

**Name**: Ekene Ike-Okoro Anthony

**Company**: Robotics Centre HQ

**Position**: Intern

Project Topic:

Face-Recognition System

Description:

A facial recognition system is a technology capable of matching a human face from a digital image or a video frame against a database of faces, typically employed to authenticate users through ID verification services. The system is divided into two parts. The first part is used for capturing, classifying and storing users faces in the database which would later be used for recognition. The second part captures live images or videos and compares them to images stored in the database. If a user’s face is matched with one in the database, he is she is granted access into the system, else he is she is locked out.

Components Used:

* Raspberry Pi
* SD Card
* Relay
* Solenoid lock
* Power pack
* Raspberry Pi Camera

Challenges:

1. Getting the required packages needed for the software was difficult to find and install’
2. Creating an efficient and fast code for comparing stored faces with live images and videos was tedious

How Challenges Were Resolved:

The first issue was resolved through the help of the links below:

<https://github.com/ageitgey/face_recognition/issues/175#issue-257710508>

<https://docs.opencv.org/4.0.0/d7/d8b/tutorial_py_face_detection.html>

The second issue was resolved by make use of an improved package gotten from:

<https://github.com/ageitgey/face_recognition>

Block-Diagram:

Raspberry

Pi

Camera

Raspberry Pi

SD

Card

Power Pack

Relay

Solenoid

Lock

**code:**

part1:

import cv2

import numpy as np

import os

cap=cv2.VideoCapture(0)

face\_cascade=cv2.CascadeClassifier("haarcascade\_frontalface\_alt.xml")

skip, count=0, 0

face\_data=[]

path="./interns\_dataset/"

name=input("Enter person's name: ")

while True:

ret, frame=cap.read()

gray=cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces= face\_cascade.detectMultiScale(gray, 1.3, 5) #parameters(name, scaling factor, number of neighbors)

if len(faces)==0:

continue

k=1

faces=sorted(faces, key=lambda x: x[2]\*x[3], reverse=True)

skip+=1

for face in faces: #possible error

x,y,w,h=face

offset=0

face\_offset= frame[y-offset:y+h+offset, x-offset:x+w+offset]

face\_selection=cv2.resize(face\_offset, (200, 200))

if skip%5==0:

cv2.imwrite(os.path.join(path , name + str(count+10)+".png"), face\_selection)

count+=1

cv2.imshow(str(k), face\_selection)

k+=1

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)

skip+=1

cv2.imshow("faces", frame)

key\_pressed=cv2.waitKey(1)

if key\_pressed==ord("q") or count==20:

cap.release()

break

print("Dataset saved at: {}".format(path+name+".npy"))

cap.release()

cv2.destroyAllWindows()

Part2:

import face\_recognition

import cv2

import numpy as np

import os

# 1. Process each video frame at 1/4 resolution (though still display it at full resolution)

# 2. Only detect faces in every other frame of video.

# Get a reference to webcam #0 (the default one)

video\_capture = cv2.VideoCapture(0)

path="./interns\_dataset/"

known\_face\_names = []

known\_face\_encodings = []

class\_id=0

for fx in os.listdir(path):

if fx.endswith('.png'):

img = face\_recognition.load\_image\_file(fx[:])

img\_face\_encoding = face\_recognition.face\_encodings(img)[0]

known\_face\_names [class\_id] = fx[:-6]

known\_face\_encodings [class\_id] = img\_face\_encoding

class\_id += 1

# Initialize some variables

face\_locations = []

face\_encodings = []

face\_names = []

process\_this\_frame = True

while True:

# Grab a single frame of video

ret, frame = video\_capture.read()

# Resize frame of video to 1/4 size for faster face recognition processing

small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

# Convert the image from BGR color (which OpenCV uses) to RGB color (which face\_recognition uses)

rgb\_small\_frame = small\_frame[:, :, ::-1]

# Only process every other frame of video to save time

if process\_this\_frame:

# Find all the faces and face encodings in the current frame of video

face\_locations = face\_recognition.face\_locations(rgb\_small\_frame)

face\_encodings = face\_recognition.face\_encodings(rgb\_small\_frame, face\_locations)

face\_names = []

for face\_encoding in face\_encodings:

# See if the face is a match for the known face(s)

matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding)

name = "Unknown"

# # If a match was found in known\_face\_encodings, just use the first one.

# if True in matches:

# first\_match\_index = matches.index(True)

# name = known\_face\_names[first\_match\_index]

# Or instead, use the known face with the smallest distance to the new face

face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

best\_match\_index = np.argmin(face\_distances)

if matches[best\_match\_index]:

name = known\_face\_names[best\_match\_index]

face\_names.append(name)

process\_this\_frame = not process\_this\_frame

# Display the results

for (top, right, bottom, left), name in zip(face\_locations, face\_names):

# Scale back up face locations since the frame we detected in was scaled to 1/4 size

top \*= 4

right \*= 4

bottom \*= 4

left \*= 4

# Draw a box around the face

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# Draw a label with a name below the face

cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)

font = cv2.FONT\_HERSHEY\_DUPLEX

cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

# Display the resulting image

cv2.imshow('Video', frame)

# Hit 'q' on the keyboard to quit!

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release handle to the webcam

video\_capture.release()

cv2.destroyAllWindows()