, the application "reads" the emotions on a human face.

Drawing verbal conclusions from visual signals, also known as "lip reading," is another application. This could aid computers in determining who is speaking, which could be useful in security applications. Face detection can also be used to decide which areas of an image should be blurred to ensure privacy.

Limitations

1. Detection has a weak point. While face detection produces more accurate results than manual techniques, it is also more vulnerable to changes in appearance or camera angles.

2. There's a lot of data to store. Face detection using machine learning necessitates large amounts of data storage, which may not be available to all users.

3. A possible invasion of privacy. Face detection's ability to assist the authorities in tracking down criminals has significant advantages; nevertheless, the same monitoring can be used to spy on private persons. To ensure that technology is used fairly and in accordance with human privacy rights, certain rules must be established.

4. The attendance system created in this project is very basic and lacks certain functionalities - i.e. There is no automated method to add more students to the database, there is no facility to store attendance for different lectures scheduled for the same day, etc.

Further Extension

1. If a student is absent for more than three classes of a course in a row, an automatic warning can be sent.
2. A cloud based module for adding students to database in realtime can be incorporated
3. Analytics tools regarding attendance percentages could be constructed
4. UI has a lot of scope for improvement
5. For additional security, attendance sheet could be password protected and the software can have a management login page as well as a student login one.
6. The face recognition module can also be used to identity people using images. For example. on campus, incase one finds someone breaking a rule but is unable to identify them, they can take an image of the act and the image can be passed through the recogniser to identify the individual.
7. Given the pandemic, developing an algorithm to identify faces wearing masks will be highly beneficial. This is accomplished by creating algorithms that concentrate on the uncovered areas of the face, such as the eyes, brows, and nose bridge. The algorithms are created using photos of masked faces.

Conclusion

Face recognition technologies provide a quick and secure approach to identify people without having to touch them. Given the quick spread of a virus through touch in today's world, this is a safe strategy.

Face verification accuracy has improved in face recognition systems thanks to advances in computer vision, and the acceptability rate is quite high. Face recognition systems' appeal is bolstered by their fast picture processing speed and ease of setup.

References

1. https://www.rs-online.com/designspark/what-is-arduino-uno-a-getting-started-guide
2. https://www.techtarget.com/searchenterpriseai/definition/face-detection
3. https://create.arduino.cc/projecthub