VIDEO (WITH PROCTORING) AND AUDIO TRANSMITTNG USING SOCKET PROGRAMMING AND OPENCV TCP IMPLEMENTATION

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE302: COMPUTER NETWORKS

Submitted by

NAME : JANARTHANAN N

(Reg. No.:123003084, B.Tech(Computer Science Engg.)

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Bonafide Certificate

This is to certify that the report titled "**Video (with Proctoring) and Audio transmitting using Socket Programming and Opencv"** submitted as a requirement for the course, **CSE302: COMPUTER NETWORKS** for B.Tech. is a bonafide record of the work done by **Shri Janarthanan N (Reg. No.123003084, B.Tech Computer Science and Engineering**) during the academic year 2021-22, in the School of Computing

Project-Based Work *Viva-voc*e held on _____

Examiner 1 **Examiner 2**

List of Figures

List of Tables

Abbreviations

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

AI – Artificial Intelligence

OpenCV-Open Source Computer Vision Library

IP – Internet Protocol

LAN- Local Area Network

WAN – Wide Area Network

SYN – Synchronization

ACK – Acknowledgment

XML – Extensible markup language

Abstract

In this modern era, internet and networking are the most common ways of communication and sharing .Due to continuous growth in this area ,developers are developing more internet enabled application. with the help of internet, and with usage of IP addresses, we are able to communicate without any physical connection and with minimal time. This is possible with concept of socket programming. We use this Socket programming in our project for effective Communication between server and client. It is the most widely used interface for transport layer of network services. Sockets establish a connection between two different processes to communicate between them. Here we use TCP/IP protocol for efficient use of sockets since they are more reliable.

Opencv is a concept of AI and it is a great tool for image processing and for performing computer vision tasks. It is a open-source library which can be used to perform tasks like face detection,object tracking etc .We use the concept of face detection in this project for proctoring.We use PyAudio module in python to transmit audio file. . It uses the concept of multithreading so more clients process can be executed at the same time .So the purpose of the project was to transmit multiple Audio And Video from client to server and in addition we will use this Video for proctoring the client from server side using Face And Eye Detection algorithm With the help of python,Opencv, and Socket programming. This proctoring algorithm have interaction between both client and server for efficient communication and the server will store this video for future purpose.

This project will be a healthy solution for Online Proctoring and this project is developed with proper architecture for future enhancement.

Table of Contents

.

CHAPTER 1

INTRODUCTION

Now a days people are widely connected to internet. This project Video(with Proctoring) and audio transmitting using Opencv and Python (TCP implementation).Here people in same LAN can be connected to server for Transmitting and Proctoring. It is made up of two python applications one is client application which is run on client's computer and the other is server application which runs on host computer. It follows a client-server architecture which is based upon request and response. Client will always start the communication by passing a request and the server accepts the request and responds to it.. Here the server can connect to multiple clients at the same time so that multiple execution can be made at the same time. We did in TCP implementation using Socket ,Opencv and, PyAudio.

1.1 TRANSPORT LAYER:

There are mainly 2 protocols in transport layer. They are

1. TCP
2. UDP

TCP is a connection oriented protocol which ensures reliable data transmission between devices. There is a three way hand shake between devices before they transmit and receive the data. UDP is a connection-less protocol that doesn't bother about the reliability and error correction.

Hence in our Project we choose TCP over UDP for reliability

Ports:

For any data to be transferred over a network to communicate it needs a port number. When we run the server with a port number, client should use the same port number to communicate with the server. There are some in-built port numbers which our system uses to run some specific applications. We should not use the port numbers specified in the range 0-1023.

IP Address:

There are two types of IPv4 addresses. They are

1. Private
2. Public

To communicate over the WAN we need public IP address and over the LAN we use private IP address. In this project we run server on local host which is a private IP address.Hence, the clients over LAN only can connect to the server.

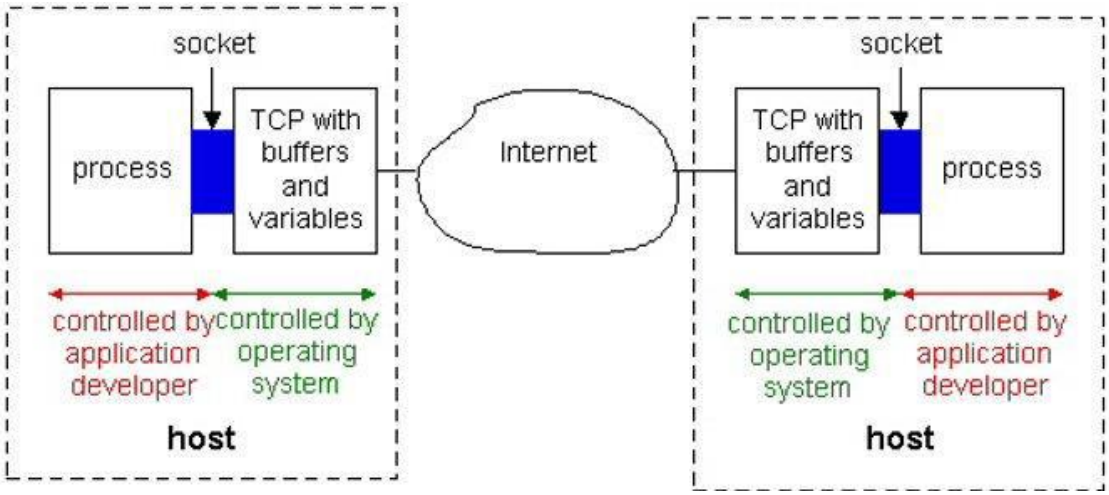


Fig 1.1- Process Communication Through TCP Sockets

1.2 THREE-WAY HANDSHAKE:

It is a process which is used in TCP/IP network to make connection between the server and client. It is a three-step process that requires both the client and server to exchange synchronization and acknowledgment packets before the real data communication process starts. It is designed in such a way that it allows you to transfer multiple TCP socket connections in both the directions at the same time.

. In this TCP handshake process, a client needs to initiate the conversation by requesting a communication session with the Server: This process takes three steps

- Step 1**:** The client establishes a connection with a server. It sends a segment with SYN and informs the server about the client should start communication, and with what should be its sequence number.
- Step 2**:** Now **s**erver responds to the client request with SYN-ACK signal set. ACK helps you to signify the response of segment that is received and SYN signifies what sequence number it should able to start with the segments.
- Step 3**:** In this final step, the client acknowledges the response of the Server and stable connection is created between them.

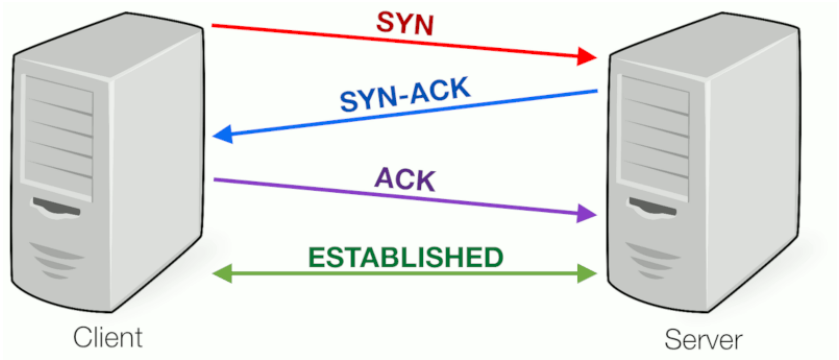


Fig 1.2-Three-way Handshake Protocol diagram

1.3 SOCKET

Sockets are the endpoints of a bidirectional communication channel.They are the most popular form of IPC especially for cross communication. Sockets may communicate within a process, between processes on the same machine,or different machines. Each socket is bound to a specific port number and each processes has their own IP. . A socket created by the server will link itself to a unique port and then it waits for other sockets created by clients to join via the port while the server will wait and listen. After the clients have joined the server, the communication between them is done via a FIFO queue, i.e., the message that is first sent out will be the first to be received by other clients.

Working of Sockets

We have inbuilt modules in python for sockets. We use those sockets by importing them. Then we create a server socket and we bind an IP address and port number to the server socket. We make the server to listen for the requests sent by the clients. Same as server for client we have to create a client socket. Then we connect the client socket to an IP address and the same port number which is used in the server. As soon as the client sends the request to the server, server accepts the client request and stores the client's IP address and port number.

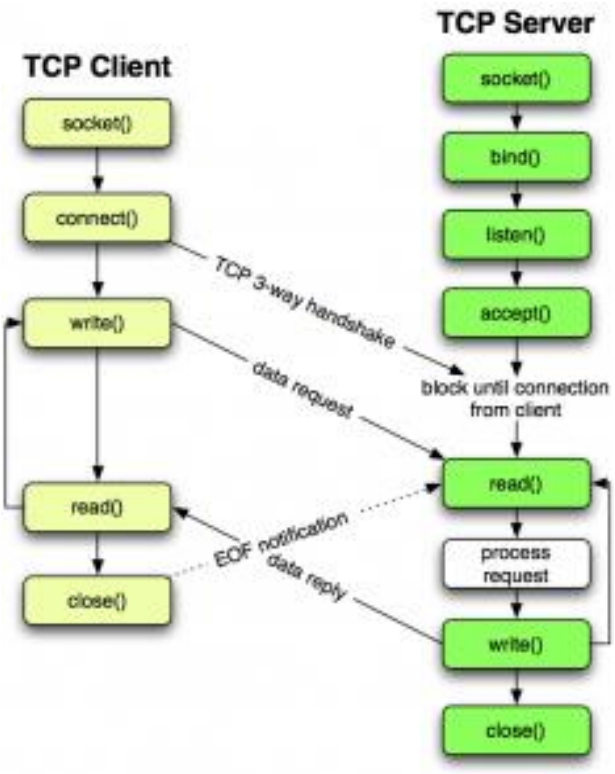


Fig 1.3- Flow Chart representing client server model

- Socket() - Endpoint for communication
- Bind() - Assigns a unique IP address and port number to a socket instance
- Listen() – It indicates readiness to accept client connect request
- Connect() - Sending a connection request
- Accept() - Accepting the request
- Send()-It is used to transmit a message form specified socket to its peer. This can be used only if the two sockets are connected.
- Recv() - It is used to read data from the incoming transmission through the socket.
- Close() - Hang up the connection

These are the methods used in connecting the client and server sockets. Socket() method creates a socket object. We use that socket object and bind() method to hold an IP address and a port number. Send() method sends the data in bytes format by using an encoding technique. Recv() method accepts the no. of bytes as an argument (gets the bytes data) and we decode the data into a string value using a similar decoding technique used in encoding Close() method closes the socket and ends the connection.

We also use some of the inbuilt modules such as pickle struct for efficient use of socket module

Pickle: Used for serializing and de-serializing a python object structure. The process of converting any kind of python objects into byte streams is called pickling.

dump() : The process of converting any kind of python objects into byte streams is called pickling and is done through dump().

load() :is used for reading the file for the unpickling process and it operates on the string

Struct: Used to convert native Python data types into string of bytes

pack() : It is the function that converts a given list of values into their corresponding string representation. It requires the user to specify the format and order of the values that need to be converted.

unpack():It is the function that converts the strings of binary representations to their original form according to the specified format. The return type is always a tuple.

1.4 VIDEO TRANMISSION AND PROTOCTING USING SOCKET PROGRAMMING

In this project we create multiple client sockets and transmitting their videos to a server in Python. The client utilizes OpenCv to access the video frames either from the live webcam or through the MP4 video. The server side code runs multi-threading to display and proctor video frame of each connected client. We mainly use two face detection algorithm for proctoring depending upon the situation.

1.Using Mediapipe -We use to detect face.

2.Using Haar-cascade -We use to detect face and eye.

The proctored video will be stored in the desktop. If face is not aligned or visible properly server will send message as warning to the client to align the face properly.

1.4.1 OpenCv

It is a library of Python bindings that is generally used to solve problems related to computer vision.It is a cross-platform library available in a wide variety of programming languages. Opencv can be used to process mages and videos to identify objects ,faces. We use vector space and perform mathematical operations on these features to identify image pattern and its various features.

Fig 1.4-OpenCV Logo

1.4.2 MediaPipe

MediaPipe is an open-source, cross-platform Machine Learning framework used for building complex and multimodal applied machine learning pipelines. It can be used to make cutting-edge Machine Learning Models like face detection, multi-hand tracking, object detection, and tracking, and many more. MediaPipe basically acts as a mediator for handling the implementation of models for systems running on any platform which helps the developer focus more on experimenting with models, than on the system.

MediaPipe Face Detection: MediaPipe Face Detection is an ultrafast face detection solution that comes with 6 landmarks and multi-face support. The detector's super-realtime performance enables it to be applied to any live viewfinder experience that requires an accurate facial region of interest as an input for other task-specific models, such as 3D facial keypoint, geometry estimation, facial features or expression classification, and face region segmentation.

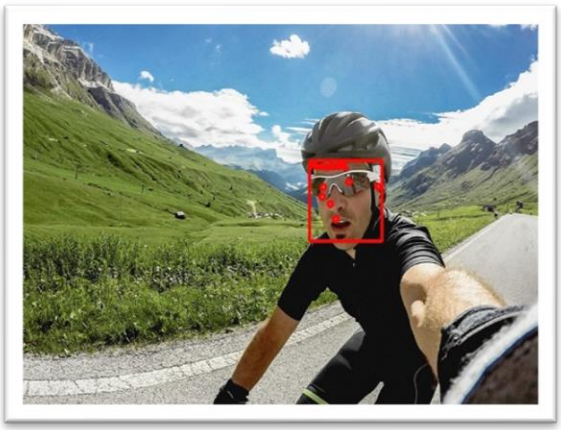


Fig 1.5-Illustration of Face detection using MediaPipe

1.4.3 Haar-cascade Detection

Basic Concept of HAAR Cascade Algorithm

The HAAR cascade is a machine learning approach where a cascade function is trained from a lot of positive and negative images. Positive images are those images that consist of faces, and negative images are without faces. In face detection, image features are treated as numerical information extracted from the pictures that can distinguish one image from another.

We apply every feature of the algorithm on all the training images. Every image is given equal weight at the starting. It founds the best threshold which will categorize the faces to positive and negative. There may be errors and misclassifications. We select the features with a minimum error rate, which means these are the features that best classifies the face and non-face images.All possible sizes and locations of each kernel are used to calculate the plenty of features.

.The model created from this training is available at the OpenCV GitHub repository.The repository has the models stored in XML files, and can be read with the OpenCV methods. These include models for face detection, eye detection, upper body and lower body detection, license plate detection etc..In this Project we will use face and eye detection.

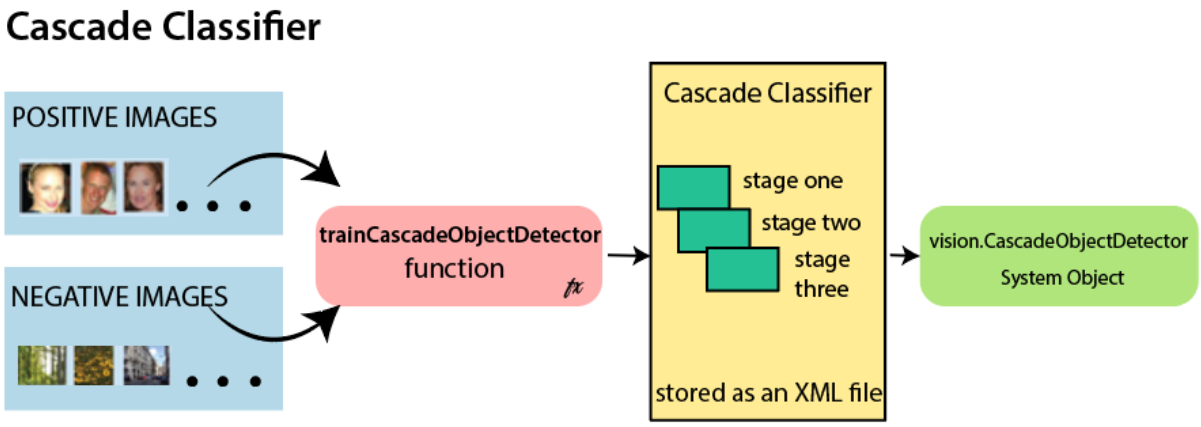


Fig 1.6-Concept of Cascade Classifier

HAAR-Cascade Detection in OpenCV

OpenCV provides the trainer as well as the detector. We can train the classifier for any object like cars, planes, and buildings by using the OpenCV. There are two primary states of the cascade image classifier first one is training and the other is detection.

OpenCV provides two applications to train cascade classifier **o**pencv_haartraining and opencv_traincascade. These two applications store the classifier in the different file format.

For training, we need a set of samples. There are two types of samples:

- Negative sample**:** It is related to non-object images.
- Positive samples**:** It is a related image with detect objects.

A set of negative samples must be prepared manually, whereas the collection of positive samples are created using the opencv create samples utility.

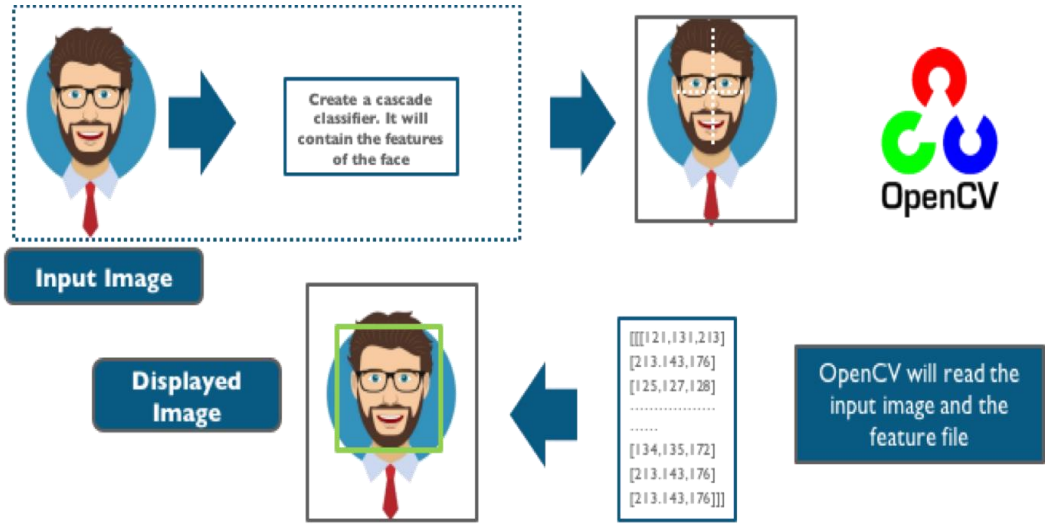


Fig 1.7- Working Model of Haar-Cascade Detection

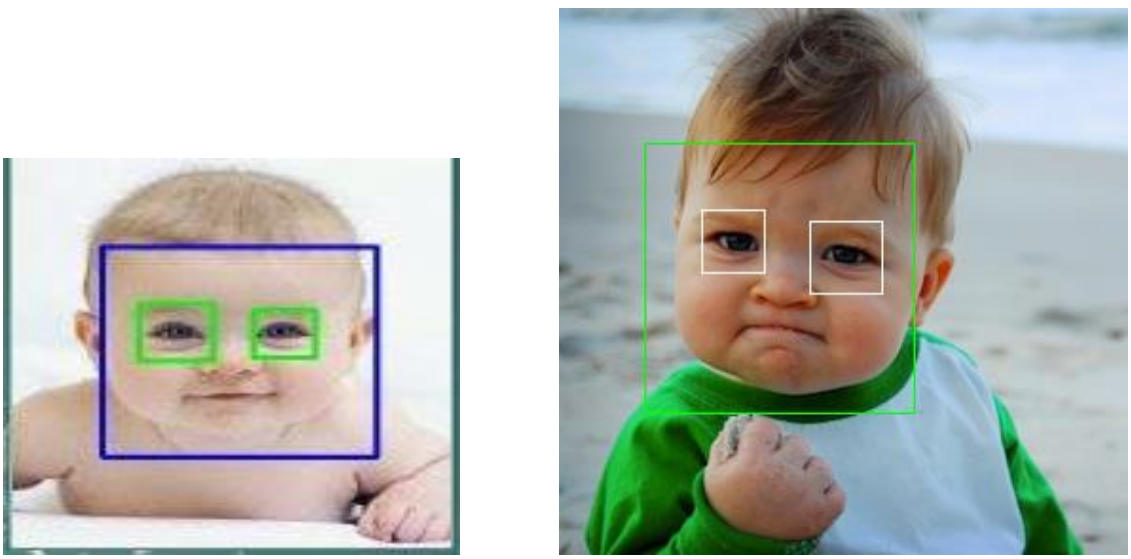


Fig 1.8- Illustration of Face and Eye detection using Haar-Cascade Detection

1.5 AUDIO TRANSMISSION USING SOCKET PROGRAMMING

PyAudio

We have an audio file (.wav), and we want to send it to the client so that the client can listen the stream as a playback in real-time. For this purpose we require PyAudio and socket programming. PyAudio is a module in python. PyAudio provides Python bindings for PortAudio, the cross-platform audio I/O library. With PyAudio, you can easily use Python to play and record audio on a variety of platforms.

The audio data normally consists of 2 channels, which means the data array will be of shape (CHUNK,2), where CHUNK is the number of data samples representing the digital sound. We commonly put CHUNK as 1024.

1.5.1 pyaudio.PyAudio

Pyaudio is a class and stored in variable(instance).It is used to start or terminate PortAudio. Open or Close streams and also used to access audio devices

1.5.2 Opening an audio stream

```
P=pyaudio.PyAudio
Stream=p.open()
```

open() has more arguments but here we are going to use six arguments

Arguments	**Description**
rate	sampling frequency (usually 44.1/48 kHz)
format	bit depth (usually 16/24 bits)
channels	number of audio channels
input	to use the stream as input
output	to use the stream as output
frames_per_buffer	length of the audio buffer

Table 1.1- Arguments of Open in PyAudio Stream and its description

It is more convinent to use .WAV file in PyAudio. So we import a module called WAVE.

1.5.3 wave

The wave module in Python's standard library is an easy interface to the audio WAV format. The functions in this module can write audio data in raw format to a file like object and read the attributes of a WAV file.

wave.open() :This function opens a file to read/write audio data. The function needs two parameters - first the file name and second the mode. The mode can be 'wb' for writing audio data or 'rb' for reading.In this project we are going to read audio data.

We use some of the Wave read methods in our project :

Arguments	**Description**
getnchannels()	Returns number of audio channels (1 for mono, 2 for stereo).
getsampwidth()	Returns sample width in bytes.
getframerate()	Returns sampling frequency.
readframes(n)	Reads and returns at most n frames of audio, as a bytes object.

Table 1.2- Arguments of Wave read methods and its description

1.6 MERITS AND DEMERITS:

Merits

The project can be used to proctor the client.so if they are in same LAN the server can proctor the client in online mode itself.

It uses the concept of multithreading so more clients can be proctored and in case of audio each clients can transfer audio to server.

It uses three-way handshake so its more reliable and there is no loss of datapackets while transmitting audio and video and assurance of the transmission.

The Video Proctoring is a two way communication so the server can interact with client, when face is not visible it send the immediate message to align the face properly.

The Video Proctored is stored in desktop as mp4 file so server can use the video for future purpose.

The Audio is transferred without any noise and video is transmited without lacking in frame

This Project is built in easy and effective way so’its more cheaper and every common users can use it.

Demerits:

If one client sends the audio or video the other clients should also do the same which becomes the main drawback in this project.

Clients in WAN cannot be joined so it is resisted only to LAN.

Video transmission is two way communication but Audio isn't and audio file is only WAV extension file, so other files need to be converted to Wave file.

Only recored audio is transmitted so live audio transmission becomes a great drawback which if done can be great addition to online Proctoring.

SOURCE CODE

Server Code

```
import socket, cv2, pickle, struct

import imutils

import threading

import cv2

import mediapipe as mp

import wave, pyaudio

import time

import pyshine as ps


path=r"C:/Users/ELCOT/Desktop/opencv/haarcascade_eye.xml"

eye_classifier = cv2.CascadeClassifier(path)


path2=r"C:/Users/ELCOT/Desktop/opencv/haarcascade_frontalface_default.xml"

face_classifier = cv2.CascadeClassifier(path2)


#For storing the video

fourcc=cv2.VideoWriter_fourcc(*'XVID')

out=cv2.VideoWriter('detection2.avi',fourcc,20.0,(640,480))

#face detector using haar-casade
```

```
def face_detector(img, size=0.5):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray, 1.3, 5)
    if faces is ():
        return 0,img

    for (x, y, w, h) in faces:
        x = x - 50
        w = w + 50
        y = y - 50
        h = h + 50
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 248), 2)
        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]
        eyes = eye_classifier.detectMultiScale(roi_gray)

        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 39, 255), 2)
    return 1,img
#face detection using media pipe
def face_detection(img):
    mpFaceDetection = mp.solutions.face_detection
    mpDraw = mp.solutions.drawing_utils

    faceDetection = mpFaceDetection.FaceDetection(0.75)
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = faceDetection.process(imgRGB)

    if results.detections:
```

```
        for id, detection in enumerate(results.detections):
            bboxc = detection.location_data.relative_bounding_box
            ih, iw, ic = img.shape
            bbox = int(bboxc.xmin * iw), int(bboxc.ymin * ih), int(bboxc.width * iw),
int(bboxc.height * ih)
            cv2.rectangle(img, bbox, (255, 0, 255), 2)
            cv2.putText(img, f'{int(detection.score[0] * 100)}%', (bbox[0], bbox[1] - 20),
                    cv2.FONT_HERSHEY_PLAIN, 3, (0, 255, 0), 2)
        return 1,img
    else:
        return 0,img
#for audio receiving pyaudio stream is used
p = pyaudio.PyAudio()
CHUNK = 1024
stream = p.open(format=p.get_format_from_width(2),
                channels=2,
                rate=44100,
                output=True,
                frames_per_buffer=CHUNK)
#For connection establishment
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_name = socket.gethostname()
host_ip = socket.gethostbyname(host_name)
print('HOST IP:', host_ip)
port = 9996
socket_address = (host_ip, port)
server_socket.bind(socket_address)
server_socket.listen()
print("Listening at", socket_address)
```

```
pop="

j=int(input("ENTER 1: FOR AUDIO  2: FOR VIDEO  :  "))
if j==1:
    pop='A'
elif j==2:
    pop='V'
else:
    print("ENTERED INCORRECT NUMBER")

def show_client(addr, server_soc):
    try:
        print('CLIENT {} CONNECTED!'.format(addr))
        if server_soc:  # if a client socket exists
            data = b""
            palo_size = struct.calcsize("Q")
            while True:
                while len(data) < palo_size:
                    packet = server_soc.recv(50*1024)  # 4K
                    if not packet: break
                    data += packet
                packed_msg_size = data[:palo_size]
                data = data[palo_size:]
                msg_size = struct.unpack("Q", packed_msg_size)[0]
                while len(data) < msg_size:
                    data += server_soc.recv(4 * 1024)
                frame_data = data[:msg_size]
                data = data[msg_size:]
```

```
            frame = pickle.loads(frame_data)

            if pop=='V':            #For video receiving and proctoring purpose
                text = f"CLIENT: {addr}"
                l,frame=face_detection(frame)
                cv2.imshow(f"FROM {addr}",frame)
                # send message to client if video is not visible properly
                if l==0:
                    server_soc.send(b"0")
                elif l==1:
                     server_soc.send(b"1")
                out.write(frame)
                key = cv2.waitKey(1) & 0xFF
                if key == ord('q'):
                    break

            if pop=='A':            #For audio receiving and playing purpose
                stream.write(frame)
                g = cv2.imread(r'C:\Users\ELCOT\Desktop\samples\AFI.png')
                g = imutils.resize(g, width=380)
                cv2.imshow("FROM IMAGE", g)
                key = cv2.waitKey(1)
                if key == ord("q"):
                    break


        server_soc.close()

except Exception as e:
```

```
        print(f"CLIENT {addr} DISCONNECTED")
        pass
while True:
        server_soc, addr = server_socket.accept()
        #uses the threading concept
        thread = threading.Thread(target=show_client, args=(addr, server_soc))
        thread.start()
        print("TOTAL CLIENTS ", threading.activeCount() - 1)
stream.close()
```

Client Code

```
import socket, cv2, pickle, struct
import imutils  # pip install imutils
import pyaudio
import wave

#To capture the videos
camera = False
if camera == True:
    vid = cv2.VideoCapture(0)
else:
    vid = cv2.VideoCapture(r"C:\Users\ELCOT\Desktop\samples\2.mp4")



#For socket connection
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '192.168.56.1'

port = 9996
```

```
client_socket.connect((host_ip, port))
print('server listening at', (host_ip, port))
#opening wave file in read mode
wf = wave.open(r'C:\Users\ELCOT\Desktop\2.wav', 'rb')
p = pyaudio.PyAudio()  # Pyaudio is a class and stored in variable(instance)
CHUNK = 1024
stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                channels=wf.getnchannels(),
                rate=wf.getframerate(),
                input=True,
                frames_per_buffer=CHUNK)


j = int(input("ENTER 1: FOR AUDIO  2: FOR VIDEO  :  "))
if client_socket:
    while True:
        if j == 2:
            #For sending video to server
            img, frame = vid.read()                    #reads the video
            picA = pickle.dumps(frame)                 #convert into byte stream
            message = struct.pack("Q", len(picA)) + picA  #makes as string of bytes to send
            client_socket.sendall(message)             #sends the message
            msg2 = client_socket.recv(2048).decode()      #recieves the message in case video is
not visible or alligned
                                                #properly
            if msg2=='0':
                #In case  the video is not aliigned or visible it shows in the client scene
                cv2.putText(frame, f'{"FACE NOT ALLIGNED PROPERLY"}', (80, 40),
cv2.FONT_HERSHEY_PLAIN, 2, (245, 55, 98), 3)
            cv2.imshow(f"TO: {host_ip}", frame)          #to show the video in client screen
            key = cv2.waitKey(1) & 0xFF
```

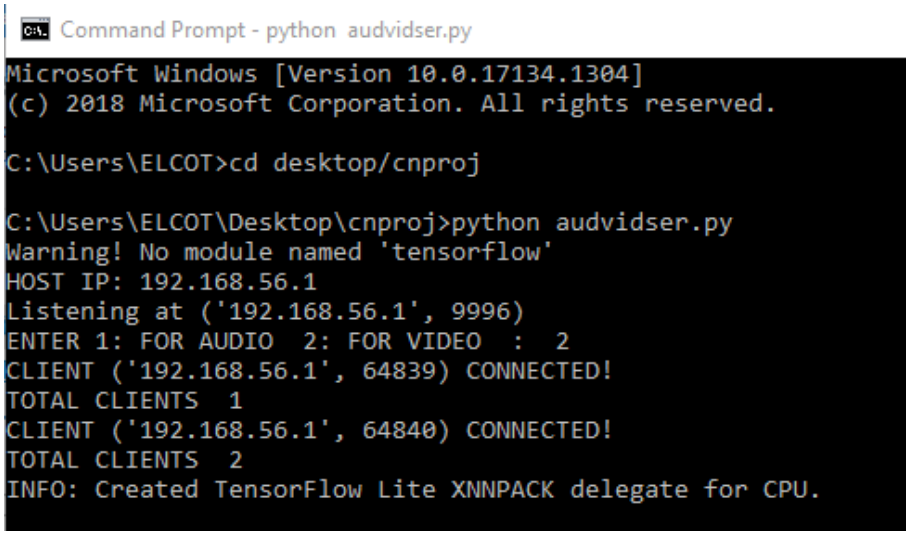
```
    if key == ord("q"):                    #quits if q is pressed
        client_socket.close()


elif j == 1:
    # For sending video to server
    data = None
    try:
        data = wf.readframes(CHUNK)
        a = pickle.dumps(data)
        message = struct.pack("Q", len(a)) + a
        client_socket.sendall(message)
        s = len(data)
        #shows image when audio is transmitting
        g = cv2.imread(r'C:\Users\ELCOT\Desktop\samples\AFI.png')
        g = imutils.resize(g, width=380)
        cv2.imshow(" TO IMAGE", g)
        key = cv2.waitKey(1)
        if key == ord("q"):
            break
    except:
        client_socket.close()
else:
    print("ENTERED A INCORRECT NUMBER")
    break
```

CHAPTER 3

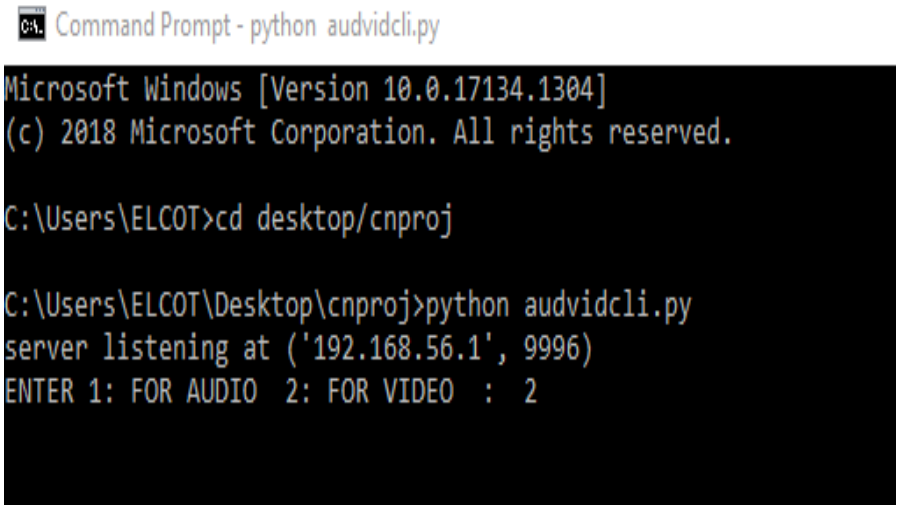
SNAPSHOTS

Server Output for Video Transmitting

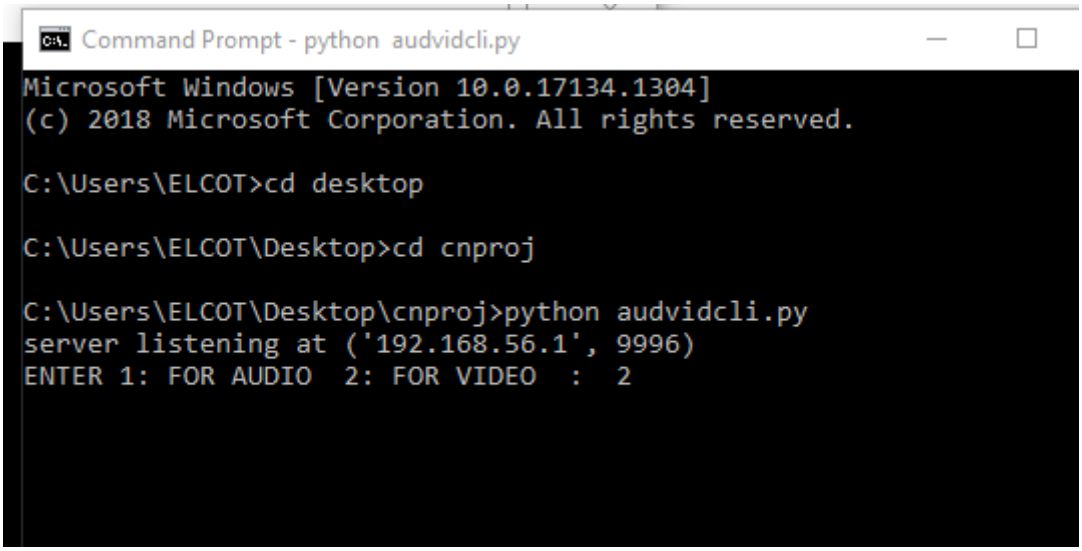


Client Output for Video Transmitting

Client 1

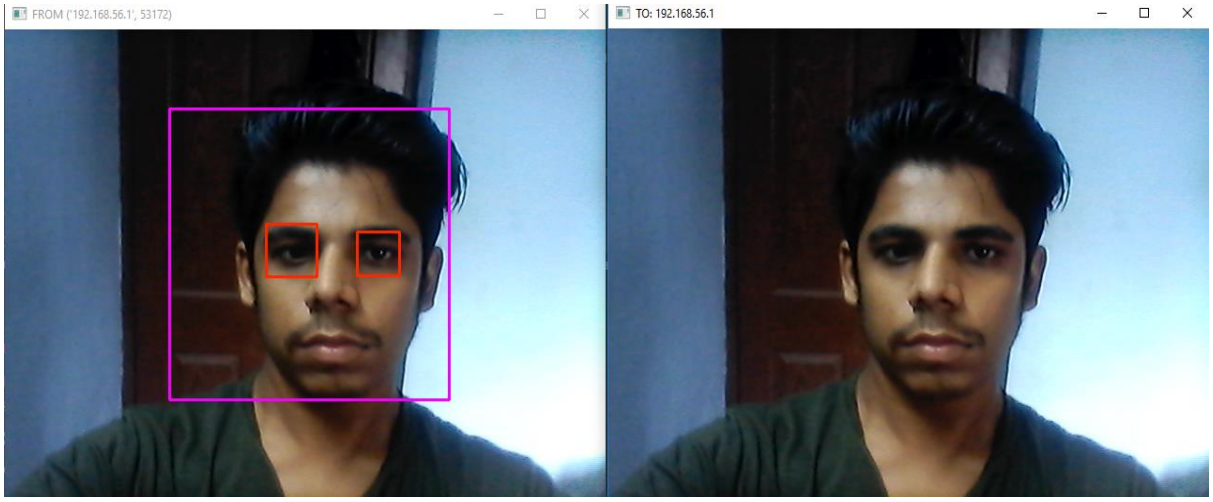


Client 2

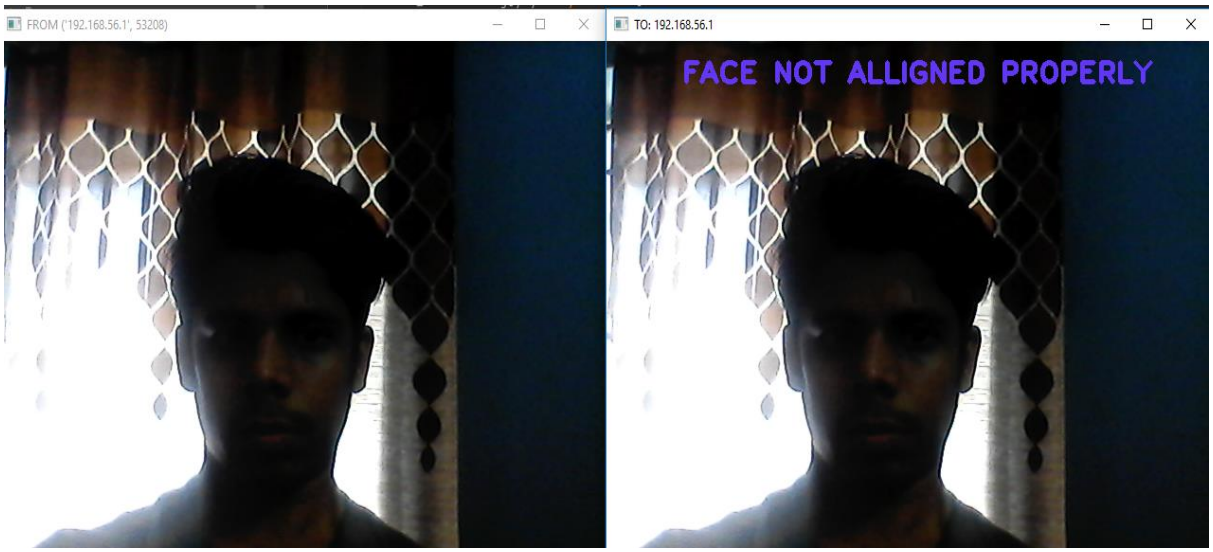


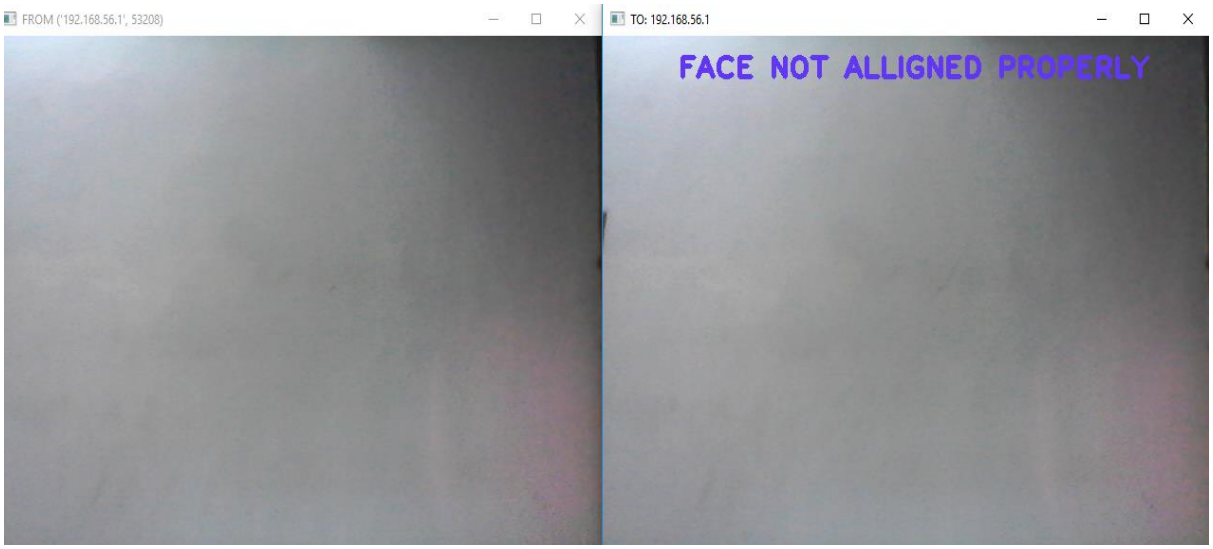
Video transmitting and proctoring [Haar-cascade]

Face And Eye detection



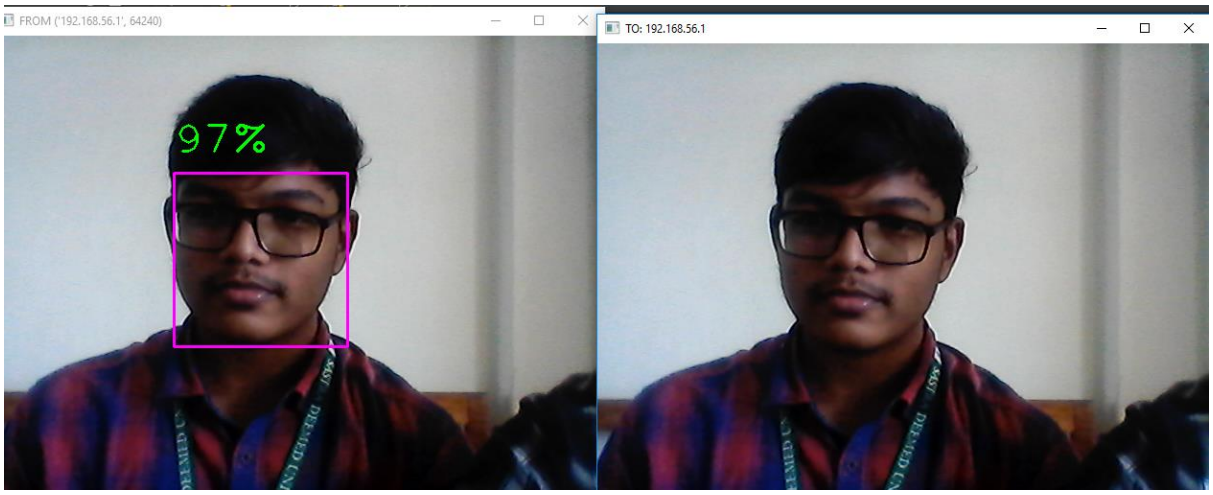
When the face is not visible properly



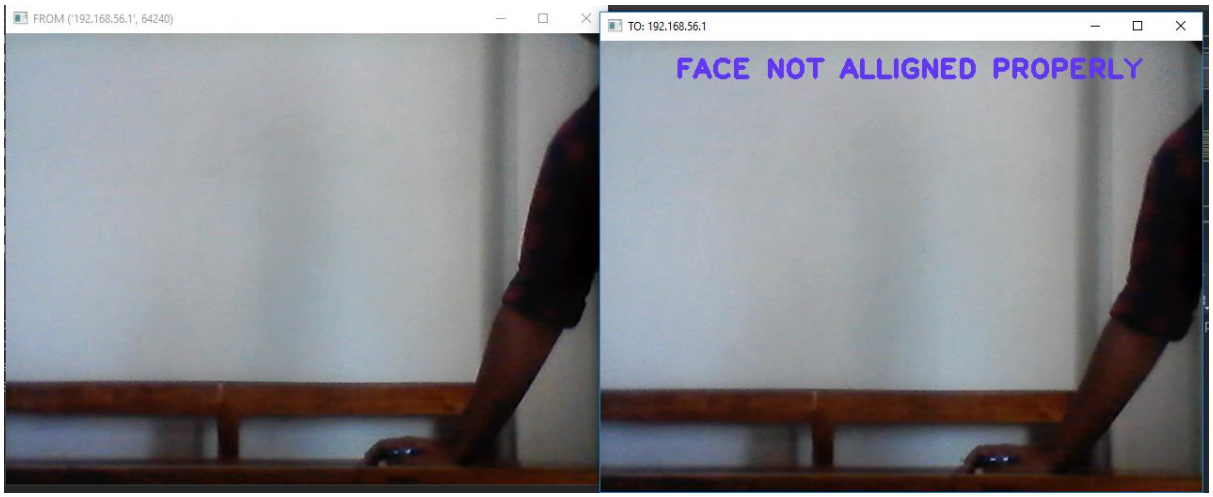


Video transmitting and proctoring [Mediapipe]:

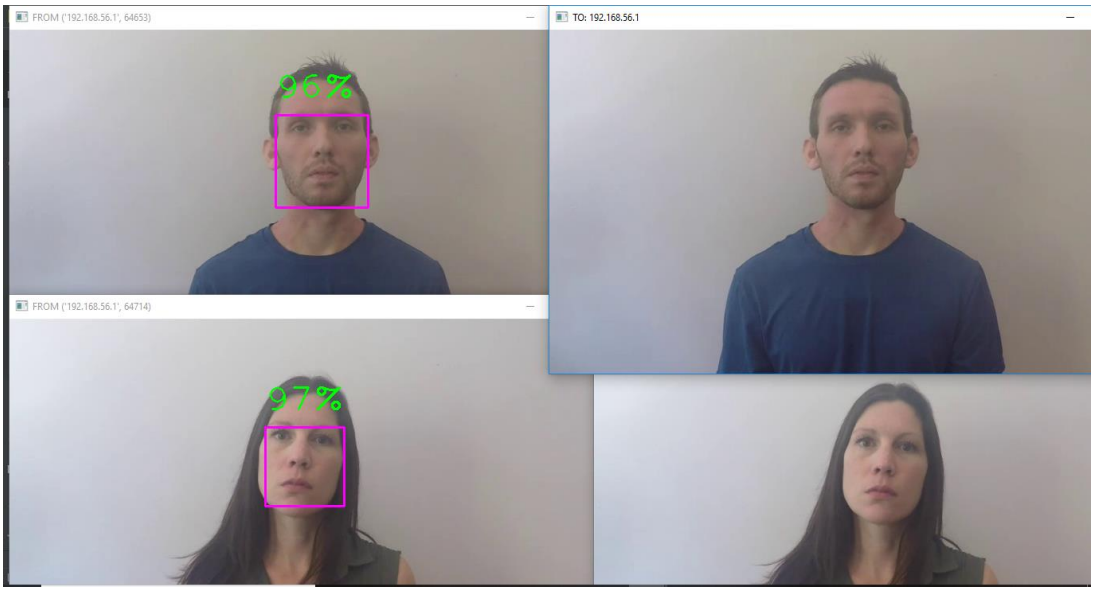
Face Detection Using Mediapipe



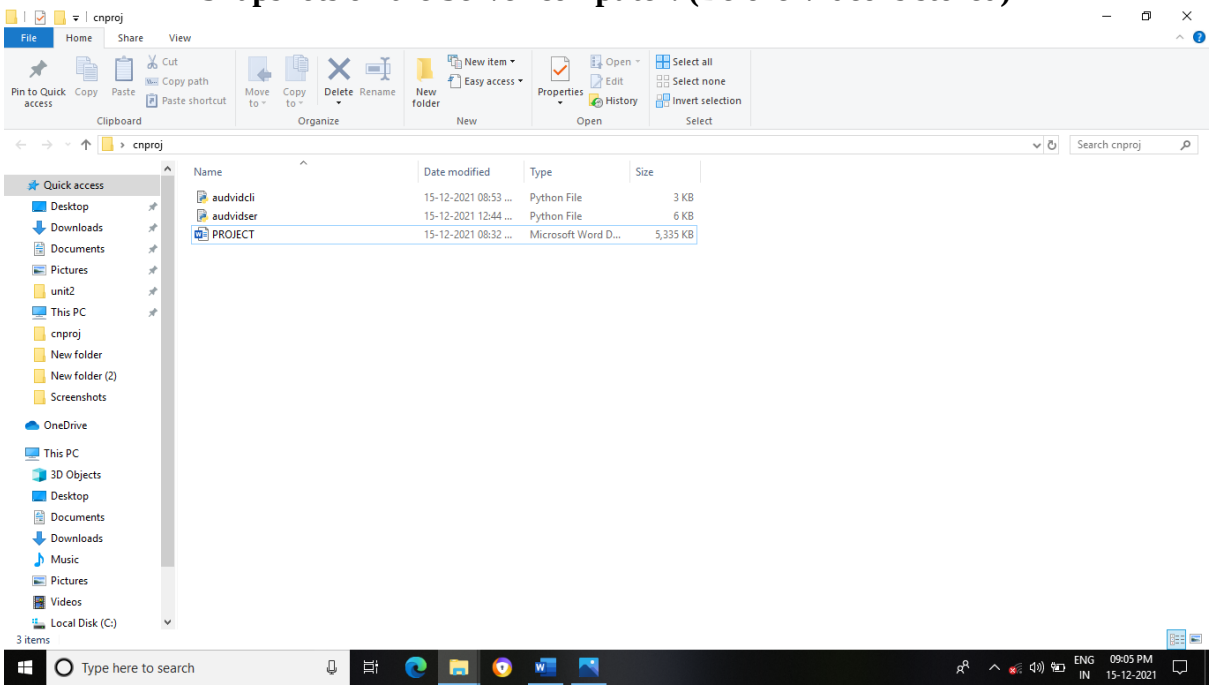
When Face is not visible properly



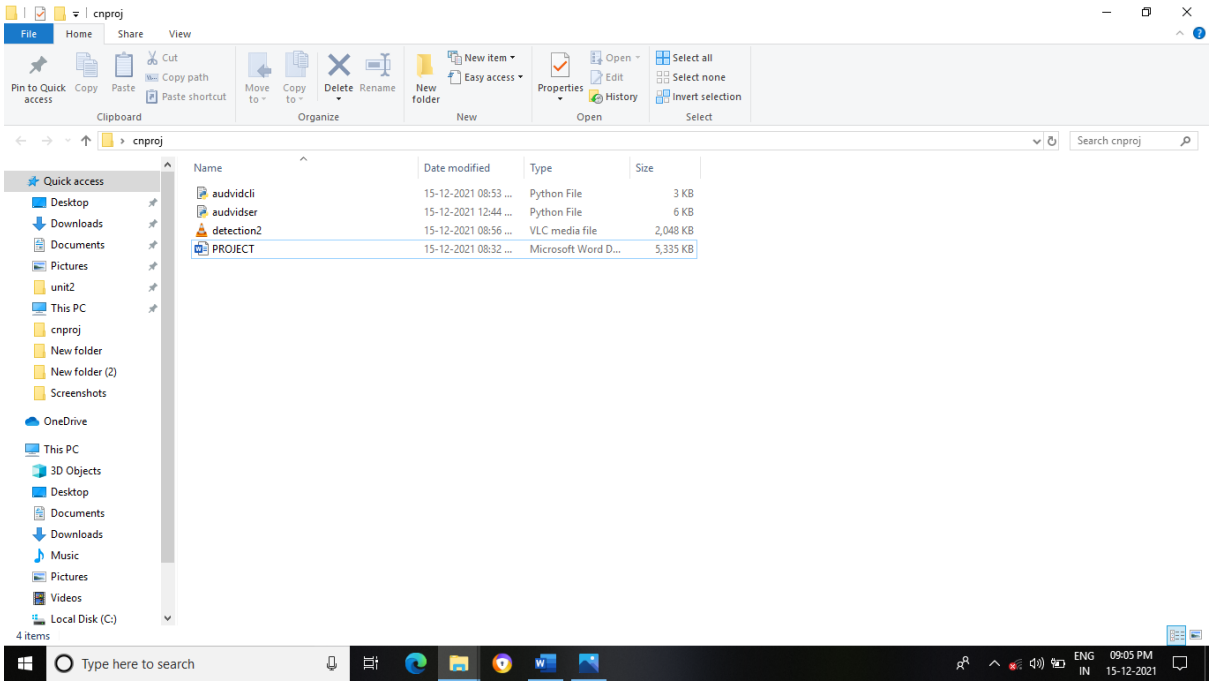
MediaPipe Face Detection using more than one client



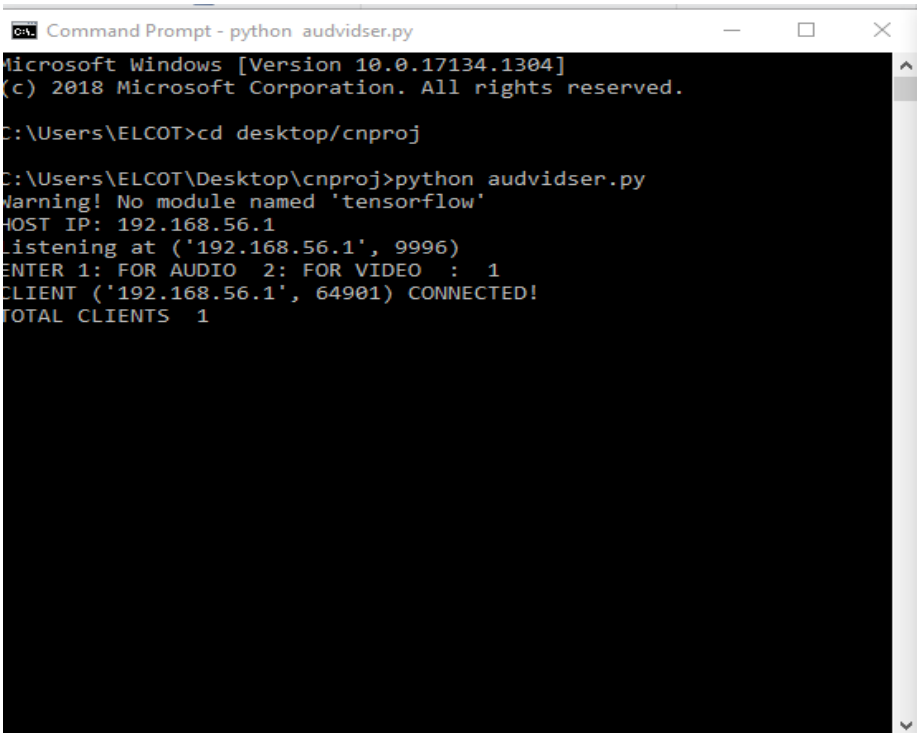
Snapshots on the Server computer: (Before Video is stored)



Snapshots on the Server computer: (After Video is stored)



Server output for Audio transmitting



Client output for Audio transmitting

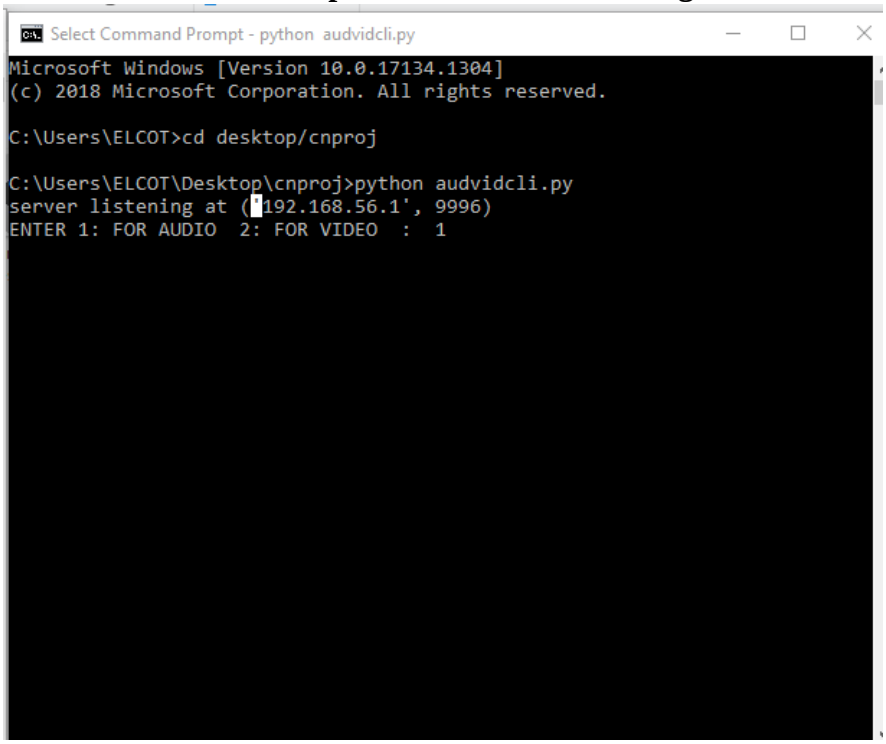
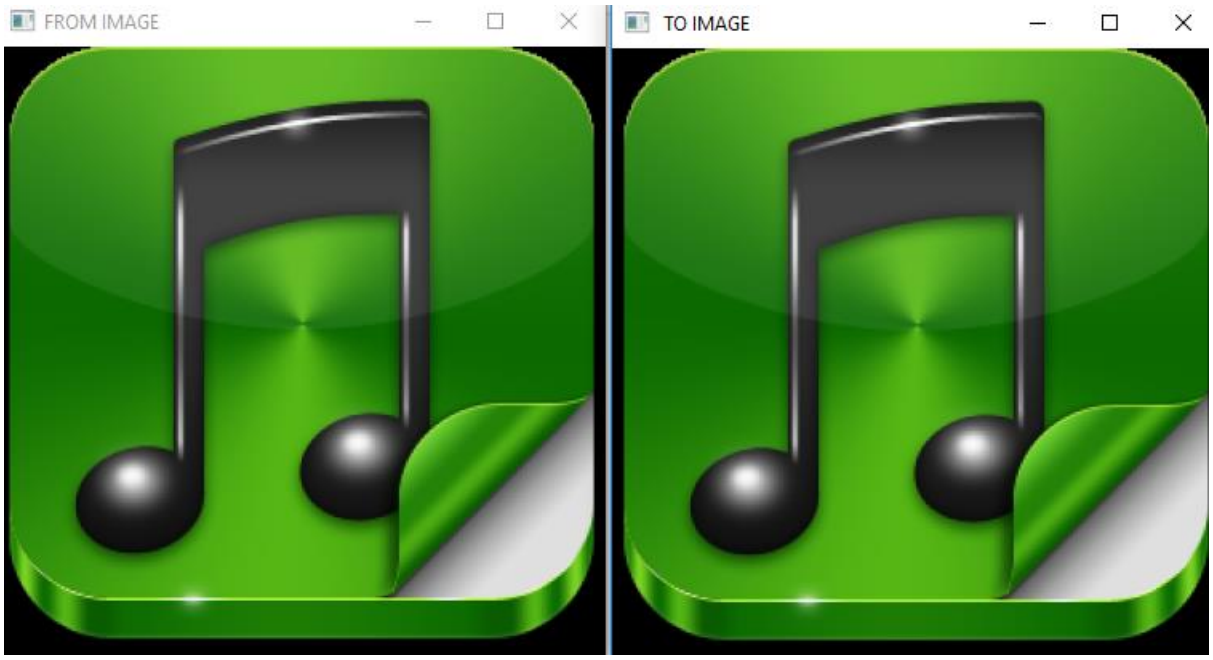


Image Shown While audio is being Transmitted



CONCLUSIONS AND FUTURE SCOPE

Based on the results obtained ,We can conclude Video(with proctoring) and audio transmiting was successful. We used TCP implementation hence the connection was more reliable,the multithreading concept we used was successful. Accuracy of the face detection was better and there was no lacking in transmitting the frames.so it becomes more reliable.Audio transmitted was not containing any noise. Eventhough the project is efficient always there will be room for improvement. Given are some of the improvements that can be made in near future.

The whole project is done with respect to LAN, but we can take this to WAN by uploading server file into an online cloud server and give that IP to client and we can access the online server through our computer. By this way we can connect remotely to any desktop from anywhere around the world.Since this project consists of video proctoring we need to ensure the client don’t switch tabs in computer. So addition of screen sharing option could be better choice. We used haarcascade algorithm for face and eye detection but its accuracy is low compared to mediapipe face detection algorithm. So we could improve this algorithm or we can add eye detection in mediapipe’s face detection for more accuracy in eye detection and face detection.
The audio we used here are the recorded audio so we could improve this by adding live stream audio too. The audio used here are only Wave extensions we could try to include all the audio extensions.We could include two way communication only audio too.The video and audio can’t be shared simultaneously at same time so we could improve in this area for more efficiency.

REFERENCES:

https://google.github.io/mediapipe/solutions/face_detection

https://people.csail.mit.edu/hubert/pyaudio/docs/

https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

https://docs.python.org/3/library/socket.html

https://medium.com/nerd-for-tech/developing-a-live-video-streaming-application-using-socket-programming-with-python-6bc24e522f19

https://pyshine.com/