Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

CSE4001 PDC

FACE RECOGNITION USING PARALLEL COMPUTING

-SANDHYA S 19BCE1614

Abstract

As the famous proverb says, “Face is the index of the mind”. A face to face interaction between human beings is considered the most important and natural way to communicate. Recognition of faces and processing the data is a challenging task with very large databases using low-cost desktop embedded computing.

The face recognizing systems available so far maintain a database that has each pre-processed human face and their corresponding unique features as determined from each face. These features are stored with the respective individual’s face. Therefore, when a query is raised, the unique features are extracted from the face and then compared with the features in the database and results in a perfect match .There are many areas of applications that use face recognition technology ranging from security applications for recognizing criminals in public spaces such as airports, and shopping centers, verifying access to private property, and casting votes, to intelligent vision-based human computer interaction

such as ATM, cell phones and intelligent buildings. Identifying the images and processing the data is a challenging task because of the various factors involved in this sophisticated process such as illumination, angle of pose, accessories, facial expression, and aging effects. There are two sets of data involved in the face recognition system. The first is the training set of data that is used in the learning stage. The second is a testing set which is used during recognition. Often real time response is understood to be in order of milliseconds and sometimes microseconds, which is the most crucial criteria in the system design.

All in all, it can be said that this project is a good example of a parallel task with omp. Dealing with multiple pieces of data to read, calculate, and compare results takes a long time when tasks have to be performed sequentially.

Motivation

So basically our motive behind this project is to aim and achieve face recognition through parallel computing. This reduces the time taken for the process to complete and also increases the performance and hence the efficiency.

Implementation Details

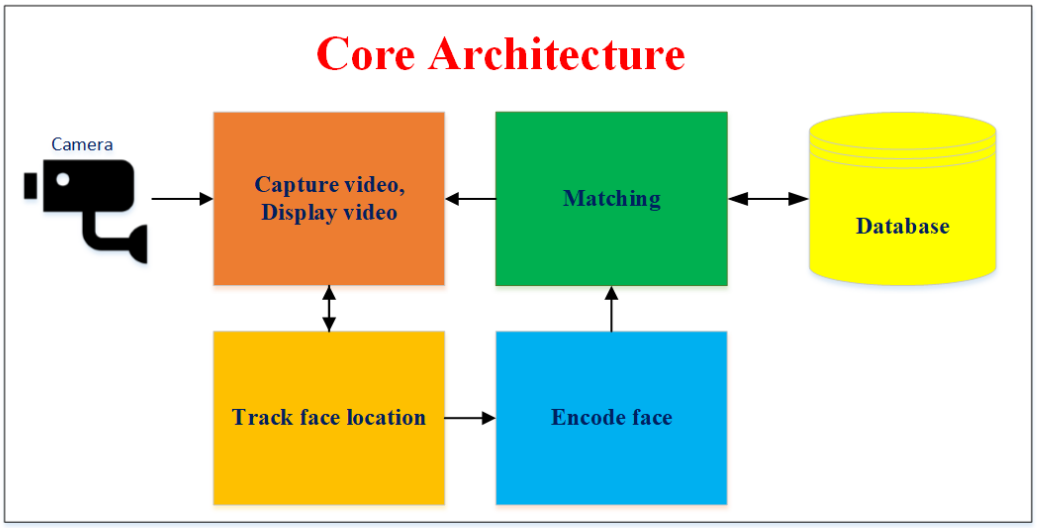
Design

The system is designed using omp programming in python language. The system has used threads concept to implement parallel algorithms in face recognition. We also use dlib, cmake, face recognition libraries for the implementation of the project. We have installed a new camera software called gucview in ubuntu virtual machine inorder to proceed with face recognition.

There are two folders, e.g., "1.original-system", "2.parallelized-system". In which, they contain original-system code, parallelized-system code respectively. To parallelize the original system in Python 3, I use Process-based “threading” interface.

System Architecture

When a frame of video comes, thread 1 receives it, displays it into the screen, and sends it to thread 2. Thread 2 is to detect faces by pointing out coordinates of vertices of the bounding boxes of faces in the image. Then, thread 3 uses the output of thread 2 to crop an image to bounding boxes of faces and encode them as face embeddings. Finally, the last thread takes extracted face embeddings so as to perform a matching with templates saved in the database.



Modules

Serial part of Face Recognition System

Parallel part of Face Recognition System
Histograms
Camera Module

Dependencies Used
Ubuntu 16.04

Python 3.5

OpenCV 3.3.1

face_recognition

Experimental results & Discussion

On comparing the execution time for the original system and parallel system we conclude that the parallel system is more effective compared to the original system as it's execution time is comparatively lesser. Lesser execution time implies better performance. And better performance implies better CPU utilization. Thus the parallel face recognition algorithm proves to be better than the original face recognition algorithm.

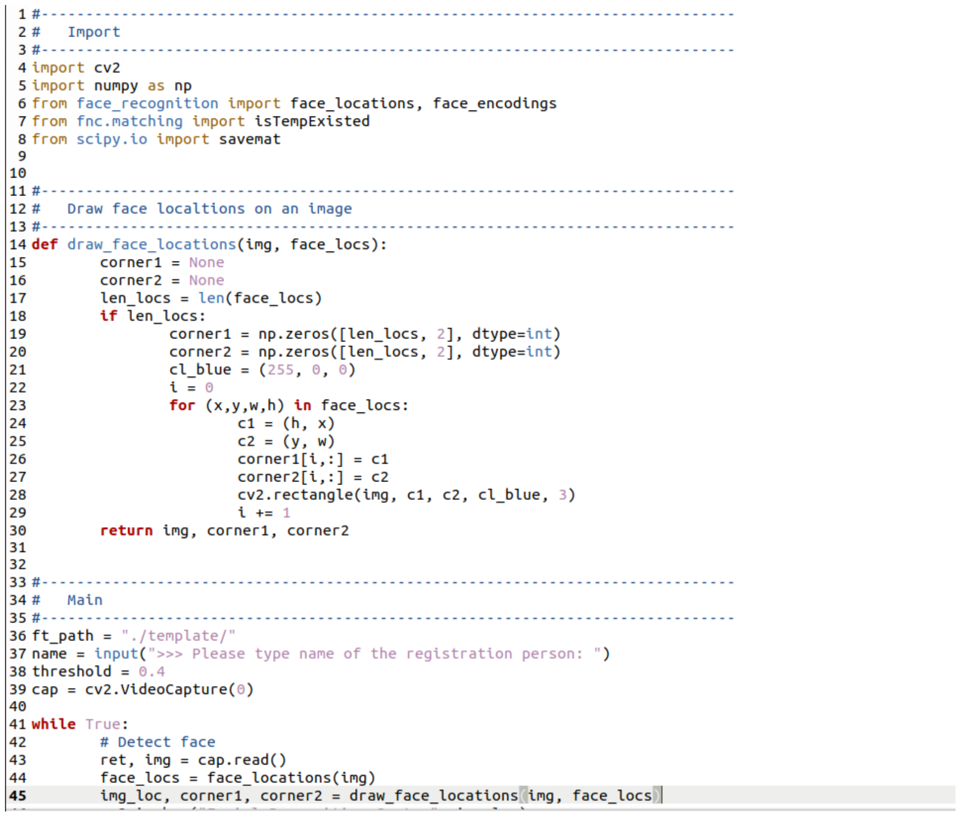
Conclusions & Future work

There are various pros and cons in this system. This parallel face recognition system utilizes the hardware and softwares resources effectively. The multiple threads we have created can run simultaneously making the process much simpler. A thread can process subsequent frames and do not need to wait for the previous thread to complete its task. The drawbacks of this system include the time calculation. The time calculation includes the time taken by the camera module to start. Since we are using a very basic camera module called gucview as the built in camera module of the ubuntu virtual machine was malfunctioning. This camera module is comparatively slower and hence takes longer time to open and capture the face. This causes a longer time duration for the face recognition process to complete. The issue is with the camera software that I have used and not the program. Hence as a future work if we could try and implement a better and an efficient camera module the system will function perfectly.

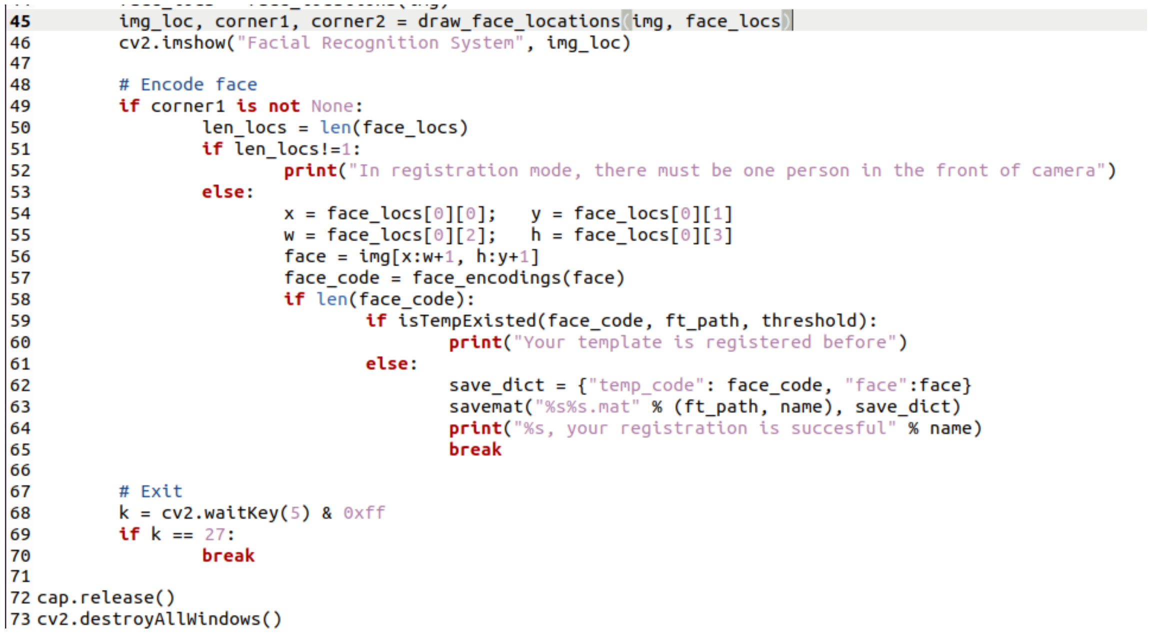
Annexure- I

a. Source code

ORIGINAL SYSTEM - registration.py

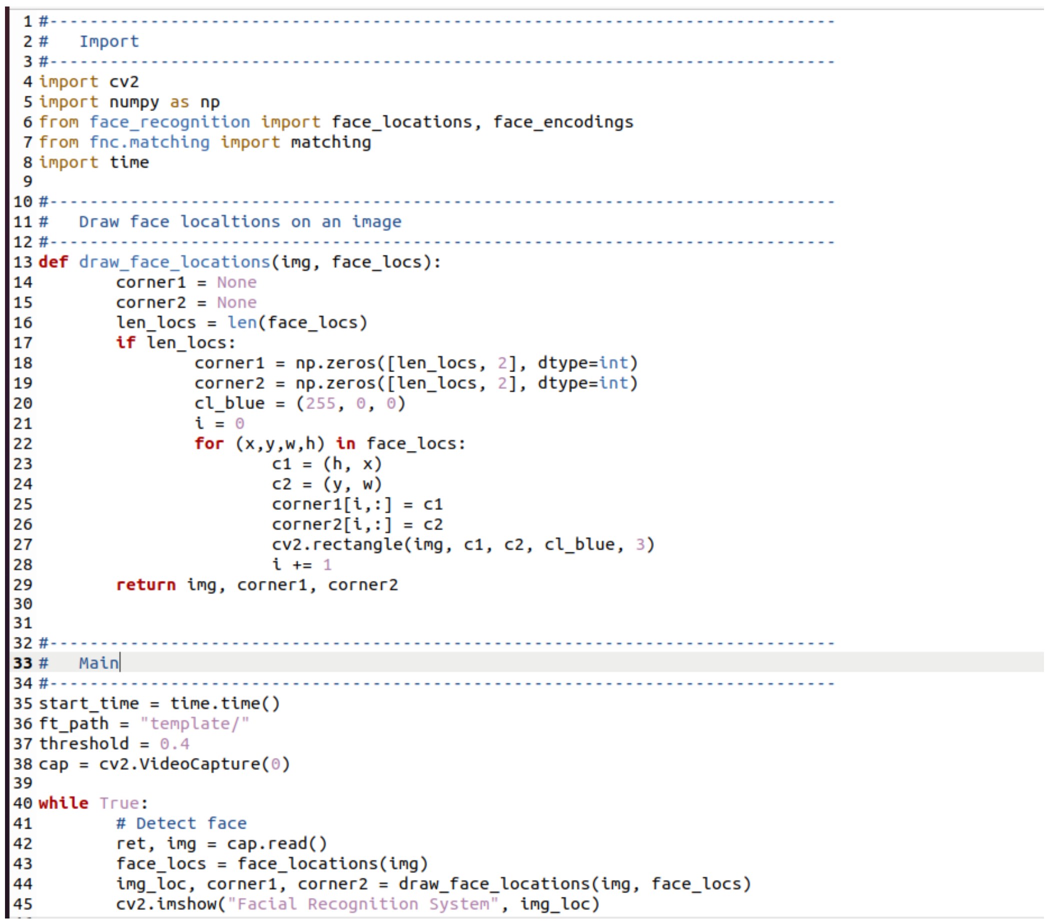


```python
#---------------------------------------------------------------------------------
#   Import
#---------------------------------------------------------------------------------
import cv2
import numpy as np
from face_recognition import face_locations, face_encodings
from fnc.matching import isTempExisted
from scipy.io import savemat


#---------------------------------------------------------------------------------
#   Draw face localtions on an image
#---------------------------------------------------------------------------------
def draw_face_locations(img, face_locs):
        corner1 = None
        corner2 = None
        len_locs = len(face_locs)
        if len_locs:
                corner1 = np.zeros([len_locs, 2], dtype=int)
                corner2 = np.zeros([len_locs, 2], dtype=int)
                cl_blue = (255, 0, 0)
                i = 0
                for (x,y,w,h) in face_locs:
                        c1 = (h, x)
                        c2 = (y, w)
                        corner1[i,:] = c1
                        corner2[i,:] = c2
                        cv2.rectangle(img, c1, c2, cl_blue, 3)
                        i += 1
        return img, corner1, corner2


#---------------------------------------------------------------------------------
#   Main
#---------------------------------------------------------------------------------
ft_path = "./template/"
name = input(">>> Please type name of the registration person: ")
threshold = 0.4
cap = cv2.VideoCapture(0)

while True:
        # Detect face
        ret, img = cap.read()
        face_locs = face_locations(img)
        img_loc, corner1, corner2 = draw_face_locations(img, face_locs)
```

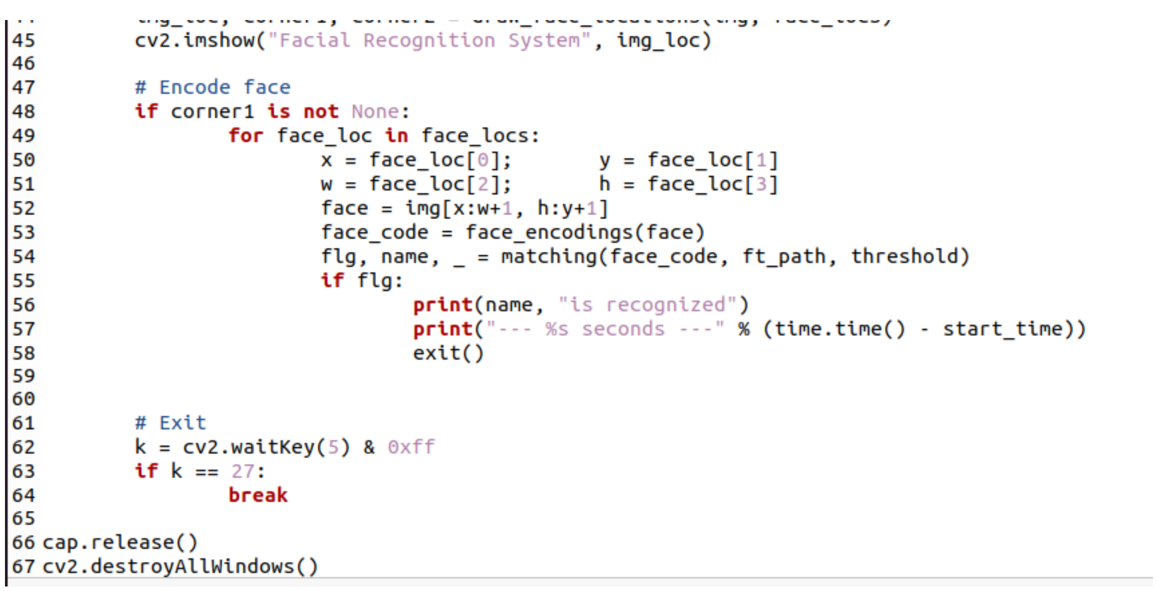


```python
        img_loc, corner1, corner2 = draw_face_locations(img, face_locs)
        cv2.imshow("Facial Recognition System", img_loc)

        # Encode face
        if corner1 is not None:
                len_locs = len(face_locs)
                if len_locs!=1:
                        print("In registration mode, there must be one person in the front of camera")
                else:
                        x = face_locs[0][0];    y = face_locs[0][1]
                        w = face_locs[0][2];    h = face_locs[0][3]
                        face = img[x:w+1, h:y+1]
                        face_code = face_encodings(face)
                        if len(face_code):
                                if isTempExisted(face_code, ft_path, threshold):
                                        print("Your template is registered before")
                                else:
                                        save_dict = {"temp_code": face_code, "face":face}
                                        savemat("%s%s.mat" % (ft_path, name), save_dict)
                                        print("%s, your registration is succesful" % name)
                                        break

        # Exit
        k = cv2.waitKey(5) & 0xff
        if k == 27:
                break

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

verification.py



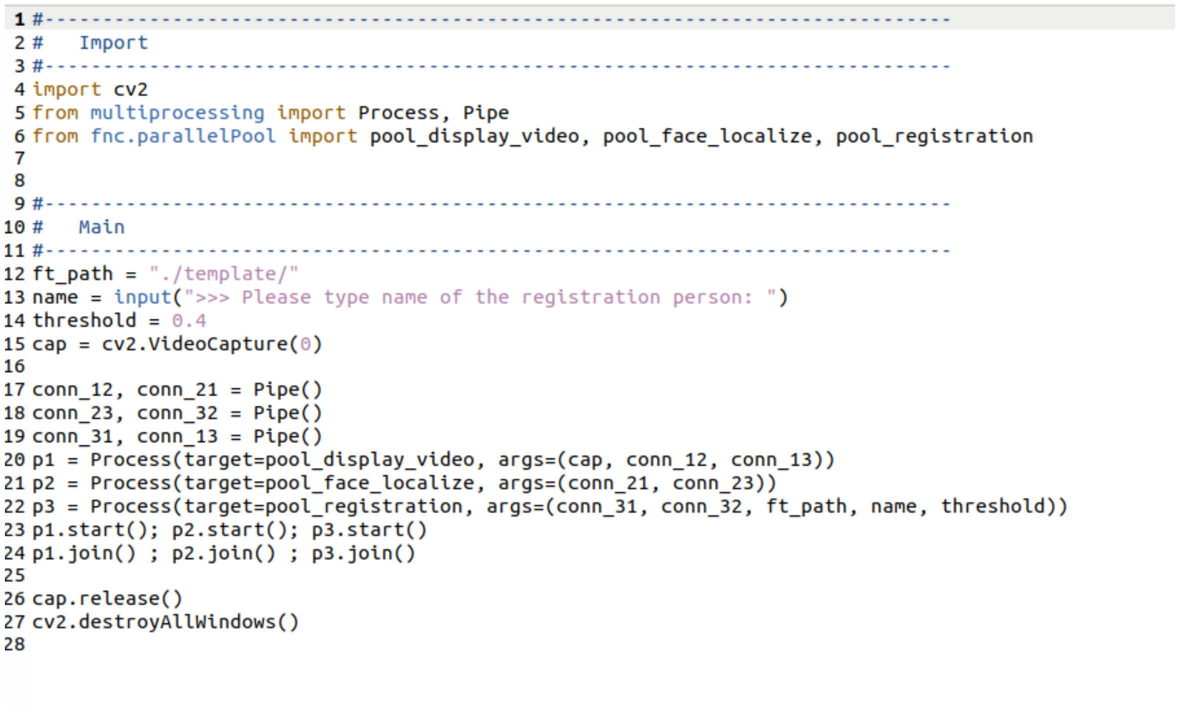
```python
#---------------------------------------------------------------------------
#    Import
#---------------------------------------------------------------------------
import cv2
import numpy as np
from face_recognition import face_locations, face_encodings
from fnc.matching import matching
import time

#---------------------------------------------------------------------------
#    Draw face localtions on an image
#---------------------------------------------------------------------------
def draw_face_locations(img, face_locs):
        corner1 = None
        corner2 = None
        len_locs = len(face_locs)
        if len_locs:
                corner1 = np.zeros([len_locs, 2], dtype=int)
                corner2 = np.zeros([len_locs, 2], dtype=int)
                cl_blue = (255, 0, 0)
                i = 0
                for (x,y,w,h) in face_locs:
                        c1 = (h, x)
                        c2 = (y, w)
                        corner1[i,:] = c1
                        corner2[i,:] = c2
                        cv2.rectangle(img, c1, c2, cl_blue, 3)
                        i += 1
        return img, corner1, corner2


#---------------------------------------------------------------------------
#    Main
#---------------------------------------------------------------------------
start_time = time.time()
ft_path = "template/"
threshold = 0.4
cap = cv2.VideoCapture(0)

while True:
        # Detect face
        ret, img = cap.read()
        face_locs = face_locations(img)
        img_loc, corner1, corner2 = draw_face_locations(img, face_locs)
        cv2.imshow("Facial Recognition System", img_loc)
```



```python
        cv2.imshow("Facial Recognition System", img_loc)
```

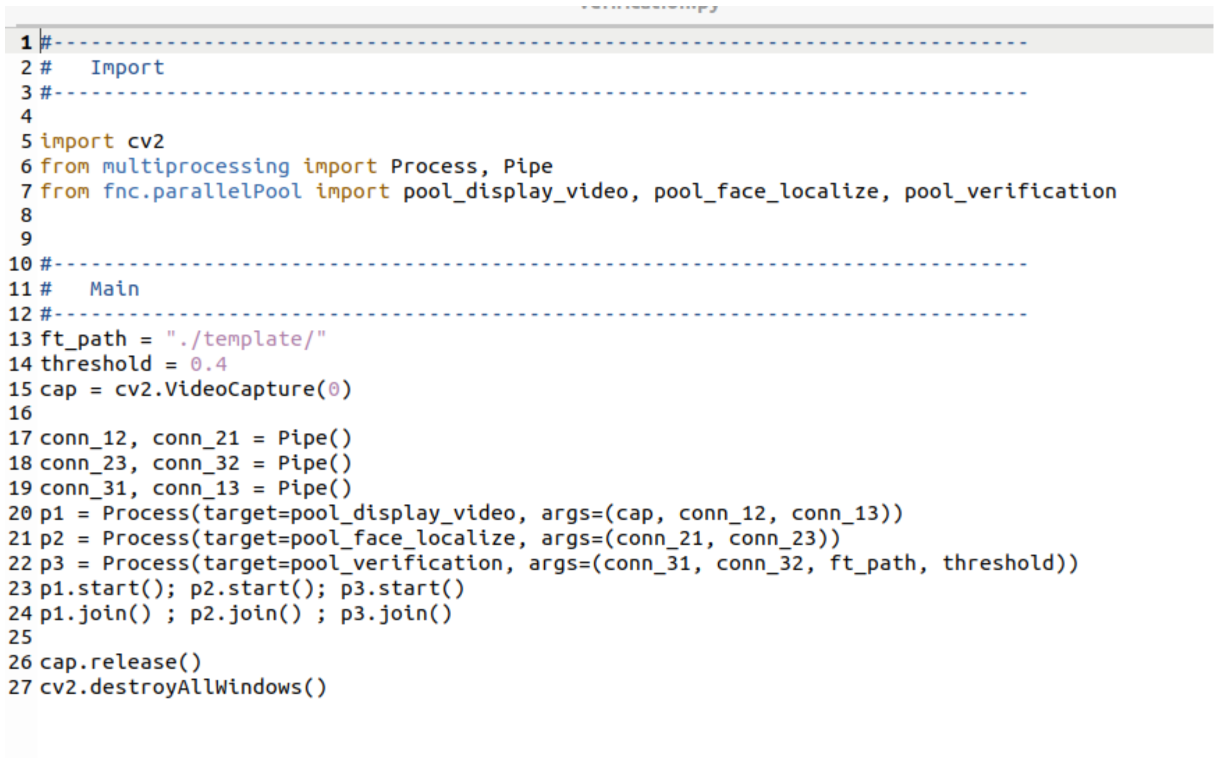
```python

        # Encode face
        if corner1 is not None:
                for face_loc in face_locs:
                        x = face_loc[0];        y = face_loc[1]
                        w = face_loc[2];        h = face_loc[3]
                        face = img[x:w+1, h:y+1]
                        face_code = face_encodings(face)
                        flg, name, _ = matching(face_code, ft_path, threshold)
                        if flg:
                                print(name, "is recognized")
                                print("--- %s seconds ---" % (time.time() - start_time))
                                exit()


        # Exit
        k = cv2.waitKey(5) & 0xff
        if k == 27:
                break

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

PARALLEL SYSTEM - registration.py



```python
#--------------------------------------------------------------------------------
#    Import
#--------------------------------------------------------------------------------
import cv2
from multiprocessing import Process, Pipe
from fnc.parallelPool import pool_display_video, pool_face_localize, pool_registration


#--------------------------------------------------------------------------------
#    Main
#--------------------------------------------------------------------------------
ft_path = "./template/"
name = input(">>> Please type name of the registration person: ")
threshold = 0.4
cap = cv2.VideoCapture(0)

conn_12, conn_21 = Pipe()
conn_23, conn_32 = Pipe()
conn_31, conn_13 = Pipe()
p1 = Process(target=pool_display_video, args=(cap, conn_12, conn_13))
p2 = Process(target=pool_face_localize, args=(conn_21, conn_23))
p3 = Process(target=pool_registration, args=(conn_31, conn_32, ft_path, name, threshold))
p1.start(); p2.start(); p3.start()
p1.join() ; p2.join() ; p3.join()

cap.release()
cv2.destroyAllWindows()

```

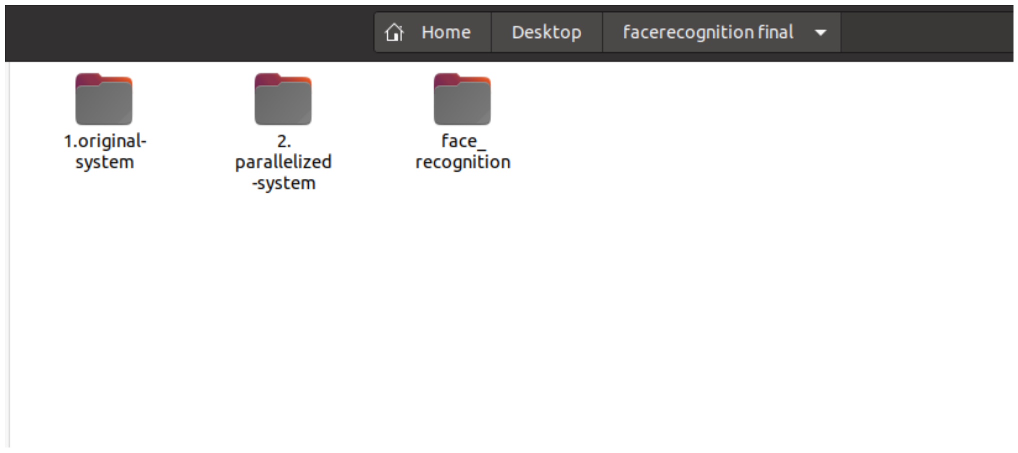
verification.py



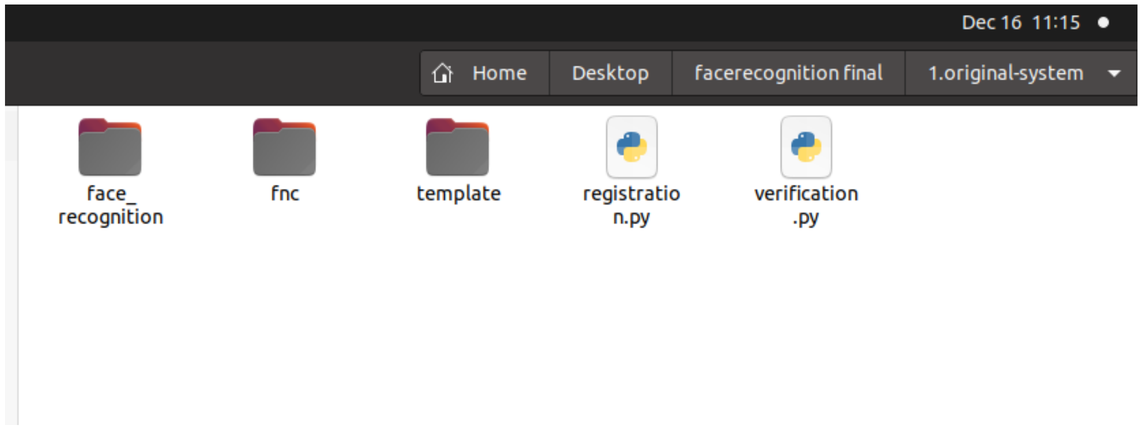
```python
#--------------------------------------------------------------------------------
#    Import
#--------------------------------------------------------------------------------

import cv2
from multiprocessing import Process, Pipe
from fnc.parallelPool import pool_display_video, pool_face_localize, pool_verification


#--------------------------------------------------------------------------------
#    Main
#--------------------------------------------------------------------------------
ft_path = "./template/"
threshold = 0.4
cap = cv2.VideoCapture(0)

conn_12, conn_21 = Pipe()
conn_23, conn_32 = Pipe()
conn_31, conn_13 = Pipe()
p1 = Process(target=pool_display_video, args=(cap, conn_12, conn_13))
p2 = Process(target=pool_face_localize, args=(conn_21, conn_23))
p3 = Process(target=pool_verification, args=(conn_31, conn_32, ft_path, threshold))
p1.start(); p2.start(); p3.start()
p1.join() ; p2.join() ; p3.join()

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

b. Screenshots

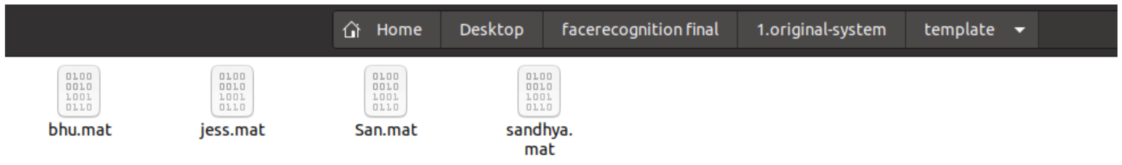
Project folder



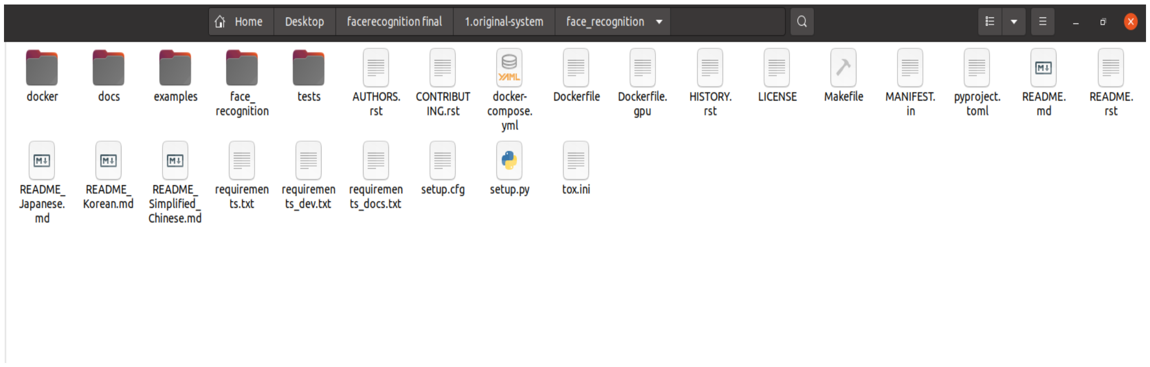
original system folder



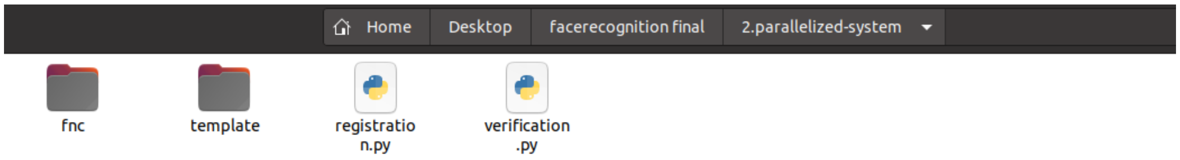
Template - database to store the pictures captured



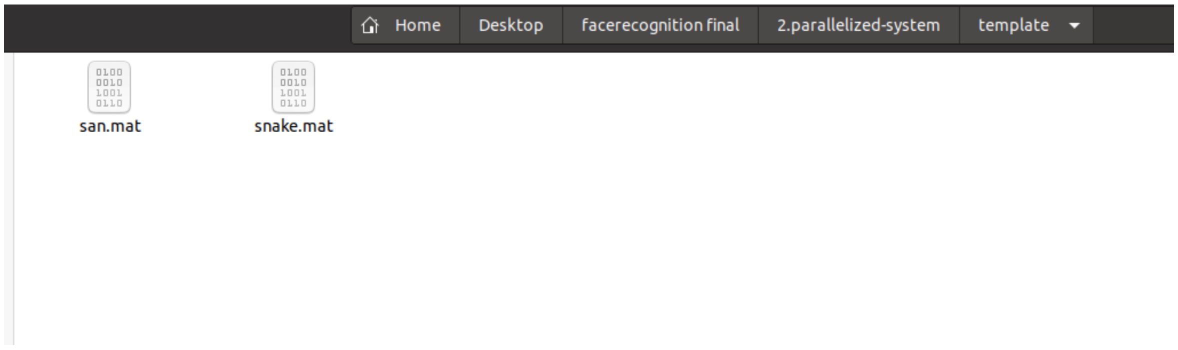
face_recognition module cloned from a github repository



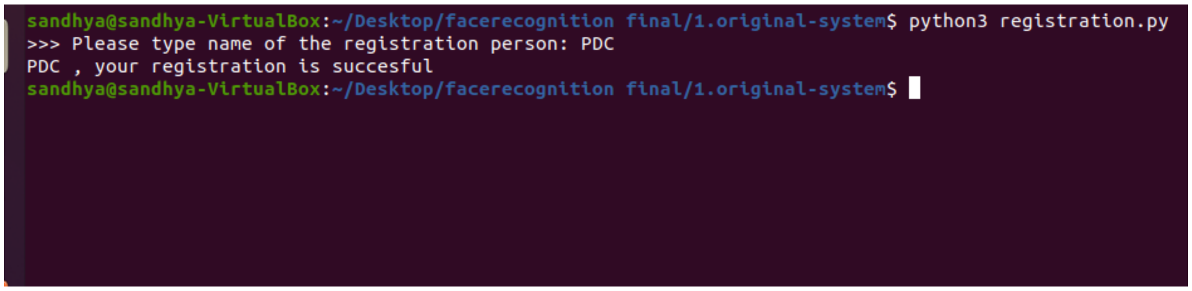
Parallelized system folder



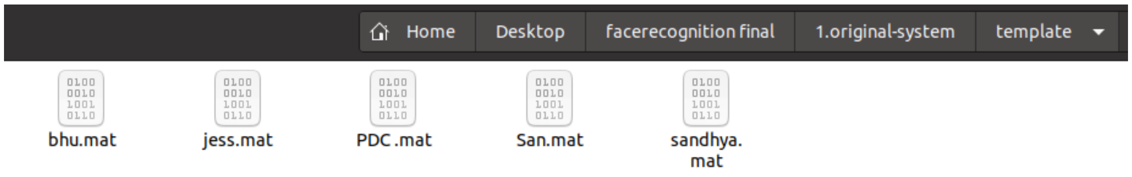
template- database of captured faces in parallelized folder



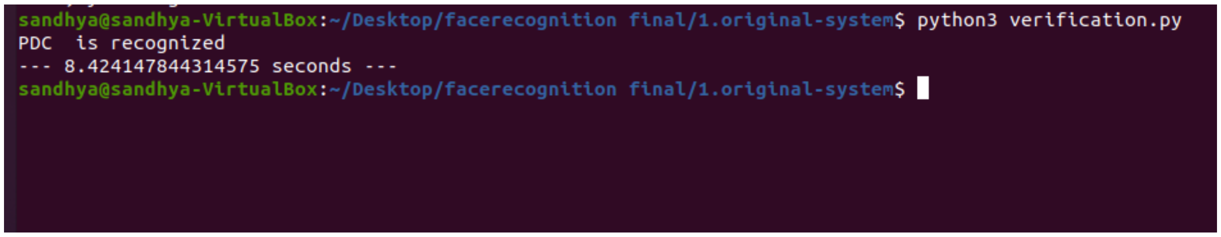
Camera module opens and captures my face and stores it in template folder under the name of PDC



```
sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/1.original-system$ python3 registration.py
>>> Please type name of the registration person: PDC
PDC , your registration is succesful
sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/1.original-system$
```

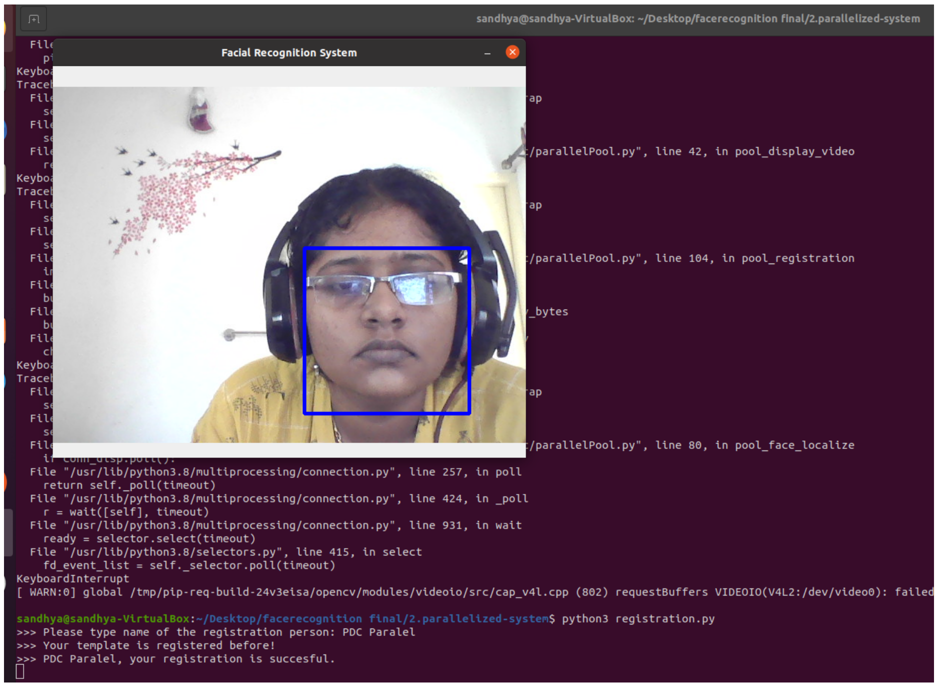


camera module opens when i run verification.py captures my face and recognizes it as PDC and prints the time taken to open camera module capture face compare recognize and print it



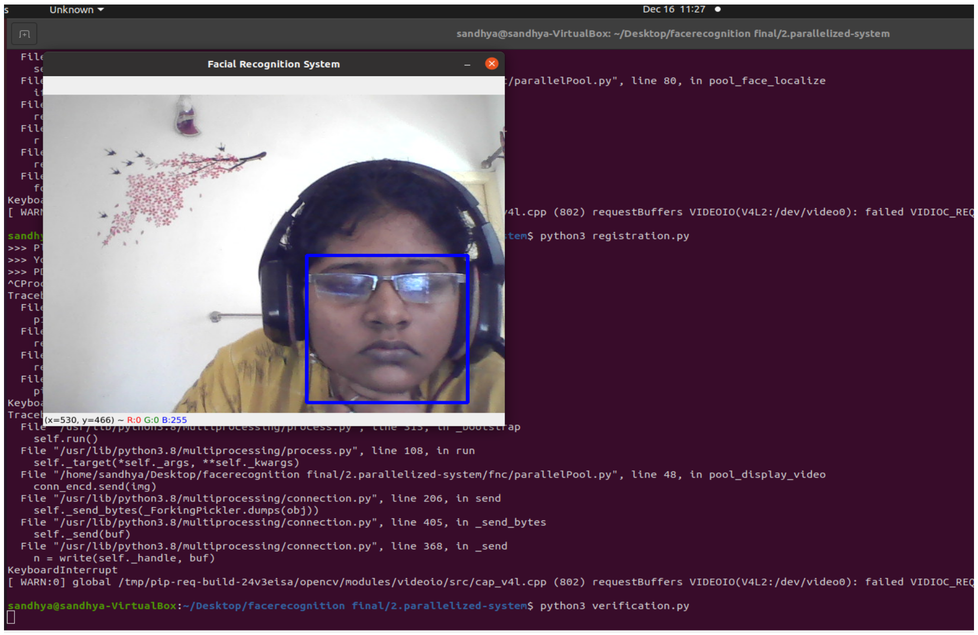
```
sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/1.original-system$ python3 verification.py
PDC  is recognized
--- 8.424147844314575 seconds ---
sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/1.original-system$
```

Initially it says template is already registered because I have already registered my face but then later i put on my headphones with which the previous registration fails and it registers my face with headphones as a new face.


sandhya@sandhya-VirtualBox: ~/Desktop/facerecognition final/2.parallelized-system
Facial Recognition System

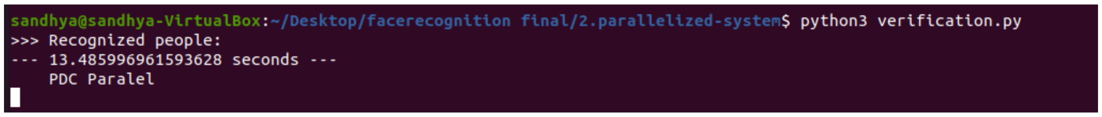
```
/parallelPool.py", line 42, in pool_display_video
/parallelPool.py", line 104, in pool_registration
/parallelPool.py", line 80, in pool_face_localize
  File "/usr/lib/python3.8/multiprocessing/connection.py", line 257, in poll
    return self._poll(timeout)
  File "/usr/lib/python3.8/multiprocessing/connection.py", line 424, in _poll
    r = wait([self], timeout)
  File "/usr/lib/python3.8/multiprocessing/connection.py", line 931, in wait
    ready = selector.select(timeout)
  File "/usr/lib/python3.8/selectors.py", line 415, in select
    fd_event_list = self._selector.poll(timeout)
KeyboardInterrupt
[ WARN:0] global /tmp/pip-req-build-24v3eisa/opencv/modules/videoio/src/cap_v4l.cpp (802) requestBuffers VIDEOIO(V4L2:/dev/video0): failed

sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/2.parallelized-system$ python3 registration.py
>>> Please type name of the registration person: PDC Paralel
>>> Your template is registered before!
>>> PDC Paralel, your registration is succesful.
```

Running verification.py


Unknown
Dec 16 11:27
sandhya@sandhya-VirtualBox: ~/Desktop/facerecognition final/2.parallelized-system
Facial Recognition System
(x=530, y=466) ~ R:0 G:0 B:255

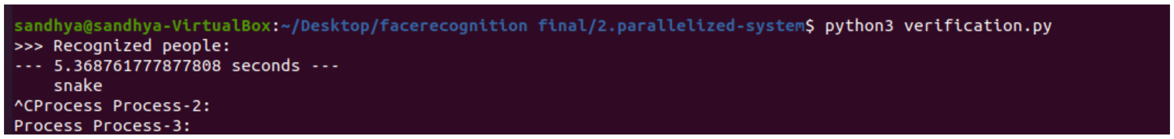
```
/parallelPool.py", line 80, in pool_face_localize
4l.cpp (802) requestBuffers VIDEOIO(V4L2:/dev/video0): failed VIDIOC_REQ
tem$ python3 registration.py
    self.run()
  File "/usr/lib/python3.8/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/home/sandhya/Desktop/facerecognition final/2.parallelized-system/fnc/parallelPool.py", line 48, in pool_display_video
    conn_encd.send(img)
  File "/usr/lib/python3.8/multiprocessing/connection.py", line 206, in send
    self._send_bytes(_ForkingPickler.dumps(obj))
  File "/usr/lib/python3.8/multiprocessing/connection.py", line 405, in _send_bytes
    self._send(buf)
  File "/usr/lib/python3.8/multiprocessing/connection.py", line 368, in _send
    n = write(self._handle, buf)
KeyboardInterrupt
[ WARN:0] global /tmp/pip-req-build-24v3eisa/opencv/modules/videoio/src/cap_v4l.cpp (802) requestBuffers VIDEOIO(V4L2:/dev/video0): failed VIDIOC_REQ

sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/2.parallelized-system$ python3 verification.py
```

This was the glitch I mentioned in the future scope of the project column. The time taken for it to open the camera module is very high. Hence the large time duration. If a better camera module is used in the project then better results can be provided.



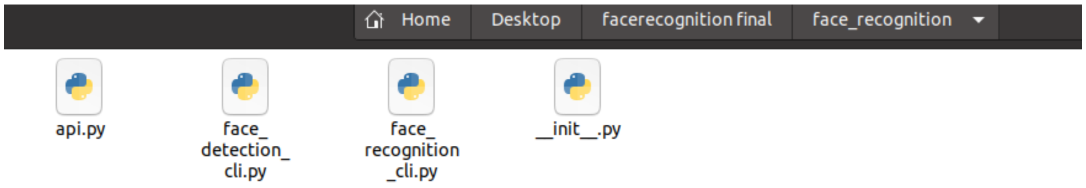
```
sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/2.parallelized-system$ python3 verification.py
>>> Recognized people:
--- 13.485996961593628 seconds ---
    PDC Paralel
```

Sometimes it recognizes us faster. It all depends on the pace at which the camera module opens



```
sandhya@sandhya-VirtualBox:~/Desktop/facerecognition final/2.parallelized-system$ python3 verification.py
>>> Recognized people:
--- 5.368761777877808 seconds ---
    snake
^CProcess Process-2:
Process Process-3:
```

These are some libraries and API's which were installed in the Ubuntu virtual machine to create the face recognition system.



My GITHUB Repository with the codes

https://github.com/Saannndddyyyyy/Face-Recognition-system-using-parallel-computing