

# Project Report

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

Name: Palak Totala

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.

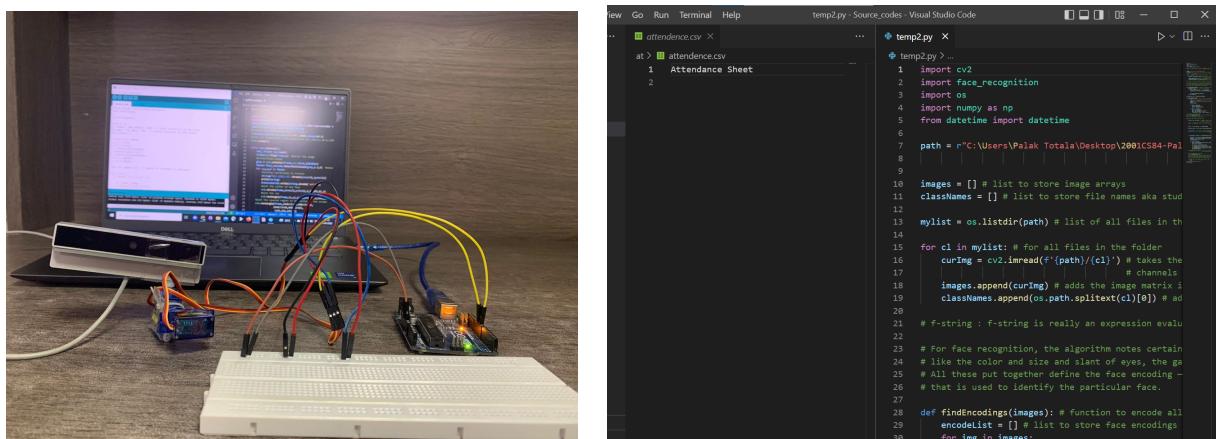
Face identification has developed from basic computer vision approaches to innovations in machine learning (ML) and associated technologies, resulting in ongoing performance gains. It now serves as a crucial first step in a variety of critical applications, including face tracking, face analysis, and 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.



```

New Go Run Terminal Help
attendance.csv X temp2.py - Source codes - Visual Studio Code
at > attendance.csv ...
temp2.py ...
1 import cv2
2 import face_recognition
3 import os
4 import numpy as np
5 from datetime import datetime
6
7 path = r'C:\Users\Palak Totala\Desktop\2001CS84-Palak\Attendance System\Attendance System\Attendance System'
8
9
10 images = [] # list to store image arrays
11 classNames = [] # list to store file names aka stud
12
13 myList = os.listdir(path) # list of all files in th
14
15 for cl in myList: # for all files in the folder
16     curImg = cv2.imread(f'{path}/{cl}') # takes the
17     # channels
18     images.append(curImg) # adds the image matrix i
19     classNames.append(os.path.splitext(cl)[0]) # ad
20
21 # f-string : f-string is really an expression evalu
22
23 # For face recognition, the algorithm notes certain
24 # like the color and size and slant of eyes, the g
25 # All these put together define the face encoding -
26 # that is used to identify the particular face.
27
28 def findEncodings(images): # function to encode all
29     encodeList = [] # list to store face encodings
30     for img in images:

```

## 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	 A photograph of an Arduino Uno microcontroller board. It is a blue printed circuit board with various electronic components, including a central Atmega328 microprocessor, several integrated circuits, and numerous pins and connectors. The board is labeled "ARDUINO" and "UNO".	<p>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.</p> <p>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.</p>
USB CAMERA	 A photograph of a USB camera. It consists of a small black rectangular sensor module attached to a longer white cable with a standard USB-A connector at one end.	<p>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.</p>



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

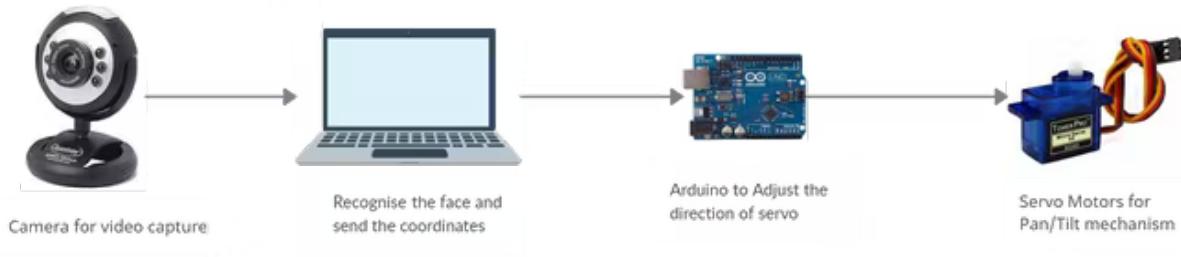


## Outline of the Code

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.

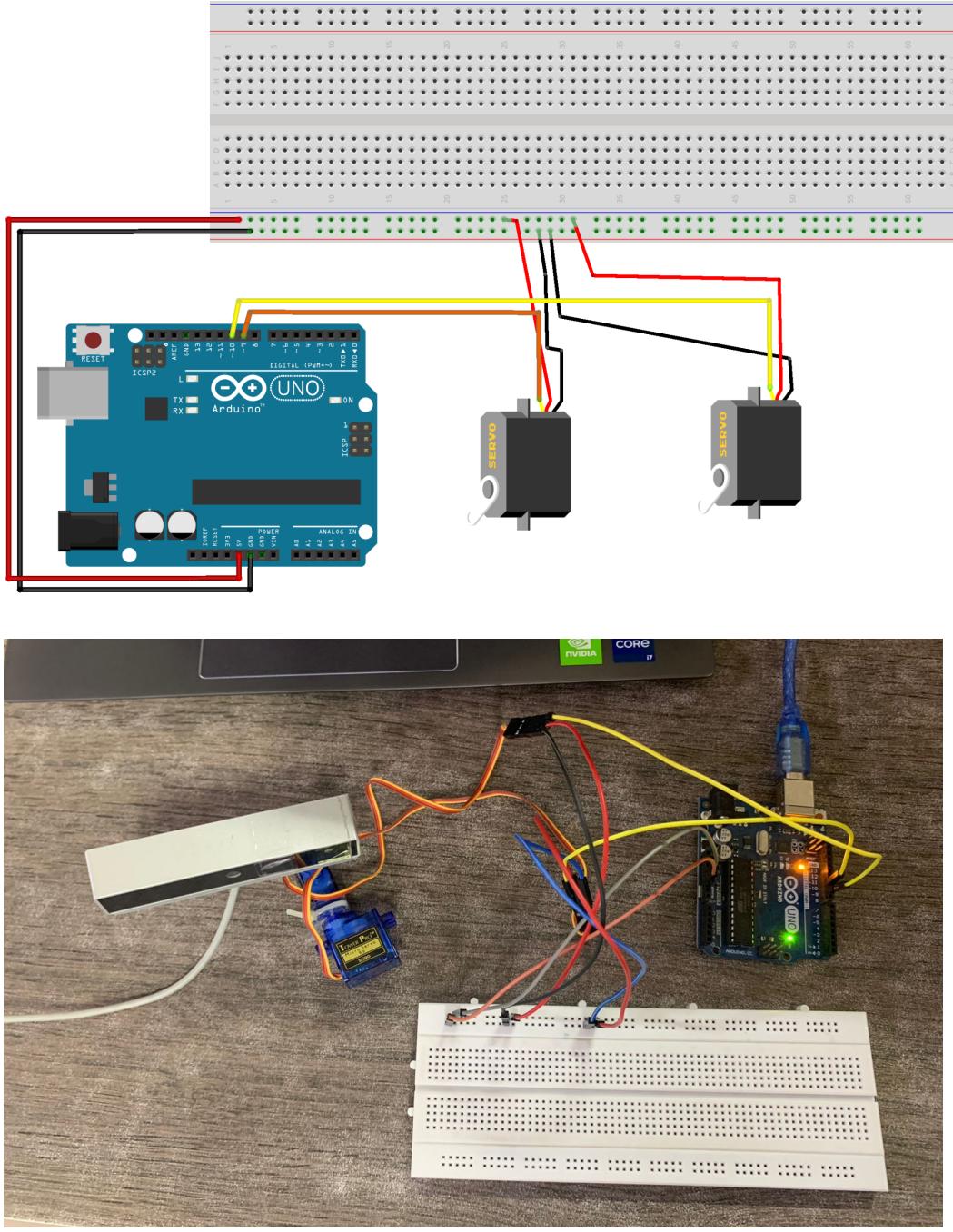


## Installations and Libraries

This project includes the use of the pyserial and OpenCV libraries, which can be downloaded using pip. I employed 'haarcascade frontaleface 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
        }
    }
}
```

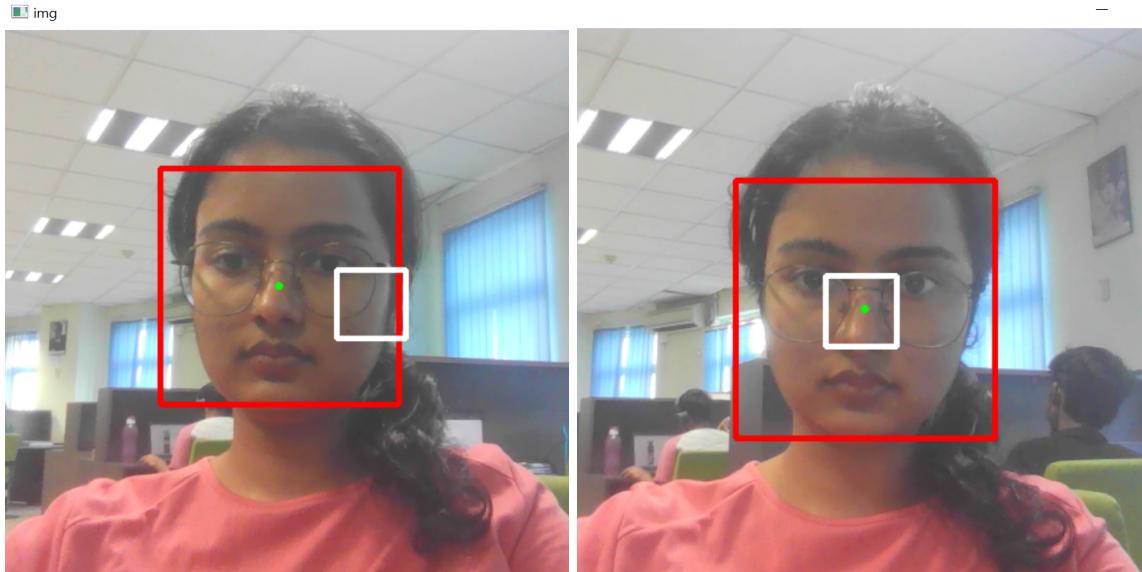
Python sends the center coordinates in a single string.

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 > ...
1  #Face tracker using OpenCV and Arduino
2
3  import cv2
4  import serial,time
5  face_cascade= cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
6  cap=cv2.VideoCapture(1, cv2.CAP_DSHOW)
7  ArduinoSerial=serial.Serial('COM3',9600,timeout=0.1)
8  time.sleep(1)
9
10 while cap.isOpened():
11     ret, frame= cap.read()
12     frame=cv2.flip(frame,1) #mirror the image
13     gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
14     faces= face_cascade.detectMultiScale(gray,1.1,6) #detect the face
15     for x,y,w,h in faces:
16         #sending coordinates to Arduino
17         string='X{0:d}Y{1:d}'.format((x+w//2),(y+h//2))
18         print(string)
19         ArduinoSerial.write(string.encode('utf-8'))
20         #plot the center of the face
21         cv2.circle(frame,(x+w//2,y+h//2),2,(0,255,0),2)
22         #plot the roi
23         cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),3)
24         #plot the squared region in the center of the screen
25         cv2.rectangle(frame,(640//2-30,480//2-30),
26                      (640//2+30,480//2+30),
27                      (255,255,255),3)
28         cv2.imshow('img',frame)
29         # press q to Quit
30         if cv2.waitKey(10)&0xFF== ord('q'):
31             break
32     cap.release()
33     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

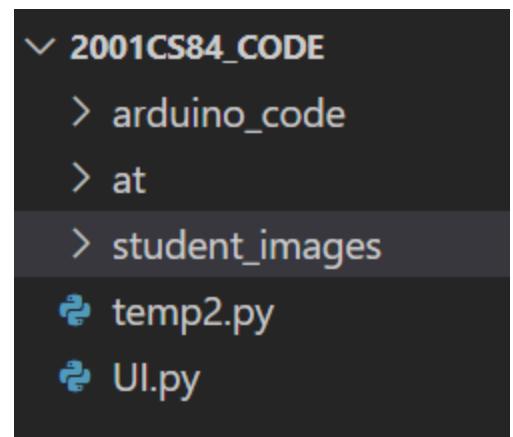
A facial recognition software captures and compares patterns on a person's face, as well as analyses the details, in order to identify and verify the person. Despite the complexity of the underlying mechanism, the entire technique may be boiled down into three steps:

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" folder** - 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

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.

```
28 def findEncodings(images): # function to encode all the images and store them in a variable encoded_face_train.  
29     encodeList = [] # list to store face encodings  
30     for img in images:  
31         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
32         encoded_face = face_recognition.face_encodings(img)[0] # face_recognition.face_encodings(known_image) basically  
33         # a list of 128-dimensional face encodings (one for each face in the image).  
34         # Now, you are using the index [0] to get the first face's encoding.  
35         encodeList.append(encoded_face)  
36     return encodeList  
37  
38 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.

```

40 def markAttendance(name):
41     with open('C:\\\\Users\\\\Palak Totala\\\\Desktop\\\\2001CS84-Palak_Project\\\\2001CS84_Code\\\\at\\\\attendance.csv','r+') as
42         f
43     myDataList = f.readlines() # returns a list containing each line in the file as a list item
44     nameList = [] # A list to store names of students whose attendance is registered in the file
45     datalist = []
46     #print(nameList)
47
48     for line in myDataList:
49         entry = line.split(',')
50         nameList.append(entry[0])
51
52     if name not in nameList:
53         now = datetime.now()
54         time = now.strftime('%I:%M:%S:%p')
55         date = now.strftime('%d-%B-%Y')
56         f.writelines(f'{name}, {time}, {date}\\n')
57

```

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 timestamp is added to the csv file. This ensures that each student gives their attendance only once on a given day.

```

58 # To capture video from webcam.
59 cap = cv2.VideoCapture(0) #This will return video from the first webcam on your computer.
60 # To use a video file as input
61 # cap = cv2.VideoCapture('filename.mp4')
62
63 while True: #initiates an infinite loop
64
65     success, img = cap.read() # Returns a bool (True/False). If the frame is read correctly, it will be True.
66     # Reads the frame and captures an image img.
67
68     imgS = cv2.resize(img, (0,0), None, 0.25,0.25) # Resize the image by 1/4 only for the recognition part.
69     # output frame will be of the original size.
70     # Resizing improves the Frame per Second.
71
72     imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
73
74     faces_in_frame = face_recognition.face_locations(imgS) # returns a list of tuples of found face locations in css
75     # (top, right, bottom, left) order
76
77     encoded_faces = face_recognition.face_encodings(imgS, faces_in_frame)
78     # Parameter 1 : face_image - The image that contains one or more
79     # Parameter 2 : known_face_locations - Optional - the bounding box
80     #               for each face if you already know them.
81

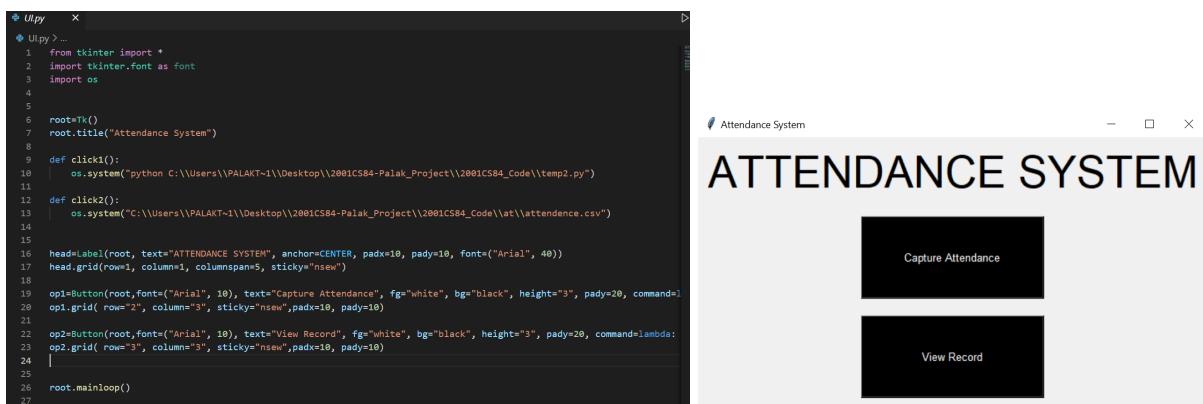
```

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 stored in the list "encoded\_faces".

```
82 for encode_face, faceloc in zip(encoded_faces,faces_in_frame): # for each encoded face in the frame
83
84     matches = face_recognition.compare_faces(encoded_face_train, encode_face)
85     faceDist = face_recognition.face_distance(encoded_face_train, encode_face)
86     matchIndex = np.argmin(faceDist)
87     print(matchIndex)
88
89     if matches[matchIndex]:
90         name = classNames[matchIndex].upper().lower()
91         y1,x2,y2,x1 = faceloc
92         # since we scaled down by 4 times
93         y1, x2,y2,x1 = y1*4,x2*4,y2*4,x1*4
94         cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
95         cv2.rectangle(img, (x1,y2-35),(x2,y2), (0,255,0), cv2.FILLED)
96         cv2.putText(img,name, (x1+6,y2-5), cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
97         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.

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



(U)

## 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 identify 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>