ESCUELA DE INGENIERÍA MECÁNICA ELÉCTRICA
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Proyectos de computación aplicados a I.E.

Tarea # 3

David Antonio Rodas Alvarez

202010039

Fecha: Guatemala, 9 de agosto del 2024

Tarea # 3[*]

David Antonio, Rodas Alvarez, 202010039[1, **]
[1] *Escuela de Ingeniería Mecánica Eléctrica , Universidad de San Carlos, Guatemala.*
(Dated: 9 de agosto de 2024)

I. RESUMEN

La tercera tarea del curso, consistió en cómo realizar un reconocimiento facial en Python utilizando OpenCV. En este reporte se detalla los pasos desde la instalación de OpenCV, la recolección de datos (rostros), el entrenamiento del clasificador con métodos como Eigenfaces, Fisherfaces y Local Binary Patterns Histograms (LBPH), hasta la prueba del modelo. Al igual se incluyen fragmentos de código y explicaciones sobre el uso de bibliotecas como cv2, os, numpy, y imutils. Finalmente, se menciona las simulaciones hechas con el programa.

II. MARCO TEÓRICO

OpenCV

(Open Source Computer Vision Library) es una biblioteca ampliamente utilizada en Python para realizar tareas de visión por computadora. Proporciona herramientas para procesar imágenes y videos, permitiendo la detección de objetos, reconocimiento facial, y seguimiento de movimientos, entre otros. OpenCV soporta múltiples lenguajes de programación y sistemas operativos, lo que lo convierte en una opción versátil para desarrolladores. Su capacidad de trabajar en tiempo real es clave en aplicaciones como el reconocimiento facial, donde la velocidad y precisión son esenciales.

Uno de los aspectos más destacados de OpenCV es su capacidad para implementar algoritmos complejos de manera eficiente. Por ejemplo, la detección de rostros en imágenes se puede realizar utilizando clasificadores en cascada basados en características Haar, los cuales son preentrenados y optimizados para detectar caras de manera rápida. OpenCV también soporta el uso de redes neuronales y aprendizaje automático, permitiendo a los desarrolladores crear sistemas de reconocimiento más avanzados y personalizados.

Además de sus funcionalidades de detección, OpenCV permite manipular y transformar imágenes con facilidad. Funciones como el redimensionamiento, la conversión de color, y la aplicación de filtros son comunes en las aplicaciones que requieren el preprocesamiento de imágenes antes de aplicar algoritmos de reconocimiento. Esto hace que OpenCV sea una herramienta integral tanto para desarrolladores novatos como para expertos en visión artificial.

NumPy

NumPy es una biblioteca fundamental en Python para el manejo de matrices y la computación numérica. Se utiliza ampliamente en el procesamiento de imágenes, ya que permite manipular píxeles y realizar operaciones matemáticas complejas de manera eficiente. NumPy proporciona una estructura de datos llamada .ªrray"que es mucho más rápida y flexible que las listas nativas de Python, lo que la hace ideal para tareas que involucran grandes volúmenes de datos.

En el contexto del reconocimiento facial, NumPy se utiliza para manejar y transformar datos de imágenes antes de aplicar algoritmos de visión por computadora. Por ejemplo, una imagen se puede representar como un array multidimensional en NumPy, donde cada elemento del array corresponde a un píxel. Esto permite realizar operaciones como el filtrado de imágenes, la normalización de los datos y la transformación de matrices, todas cruciales para el entrenamiento y la prueba de modelos de reconocimiento facial.

Además, NumPy se integra perfectamente con otras bibliotecas de Python como OpenCV, TensorFlow, y SciPy, lo que facilita la construcción de pipelines de procesamiento de imágenes y aprendizaje automático. Su eficiencia y capacidad para realizar operaciones vectorizadas lo convierten en una herramienta esencial en la ciencia de datos y la visión por computadora, permitiendo a los desarrolladores manejar y procesar grandes cantidades de datos con facilidad.

OS

El módulo os en Python es una biblioteca estándar que proporciona una forma de interactuar con el sistema operativo. Es especialmente útil en aplicaciones que requieren gestión de archivos y directorios, como es el caso del reconocimiento facial, donde las imágenes deben ser almacenadas, organizadas y accesadas eficientemente.

[*] Escuela de Ingeniería Mecánica Eléctrica
[**] e-mail: 3711111370101@ingenieria.usac.edu.gt

Con os, los desarrolladores pueden automatizar tareas de sistema como la creación de carpetas, la lectura de archivos, y la navegación en directorios.

En el proceso de reconocimiento facial, el módulo os se utiliza comúnmente para gestionar grandes conjuntos de datos de imágenes. Por ejemplo, al entrenar un modelo de reconocimiento, os permite recorrer directorios que contienen imágenes etiquetadas de diferentes personas, accediendo a cada archivo para procesarlo individualmente. Esto es crucial para organizar datos de entrenamiento de manera que el modelo pueda ser entrenado de forma precisa y eficiente.

Además, os facilita la portabilidad del código Python entre diferentes sistemas operativos. Al usar funciones como os.path, los desarrolladores pueden escribir código que maneje rutas y archivos de manera compatible tanto en Windows, macOS, como en Linux. Esto es especialmente importante en proyectos de visión por computadora donde los datos pueden estar alojados en diferentes plataformas y el código necesita ser adaptable.

Haarcascades

Haarcascades es un algoritmo de detección de objetos, comúnmente utilizado en la detección de rostros en imágenes y videos. Se basa en el uso de clasificadores en cascada entrenados con características Haar, que son patrones simples de contraste de píxeles. Estos clasificadores pueden identificar regiones de interés en una imagen, como ojos, nariz, y boca, que son característicos de un rostro humano.

El entrenamiento de un clasificador Haarcascade implica proporcionar al algoritmo un conjunto de imágenes positivas (con rostros) y negativas (sin rostros). A partir de estas imágenes, el algoritmo aprende a distinguir las características faciales mediante la combinación de varias características Haar simples en clasificadores más complejos y precisos. El resultado es un clasificador en cascada que puede detectar rostros de manera rápida y eficaz en nuevas imágenes.

Una de las ventajas de Haarcascades es su rapidez en la detección, lo que lo hace adecuado para aplicaciones en tiempo real. Sin embargo, su precisión puede verse afectada por condiciones de iluminación variables y oclusiones parciales del rostro. A pesar de estas limitaciones, sigue siendo una opción popular debido a su simplicidad y velocidad, y es ampliamente utilizado en aplicaciones de visión por computadora donde se requiere detección de rostros en tiempo real.

Eigenfaces

El método de Eigenfaces es una técnica de reconocimiento facial basada en el Análisis de Componentes Principales (PCA). La idea central es reducir la dimensionalidad de los datos faciales al proyectar las imágenes en un espacio de menor dimensión, capturando las características más importantes que distinguen una cara de otra. Las .eigenfaces"son los vectores propios (eigenvectors) que resultan de este proceso, y representan patrones básicos que, combinados, pueden reconstruir una cara.

El proceso de entrenamiento con Eigenfaces implica tomar un conjunto de imágenes faciales y calcular su media. Luego, se realiza la descomposición en valores singulares (SVD) para encontrar las eigenfaces que mejor representan las variaciones dentro del conjunto de datos. Cada imagen en el conjunto de datos original puede ser representada como una combinación lineal de estas eigenfaces, lo que permite clasificar nuevas imágenes basadas en su proyección en este espacio reducido.

Aunque Eigenfaces es un método eficiente y ha sido ampliamente utilizado en el pasado, tiene limitaciones. Es sensible a cambios en la iluminación, poses y expresiones faciales, lo que puede afectar su precisión. Sin embargo, sigue siendo una técnica fundamental en la historia del reconocimiento facial y se utiliza a menudo como punto de partida para desarrollar métodos más robustos.

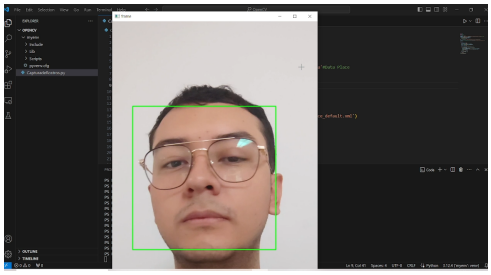
III. CÓDIGO

A. Capturación de Rostros

```
import cv2
import os
import imutils

personName = 'DavidR'
dataPath = 'C:/Users/Lenovo/Documents/Proyectos
     DavidR/Reconocimiento Facial/Data'#Data Place
personPath = dataPath + '/' + personName
if not os.path.exists(personPath):
    print('Carpeta creada: ',personPath)
    os.makedirs(personPath)

cap = cv2.VideoCapture('C:/Users/Lenovo/Documents/Proyectos
     DavidR/David4.mp4')

faceClassif =
     cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_
frontalface_default.xml')
count = 900

while True:

    ret, frame = cap.read()
    if ret == False: break
    frame = imutils.resize(frame, width=640)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    auxFrame = frame.copy()

    faces = faceClassif.detectMultiScale(gray,1.3,5)
```

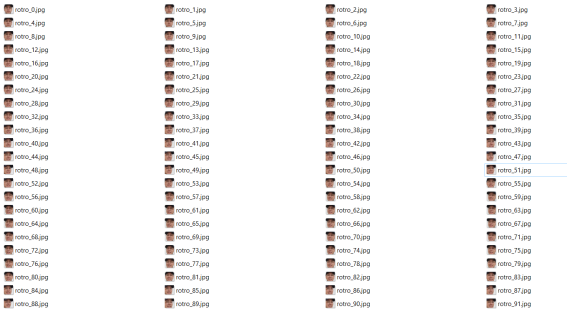
```
    for (x,y,w,h) in faces:
        cv2.rectangle(frame, (x,y),(x+w,y+h),(0,255,0),2)
        rostro = auxFrame[y:y+h,x:x+w]
        rostro = cv2.resize(rostro,(150,150),interpolation
               =cv2.INTER_CUBIC)
        cv2.imwrite(personPath +
              '/rotro_{}.jpg'.format(count),rostro)
        count = count + 1
    cv2.imshow('frame',frame)

    k =  cv2.waitKey(1)
    if k == 27 or count >= 1200:
        break
cap.release()
cv2.destroyAllWindows()
```

Figura 1: Generando imágenes



Fuente: Elaboración Propia, 2024.

Figura 2: Carpeta con las imágenes



Fuente: Elaboración Propia, 2024.

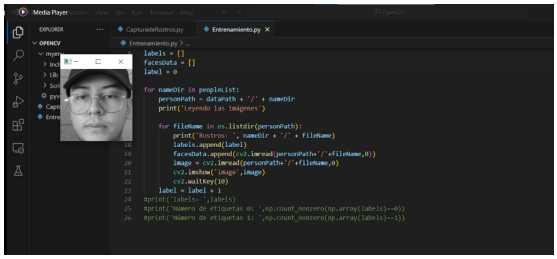
Figura 3: Carpeta con las imágenes (1199 imágenes)

Fuente: Elaboración Propia, 2024.

B. Entrenamiento del Detector Facial

```
import cv2
import os
import numpy as np
dataPath = 'C:/Users/Lenovo/Documents/Proyectos
     DavidR/Reconocimiento Facial/Data'#Data Place
peopleList = os.listdir(dataPath)
print('Lista de personas: ', peopleList)

labels = []
facesData = []
label = 0

for nameDir in peopleList:
    personPath = dataPath + '/' + nameDir
    print('Leyendo las imgenes')

    for fileName in os.listdir(personPath):
        print('Rostros: ', nameDir + '/' + fileName)
        labels.append(label)
        facesData.append(cv2.imread(personPath+'/'+fileName,0))
        image = cv2.imread(personPath+'/'+fileName,0)
        #cv2.imshow('image',image)
        #cv2.waitKey(10)
    label = label + 1
#print('labels= ',labels)
#print('Nmero de etiquetas 0:
     ',np.count_nonzero(np.array(labels)==0))
#print('Nmero de etiquetas 1:
     ',np.count_nonzero(np.array(labels)==1))

face_recognizer = cv2.face.EigenFaceRecognizer_create()
#face_recognizer = cv2.face.FisherFaceRecognizer_create()
#face_recognizer = cv2.face.LBPHFaceRecognizer_create()

#Entrenando el reconocedor de rostros
print("Entrenando")
face_recognizer.train(facesData, np.array(labels))

#Almacenando el modelo obtenido
face_recognizer.write('modeloEigenFace.xml')
print("Modelo almacenado...")
```

Figura 4: Entrenamiento en proceso - (Escala de grises)

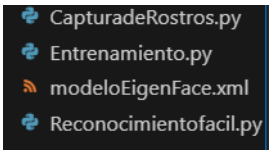


Fuente: Elaboración Propia, 2024.

Figura 5: Detector Facial analizando cada imagen

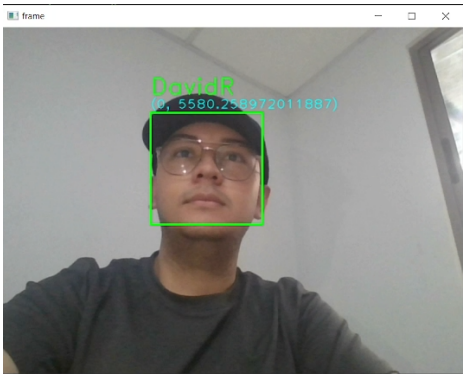
Fuente: Elaboración Propia, 2024.

Figura 6: Modelo Detector Facial (.XML) - Creado



Fuente: Elaboración Propia, 2024.

Figura 7: Reconocimiento Facial #1

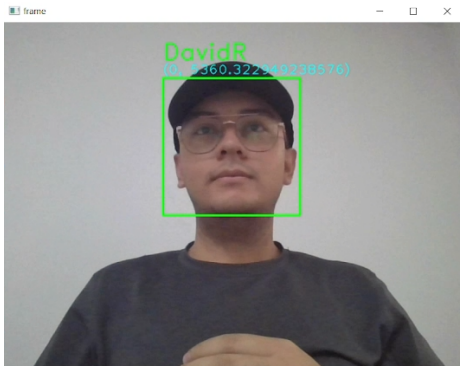


Fuente: Elaboración Propia, 2024.

C. Reconocimiento Facial

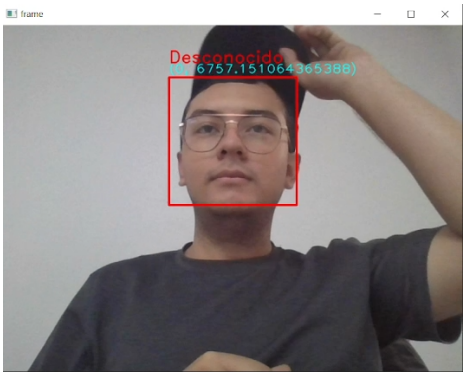
```
    import cv2
import os
dataPath = 'C:/Users/Lenovo/Documents/Proyectos
     DavidR/Reconocimiento Facial/Data' #Data Place
imagePaths = os.listdir(dataPath)
print('imagePaths=',imagePaths)
face_recognizer = cv2.face.EigenFaceRecognizer_create()
#face_recognizer = cv2.face.FisherFaceRecognizer_create()
#face_recognizer = cv2.face.LBPHFaceRecognizer_create()

# Leyendo el modelo
face_recognizer.read('modeloEigenFace.xml')
#face_recognizer.read('modeloFisherFace.xml')
#face_recognizer.read('modeloLBPHFace.xml')

#Tipo de medio de deteccin
cap = cv2.VideoCapture(0,cv2.CAP_DSHOW) #camara
#cap = cv2.VideoCapture('C:/Users/Lenovo/Documents/Proyectos
     DavidR/David1.mp4') #video

faceClassif = cv2.CascadeClassifier(cv2.data.haarcascades
+'haarcascade_frontalface_default.xml')
while True:
    ret,frame = cap.read()
    if ret == False: break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    auxFrame = gray.copy()
    faces = faceClassif.detectMultiScale(gray,1.3,5)
    for (x,y,w,h) in faces:
        rostro = auxFrame[y:y+h,x:x+w]
        rostro = cv2.resize(rostro,(150,150),interpolation=
             cv2.INTER_CUBIC)
        result = face_recognizer.predict(rostro)
cv2.putText(frame,'{}'.format(result),(x,y-5),1,1.3,(255,255,0),1,cv2.LINE_AA)

        # EigenFaces
        if result[1] < 5700:
            cv2.putText(frame,'{}'.format(imagePaths[result[0]]),(x,y-25),2,1.1,
(0,255,0),1,cv2.LINE_AA)
cv2.rectangle(frame, (x,y),(x+w,y+h),(0,255,0),2)
        else:
cv2.putText(frame,'Desconocido',(x,y-20),2,0.8,(0,0,255),1,cv2.LINE_AA)
cv2.rectangle(frame, (x,y),(x+w,y+h),(0,0,255),2)


    cv2.imshow('frame',frame)
    k = cv2.waitKey(1)
    if k == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

Figura 8: Reconocimiento Facial #2



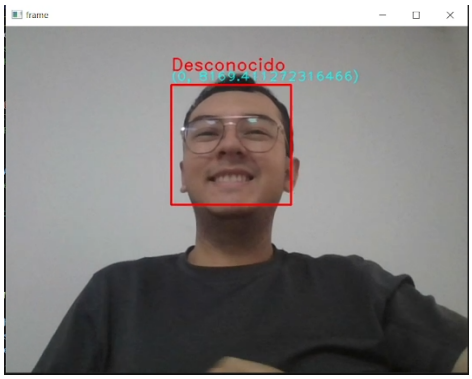
Fuente: Elaboración Propia, 2024.

Figura 9: Reconocimiento Facial #3 (Quitando Gorra)



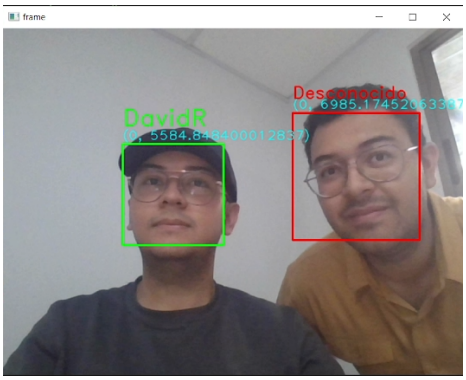
Fuente: Elaboración Propia, 2024.

Figura 10: Reconocimiento Facial #4 (Sin Gorra)



Fuente: Elaboración Propia, 2024.

Figura 11: Prueba de reconocimiento (Hermanos)



Fuente: Elaboración Propia, 2024.

D. Videos en Drive

-Click en este texto-

`https://drive.google.com/drive/folders/1eGHq1UVBP99vmp_UyrpCE5fzgfeDYSNC?usp=sharing`

E. Link de GitHub

`https://github.com/dvd-r16/proyectos`

IV. CONCLUSIONES

1. Se logro registrar una cara para la demostración, al igual entrenar a nuestro detector facial para la identifiación de un usuario, siendo en este caso, el estudiante David Rodas.
2. Se debe realizar múltiples pruebas a diferentes iluminaciones para la óptima detección de rostro, ya que se obtuvo problemas con prendas de vestir que llegan a afectar la detección del usuario.
3. Se logró entender los conceptos principales para el uso de OpenCV para el proceso de imagenes y videos, y su importancia en la detección de objetos y rostros en movimiento..

[1] Ing. Jose Anibal Silva de los Angeles. Proyectos Aplicados para I.E. *Programa del curso*. Ciudad de Guatemala: Universidad de San Carlos, Facultad de Ingeniería, Escuela de Ingeniería Mecánica Eléctrica.