

Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Implementación del Algoritmo de Eigenfaces y las Características de Haar para Autenticación Facial por medio de un Raspberry Pi 3

Modalidad: Proyecto Tecnológico
Trimestre 2018 Invierno

Alumno
Mario Alberto López Reyes
Matrícula: 2132000243
al2132000243@correo.azc.uam.mx

Asesor
Mario Alberto Lagos Acosta
Departamento de Electrónica
mlagos@correo.azc.uam.mx

26 de marzo de 2018

Declaratoria

Yo, Mario Alberto Lagos Acosta, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Mario Alberto Lagos Acosta

Yo, Mario Alberto López Reyes, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Mario Alberto López Reyes

Resumen

En el presente proyecto tecnológico se aborda el problema de reconocimiento facial a través del algoritmo de Eigenfaces y las características de Haar, utilizando un sistema embebido Raspberry Pi 3 con su respectiva cámara (*Raspberry Pi Camera*) y la librería de visión por computadora OpenCV.

Se presentan cada una de las etapas comprendidas por el sistema, y al final, se muestran los resultados haciendo énfasis a lo que se esperaba en la parte teórica.

Índice

1. Introducción	1
2. Antecedentes	2
2.1. Proyectos de Integración o Terminales de la División de Ciencias Básicas e Ingeniería de la UAM	2
2.2. Artículos	2
3. Justificación	3
4. Objetivos	4
4.1. Objetivo General	4
4.2. Objetivos Específicos	4
5. Marco Teórico	5
5.1. Visión por Computadora	5
5.2. Características de Haar	5
5.3. Análisis de Componentes Principales (PCA)	7
5.3.1. PCA para la representación de Eigenfaces	8
5.4. Descomposición en Valores Singulares (SVD)	9
5.5. Proceso de Generación de Eigenfaces	11
5.6. Sistema de Reconocimiento	13
5.6.1. Normalización o Preprocesamiento de la Imagen	13
5.6.2. Extracción de Características	13
5.6.3. Fase de Reconocimiento	14
6. Desarrollo	15
6.1. Recursos	15
6.1.1. Hardware	15
6.1.2. Software	17
6.2. Diseño del Sistema	18

7. Resultados	19
7.1. Identificación mediante características de Haar	19
7.2. Preprocesamiento de la imagen	20
7.3. Extracción de características	21
7.4. Fase de reconocimiento	22
7.5. Autenticación del usuario	22
8. Conclusiones	25
A. Instalación de Raspbian	27
B. Instalación de OpenCV	31
C. Código	36

Índice de figuras

1. Diagrama general de la visión por computadora. La imagen es procesada, se extraen sus atributos y a la salida se da una descripción detallada de esta.	5
2. (a) Características de Haar, (b) Características de Haar aplicadas a una imagen, (c) Imagen integral del píxel $I(i,j)$, (d) Cómputo de la imagen integral para el rectángulo $D = I1 - I2 - I3 + I4$, donde $I1, I2, I3$ e $I4$ son las imágenes integrales en las coordenadas $(a,c), (a,d), (b,c), (b,d)$.	6
3. Ejemplo gráfico de una Descomposición en Valores Singulares.	10
4. Primeras cinco Eigenfaces generadas a partir de 57 imágenes de rostros normalizados.	11
5. Proceso de generación de Eigenfaces.	11
6. Placa de desarrollo Raspberry Pi 3 Modelo B.	16
7. Raspberry Pi Camera V1.	16
8. Logotipo oficial de Raspbian.	17
9. Logotipo oficial de OpenCV.	17
10. Sistema de reconocimiento modular en Raspberry Pi 3.	18
11. Captura de imágenes para el conjunto positivo de entrenamiento.	19

12.	(a) Rostros capturados para el conjunto positivo de entrenamiento. (b) Rostros capturados para el conjunto negativo de entrenamiento.	20
13.	Ventana de entrenamiento del sistema.	21
14.	(a) Eigenface del promedio de todos los rostros capturados. (b) Eigenface de rostros del conjunto positivo. (c) Eigenface de rostros del conjunto negativo.	21
15.	Rostro capturado, Eigenface positiva y media de: (a) Capturas con poca exposición de luz. (b) Capturas con exposición de luz media. (c) Capturas con alta exposición de luz.	23
16.	(a) Resultados con poca exposición de luz. (b) Resultados con exposición de luz media. (c) Resultados con alta exposición de luz.	24
17.	Ventana de configuración de Raspbian.	28
18.	Ventana de configuración de Raspbian para elegir el tipo de Inicio.	29
19.	Escritorio de Raspbian Stretch.	30
20.	Ambiente Virtual en la terminal de Raspbian.	33
21.	Instalación de OpenCV 3 en Raspbian Stretch 2017-09-07.	34
22.	Confirmación de que OpenCV 3 ha sido instalado exitosamente en Raspbian Stretch.	35

1. Introducción

Utilizar rasgos y características tanto físicas como conductuales de los individuos como una herramienta de identificación se llevaba a cabo desde tiempos antiguos. Se sabe que, en el siglo XIV en China, mercaderes estampaban huellas de la palma de la manos y pies de niños en un papel con tinta para distinguirlos unos de otros. [1]

A partir del estudio detallado de la biometría dactilar se abre paso a un método de clasificación que permite identificar las características propias de las huellas y cómo pueden ser diferenciadas. El cálculo de la probabilidad de que dos huellas sean parecidas es de 1 en 64 billones, llevando así a la biometría a buscar un métodos más seguros e infalibles de autenticación.

Es en los noventa cuando se puede hablar del surgimiento de la biometría facial tal y como la entendemos hoy en día, aunque su implementación práctica llegaría en 2001, con el evento deportivo *Super Bowl XXXV* de la NFL (*National Football League*), donde secretamente se archivaron fotografías de los sistemas de vigilancia y se compararon con bases de datos digitales para detectar posibles criminales. [2]

La identificación por medio del reconocimiento facial ha crecido últimamente debido al avance en tecnologías de fotografía y vídeo, las exigencias de sistemas de seguridad son cada vez mayores, propiciando un aumento de cámaras en los lugares de trabajo y residenciales a un reducido costo, además de ser un gran negocio para empresas de tecnología y un campo de estudios amplio para el cómputo.

Utilizar los rasgos faciales propios de cada persona, es una de las formas más seguras que muchas empresas están utilizando para permitir accesos y mantener sus datos confidenciales resguardados.

La biometría compite con otros métodos de identificación, como la NFC (*Near Field Communications*, usado actualmente en tarjetas de identificación), que es una tecnología de comunicación inalámbrica, y, por supuesto, también requiere una inversión significativa en cuanto a infraestructura y sensores.

La biometría garantiza uno de los niveles de autenticación más confiables en la actualidad, además de una reducción de costos y una mayor comodidad para el usuario.

Debido a las capacidades de hardware existentes entre los distintos dispositivos (cámaras, computadoras, *smartphones*, entre otros), se espera que la mayor parte del crecimiento en cuanto a autenticación, provenga de las tecnologías de reconocimiento facial y de voz. [3]

En este Proyecto Terminal se diseñará un sistema de reconocimiento facial que permita a distintos usuarios autenticarse por medio de sus Eigenfaces y características de Haar utilizando software libre en una Raspberry Pi 3.

2. Antecedentes

2.1. Proyectos de Integración o Terminales de la División de Ciencias Básicas e Ingeniería de la UAM

- Cerradura Electrónica con Reconocimiento Facial. [4].

En este proyecto se desarrolló un sistema de reconocimiento facial con una cerradura electrónica, un Raspberry Pi y técnicas de visión artificial. La diferencia con este proyecto es el modelo empleado en la tarjeta Raspberry, el lenguaje de programación donde no se profundizó en algún método; en esta propuesta se plantea indagar en el método de Eigenfaces a detalle.

- Sistema de Identificación de Rostros Humanos a través de Redes Neuronales. [5] Este proyecto está basado en el aprendizaje mediante redes neuronales, a diferencia del proyecto donde se implementa el uso de características y Eigenvectores para el aprendizaje.
- Procesamiento de Imágenes con Raspberry. [6] En este proyecto se explica el uso de una librería de visión por computadora para realizar el procesamiento de imágenes, a diferencia de este proyecto donde se emplea la librería para elaborar un sistema de reconocimiento facial.

2.2. Artículos

- *Image Processing Platform on Raspberry Pi for Face Recognition.* [7]

En este artículo se implementan dos métodos de reconocimiento (PCA y LDA) y se comparan en tiempo real usando una cámara web, la diferencia con este proyecto es el uso de distancias para que el sistema pueda reconocer a los usuarios.

- Sistema Biométrico Facial para el Acceso a un Área de Trabajo Implementado en Diferentes MiniPC's Comparando su Rendimiento. [8]

En este artículo se desarrolló un sistema biométrico que permite al usuario identificado abrir una cerradura con el método de Fisherfaces y la comparación de su rendimiento en distintas MiniPC's. La diferencia principal con este proyecto es el uso de Eigenfaces como método de reconocimiento facial.

- *Face Recognition Using Eigenface Approach.* [9]

Este artículo tiene como objetivo principal el uso de Eigenfaces para reconocimiento facial usando la distancia Euclideana y la distancia *Manhattan* para diferenciar un conjunto de entrenamiento de rostros. La diferencia con este proyecto es el uso de un sistema embebido que hará todo el procesamiento y características de Haar para describir el rostro.

3. Justificación

Es más frecuente el desarrollo de dispositivos biométricos que combinan diferentes formas de identificación, un sistema de reconocimiento facial permite la variación entre seguridad y usabilidad.

El algoritmo de Eigenfaces realiza una proyección lineal del espacio de imágenes a un espacio de características de menor dimensión. Esta reducción se realiza utilizando la técnica PCA (*Principal Component Analysis*) la cual toma aquella proyección lineal que maximiza la dispersión de todas las imágenes proyectadas, lo que permite una reconstrucción más precisa de los rostros. [10] Utilizando un sistema embebido, que integra todos los componentes de una computadora es ideal para la creciente demanda de dispositivos biométricos a un costo más accesible y con un espacio más reducido. Se pretende que el sistema sea más portable y que facilite tanto el aprendizaje como la implementación de técnicas de visión artificial.

4. Objetivos

4.1. Objetivo General

Diseñar e implementar un sistema de reconocimiento y autenticación facial usando el algoritmo de Eigenfaces y las características de Haar mediante el uso de un Raspberry Pi 3 y software libre.

4.2. Objetivos Específicos

- Diseñar e implementar el módulo de preprocesamiento del rostro capturado (acondicionamiento y normalización).
- Diseñar e implementar el módulo de extracción de características del rostro mediante el uso de características de Haar.
- Diseñar e implementar el módulo de entrenamiento de rostros mediante aprendizaje supervisado (eigenvectores).
- Diseñar e implementar el módulo de reconocimiento utilizando los resultados del entrenamiento descritos en el objetivo anterior y el uso del algoritmo reconocimiento de Eigenfaces.

5. Marco Teórico

En esta sección se presentan antecedentes relacionados a este proyecto tecnológico, y se da una descripción un tanto extensa en cuanto a la teoría que comprende el sistema.

5.1. Visión por Computadora

La visión por computadora es un campo de la Inteligencia Artificial que sugiere que las computadoras puedan extraer información necesaria a partir de imágenes. Las computadoras ven a través de cámaras conectadas a ellas, imágenes guardadas o animaciones (vídeo o alguna secuencia de imágenes), que son tratadas y procesadas para convertirlas en nuevas imágenes que permiten extraer sus características para su descripción e interpretación por la computadora (procesamiento de imágenes).



Figura 1: Diagrama general de la visión por computadora. La imagen es procesada, se extraen sus atributos y a la salida se da una descripción detallada de esta.

5.2. Características de Haar

Las características de Haar se han utilizado a menudo en los sistemas de reconocimiento para el seguimiento facial y problemas de clasificación, la razón principal de esto es el hecho de que las características de Haar no son invariables con respecto a la rotación, esto significa que cualquier objeto que gira y es sensible a los cambios de ángulo (como las manos) será difícil de resolver usando características de Haar.

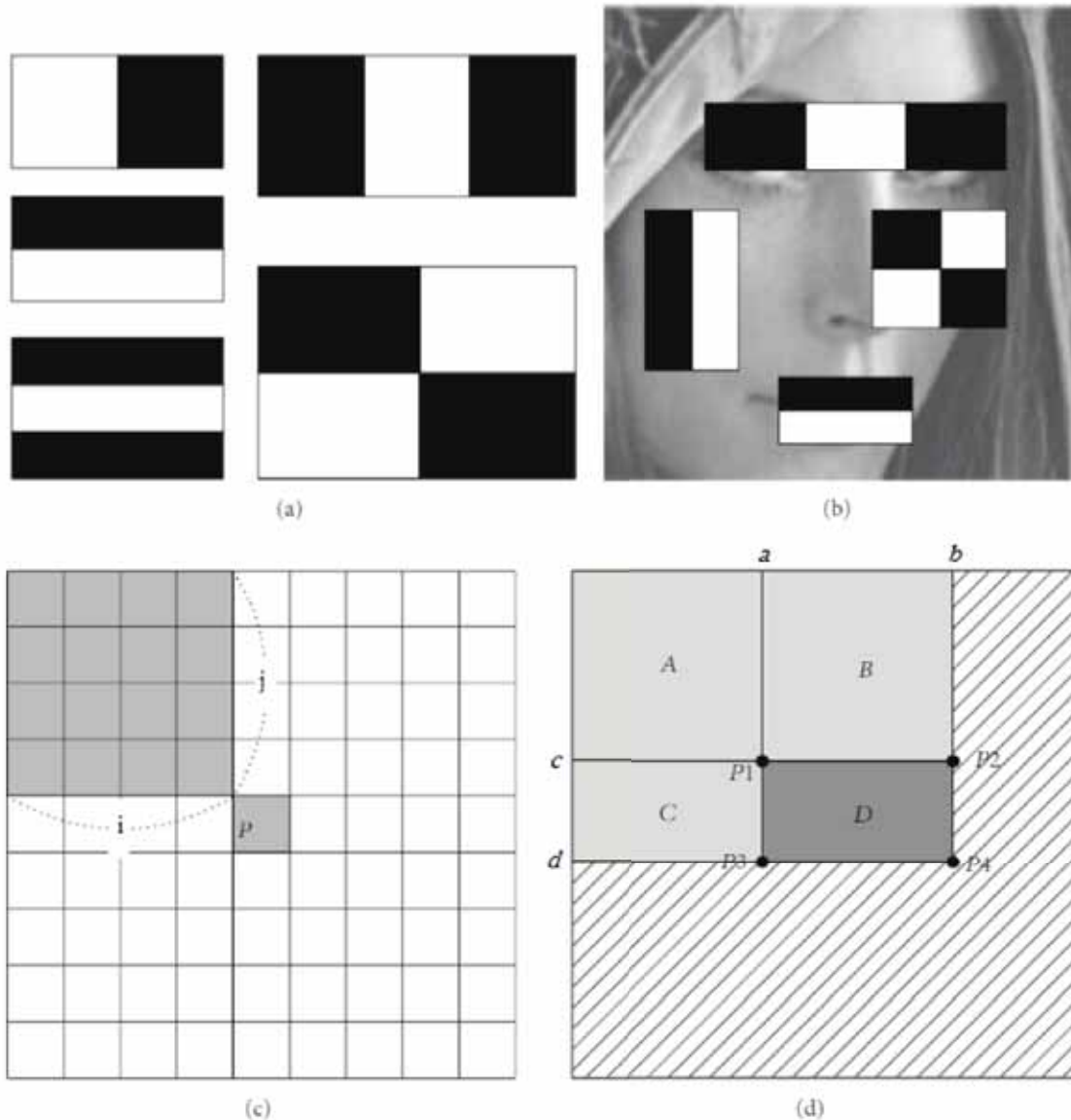


Figura 2: (a) Características de Haar, (b) Características de Haar aplicadas a una imagen, (c) Imagen integral del píxel $I(i,j)$, (d) Cómputo de la imagen integral para el rectángulo $D = I1 - I2 - I3 + I4$, donde $I1, I2, I3$ e $I4$ son las imágenes integrales en las coordenadas $(a,c), (a,d), (b,c), (b,d)$.

1

Las características que definen los rostros tienden a ser insensibles a las pequeñas variaciones de ángulo y las características de Haar se han utilizado para detectar rotaciones de la cabeza de hasta 15° desde la vertical (Jones y Viola, 2003). Cuando la gente está parada, su cabeza naturalmente se alinea verticalmente con respecto a la gravedad, esta sensibilidad rotacional no tiende a ser un problema significativo para los rostros.

¹Imagen disponible en: *System Architecture for Real-Time Face Detection on Analog Video Camera*, p-3. E.a. Kim M.

Las características de Haar consisten en una clase de características locales, como se observa en la Figura 2. Estas características se distinguen por el hecho de que son fáciles de calcular y con el uso de una imagen integral, muy eficiente para calcular.

Las imágenes integrales o tablas de área sumadas son una estructura de datos que contiene la suma de todos los píxeles arriba y a la izquierda del píxel actual. La complejidad del tiempo de el algoritmo es $2MN$ (donde M y N son el alto y ancho de la imagen) ya que cada el píxel en la imagen integral requiere dos operaciones de suma, como se observa en la Ecuación 1.

$$II(i, j) = II(i - 1, j) + II(i, j - 1) - II(i - 1, j - 1) + I(i, j) \quad (1)$$

donde $I(i,j)$ es la posición del píxel en la posición (i,j) , $II(i,j)$ es el valor de la imagen integral en la posición (i,j) .

Las imágenes integrales son importantes ya que permiten que la suma de un área rectangular de píxeles de cualquier tamaño se calcule con sólo 4 búsquedas en la estructura de datos de la imagen integral

$$\sum_{i=a}^b, \sum_{j=c}^d I(i, j) = II(a, c) - II(a, d) - II(b, c) + II(b, d) \quad (2)$$

donde $I(i,j)$ es el valor del píxel en la posición (i,j) , $II(i,j)$ es el valor de la imagen integral en la posición (i,j) , (a,c) es la coordenada del píxel superior izquierdo y (b,d) es la coordenada del píxel inferior derecho de la región rectangular que se está sumando. Las características de Haar consisten en una clase de características locales calculadas restando la suma de una subregión de la característica de la suma de la región restante de esta. Generalmente no se encuentran características de rostros a lo largo de la imagen y la primera conclusión es que no hay un rostro, en caso contrario, se hace el cálculo de la imagen integral y se concluye que existe un rostro. [11]

5.3. Análisis de Componentes Principales (PCA)

Se comienza definiendo el término PCA (Análisis de Componentes Principales) por sus siglas en inglés. ¿Qué es? Es una forma de identificar patrones en los datos, y expresar esos datos de tal forma que se muestren sus diferencias y semejanzas.

Los componentes (píxeles) de las imágenes de los rostros están relacionadas entre sí. El Análisis de Componentes Principales se basa en las propiedades estadísticas de las imágenes y es un método que permite reducir el número de dimensiones necesarias para representar un conjunto de vectores (en este trabajo, rostros).

Ahora, se considera a un vector aleatorio \mathbf{X} de dimensión \mathbf{n} , con su correspondiente media μ

$$\mu = E\{X\} \quad (3)$$

La matriz de covarianza se define como

$$\Sigma = E\{(X - \mu)(X - \mu)'\} = \begin{vmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_{nn} \end{vmatrix} \quad (4)$$

El análisis de componentes principales consiste en hallar los vectores propios de esta matriz y expresar \mathbf{X} en función de estos vectores. De esta manera, el vector aleatorio \mathbf{X} se puede representar sin error mediante una combinación lineal de vectores de la forma

$$X = \sum_{i=1}^n y_i A_i \quad (5)$$

donde los vectores A_i son los vectores propios de la matriz de covarianza. La cualidad más importante de esta representación es que si se quiere representar X con sólo \mathbf{m} (menor que \mathbf{n}) componentes la mejor elección posible en términos del error que se comete son los \mathbf{m} vectores propios de la matriz de covarianza con mayores valores propios asociados.

5.3.1. PCA para la representación de Eigenfaces

El Análisis de Componentes Principales se reduce a encontrar los vectores propios de la matriz de covarianza Σ_X del vector al que se le aplica el PCA.

Se utiliza toda esta teoría para resolver el problema particular de reconocimiento facial, donde el objetivo es caracterizar un conjunto de caras (que serán las del módulo de entrenamiento) en un espacio de menor dimensión.

Se considera al conjunto de todas las imágenes posibles cuyas dimensiones son w píxeles de ancho por h píxeles de alto como realizaciones del vector aleatorio X de dimensión $w \cdot h$, con su correspondiente vector media. Este vector se llamará vector rostro, y se desea aplicarle el PCA.

La media de este vector rostro no la podemos obtener al no conocer la función de densidad de probabilidad, pero se puede estimar a partir las imágenes que se tienen

$$\mu \approx \frac{1}{N} \sum_{i=1}^N X_i \quad (6)$$

La matriz de covariancia tampoco se conoce pero se también se puede estimar

$$\Sigma_X \approx \frac{1}{N} \sum_{i=1}^N (X_i - \mu)(X_i - \mu)' = \frac{1}{N} M_X M_X' \quad (7)$$

Ahora sólo queda el cálculo de los vectores propios de la matriz de covariancia; existen varios algoritmos automáticos fácilmente implementables. Pero la implementación de este algoritmo lleva a un problema; la cantidad memoria necesaria para almacenar la matriz es demasiado grande. Si los vectores rostro del entrenamiento son de dimensión $w \times h$, las dimensiones de la matriz de covariancia estimada Σ_X serán de $w \times h$ de ancho y $w \times h$ de alto. Por ejemplo, si se utilizan imágenes de entrenamiento con unas dimensiones contenidas de $w=128$ y $h=128$, el número de elementos de la matriz de covariancia sería de 128^4 , o sea más de 268 millones. La solución de este problema se puede ver a continuación.

5.4. Descomposición en Valores Singulares (SVD)

Esta operación descompone una matriz en un producto de tres matrices, aplicada a la matriz M_X

$$M_X = U A^{1/2} V^T \quad (8)$$

donde si M_X es una matriz de $w \times h$ filas y columnas con $w \times h$ mayor que N (lo que siempre será nuestro caso) la matriz U es de las mismas dimensiones y las matrices V y $A^{1/2}$ son cuadradas de dimensiones N por N . Además, la matriz $A^{1/2}$ es diagonal.

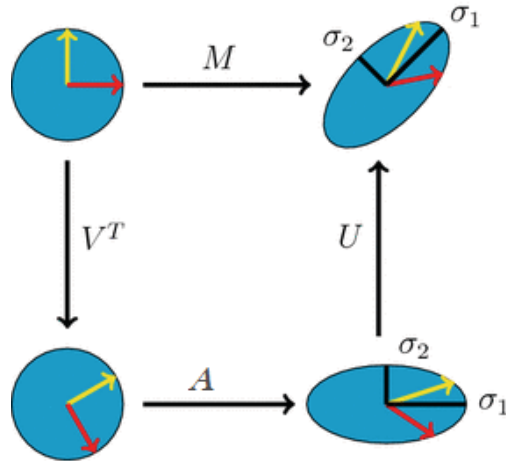


Figura 3: Ejemplo gráfico de una Descomposición en Valores Singulares.

2

La propiedad que resulta útil de la Descomposición por Valores Singulares es que las N columnas de la matriz U son vectores propios de la matriz XX^T , y los elementos de la matriz $A^{1/2}$ son las raíces cuadradas de los valores propios correspondientes a estos vectores propios. La estimación de la matriz Σ_X vemos que esta no era exactamente XX^T , sino que además se dividía este producto por N . Pero esto no importa ya que los vectores y valores propios de una matriz y los de la misma matriz dividida por un escalar son iguales, salvo por el hecho de que los valores propios también quedan divididos por el mismo número.

Los N vectores propios que genera la SVD son capaces de caracterizar a los vectores correspondientes al conjunto de imágenes de rostros de entrenamiento, y el resto de vectores propios sólo servirían para abarcar el resto del espacio original de X (el espacio de las imágenes de w por h píxeles), pero no servirían para representar los rostros. De entre estos vectores propios, el número de ellos que contienen información útil (valor propio mayor de cero) es igual al número de vectores linealmente independientes del conjunto de entrenamiento menos uno.

De esta manera se obtienen los vectores propios que definen el nuevo espacio, el espacio de los rostros. Los vectores resultantes tienen apariencia de rostros, y por ello en un inicio fueron llamadas *Eigenpictures* (imágenes propias) y posteriormente Eigenfaces (caras propias). En la Figura 4 se muestran las 5 primeras Eigenfaces generadas con un conjunto de 57 imágenes de entrenamiento.

²Imagen disponible en: *Metamorphic Detection Using Singular Value Decomposition*, p-14. Kumar R.



Figura 4: Primeras cinco Eigenfaces generadas a partir de 57 imágenes de rostros normalizados.

3

5.5. Proceso de Generación de Eigenfaces

En este punto, se cuenta con un conjunto de N imágenes de rostros X_i (que utilizan como conjunto de entrenamiento) que se busca representar en un espacio de menor dimensión. Para esto, se han considerado imágenes como realizaciones del proceso aleatorio vectorial imágenes de rostros y se ha aplicado el PCA a este vector. Como resultado, se obtienen N vectores propios (Eigenfaces) ortonormales y con $N-1$ de ellos se pueden generar sin error las N caras del conjunto de entrenamiento, como se puede ver en la Figura 5

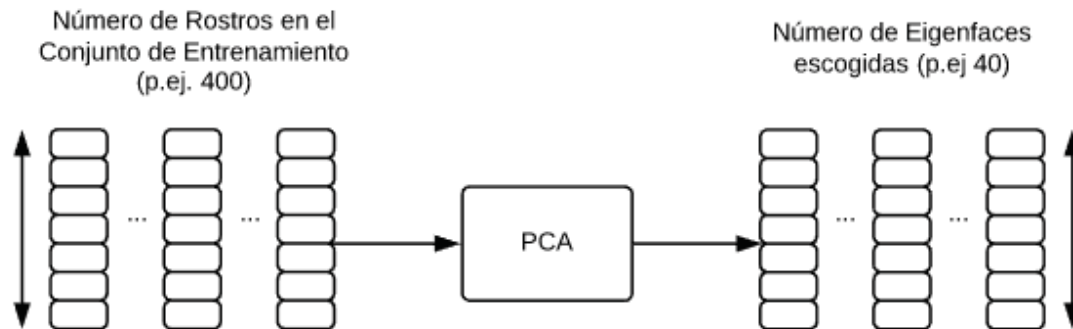


Figura 5: Proceso de generación de Eigenfaces.

³Imagen disponible en: Representación de Caras mediante Eigenfaces. Lorente L. [12]

Para hallar las coordenadas Y_i de una cara X_i sobre este nuevo espacio se tiene que proyectar X_i sobre las Eigenfaces. Para hallar la proyección de X_i sobre cada Eigenface basta con calcular el producto escalar (las Eigenfaces son ortonormales). Por lo tanto, si las columnas de la matriz \mathbf{A} son las Eigenfaces, se pueden hallar las nuevas coordenadas en el espacio de los rostros

$$Y_i = P^T X_i \quad (9)$$

Al haber $N-1$ Eigenfaces la dimensión de los vectores Y_i es $N-1$ (número de imágenes de entrenamiento). Esto supone una reducción de dimensión frente al espacio de las imágenes ($w \cdot h$) y es totalmente normal, ya que para representar a N vectores a los que se ha restado su media basta con $N-1$ vectores. La reducción de dimensión de la representación mediante Eigenfaces consiste en utilizar sólo las Eigenfaces con mayor valor propio asociado.

Cabe recordar que el valor propio asociado a una Eigenface es igual a la varianza de la proyección de los rostros sobre esa Eigenface. De esta manera se puede reducir notablemente el número de dimensiones de la representación sin perder calidad en la representación de los rostros. En este aspecto, las Eigenfaces consiguen la máxima eficiencia que se puede conseguir ya que las primeras Eigenfaces (las de mayor valor propio asociado) consiguen capturar la información más importante de los rostros, la que explica la máxima varianza, mientras que las últimas apenas aportan información pues sus valores propios son muy bajos.

La mejor representación posible de un conjunto de N imágenes de rostros utilizando sólo M dimensiones (con $M < N$) se consigue proyectándolos sobre las M Eigenfaces con mayor valor propio. Con un número muy bajo de Eigenfaces se consigue una representación con una notable calidad, lo que demuestra que las primeras Eigenfaces consiguen capturar la mayor parte de la información de los rostros.

Para expresar la calidad numéricamente se debe hallar una medida del error cometido al representar un rostro. Si la representación de un rostro \mathbf{X} con M Eigenfaces es

$$\tilde{X} \approx \mu + \sum_{i=1}^M y_i A_i \quad (10)$$

donde μ es el vector media, A_i son las Eigenfaces ordenadas de mayor a menor valor propio y y_i la proyección de \mathbf{X} sobre A_i , y, el error de representación es definido como

$$e = \left(\frac{\|X - \tilde{X}\|}{\|X\|} \right)^2 \quad (11)$$

Así, el error está normalizado y su rango de valores va de cero a uno. Sin embargo, debido a que la media es una buena estimación de todas las caras en la práctica los valores siempre son mucho más reducidos. [12]

5.6. Sistema de Reconocimiento

En esta sección se hablará acerca de las partes que conforman el sistema de reconocimiento facial, para posteriormente ver su desarrollo.

5.6.1. Normalización o Preprocesamiento de la Imagen

El potencial de las Eigenfaces reside en su gran capacidad para caracterizar con un mínimo error una imagen de un rostro. Esta capacidad siempre es muy alta con las imágenes con las que se han generado las Eigenfaces, pero con otras imágenes de rostros (pe. las que se tendrán que reconocer) depende bastante del procesado que se le aplique a la imagen antes de proyectarla sobre las Eigenfaces.

Esta fase intenta compensar las diferencias de otros rostros en los siguientes aspectos:

- Normalización del alto y ancho del rostro.
- Rotación de la imagen del rostro de manera tal que los dos ojos queden horizontales.
- Recorte de la zona del rostro; se elimina información adicional, es decir, el fondo de la imagen y el cabello.
- La imagen del rostro se convierte a escala de grises. Como la representación es de 8 bits, el valor mínimo de los píxeles pasa a cero y el máximo a 255.

5.6.2. Extracción de Características

En esta fase se realiza el Análisis de Componente Principal con el cual se extraen las Eigenfaces a consideración de lo siguiente:

- Se hace el cálculo de la media del rostro y se resta de las imágenes normalizadas.
- Se genera la matriz M_X , donde las columnas son imágenes de rostros del entrenamiento normalizados.
- Se aplica Descomposición en Valores Singulares a la matriz M_X generando tres matrices, una de ellas contiene las Eigenfaces (valores propios) y otra las raíces cuadradas de los valores propios.
- Se guardan las Eigenfaces y las proyecciones de las imágenes de rostros del entrenamiento sobre ellas.

- Finalmente, se halla la proyección de las imágenes de rostros del entrenamiento sobre las Eigenfaces y se realiza el producto escalar de la imagen sobre cada una de las Eigenfaces.

5.6.3. Fase de Reconocimiento

Ya obtenido el vector de características, el objetivo de esta fase es determinar que imagen del conjunto de entrenamiento es más parecida a la imagen prueba a partir de sus proyecciones.

Finalmente, se compara el vector de prueba y_{Pr} formado por las proyecciones de la imagen de prueba sobre las Eigenfaces con cada uno de los vectores del entrenamiento y_{En} y se utiliza la distancia euclidiana como criterio de decisión, es decir $|y_{Pr} - y_{En}|$

6. Desarrollo

6.1. Recursos

En esta sección se listan los recursos utilizados durante el desarrollo del proyecto, con una breve explicación de cada uno de ellos.

6.1.1. Hardware

Raspberry Pi 3

Es una mini-computadora del tamaño de una tarjeta de crédito es capaz de realizar muchas labores que hace una computadora de escritorio, como hojas de cálculo, tratamiento de textos, reproducción de vídeo de alta definición y en este caso, visión por computadora. Se pueden ejecutar varias versiones de sistemas basados en Linux e incluso Windows 10. Algunas de sus características son las siguientes:

- Broadcom BCM2837.
- ARM Cortex A53 con procesador de 64 bits con cuatro núcleos a 1.2GHz.
- BCM43438 WLAN y Bluetooth Low Energy (BLE).
- Memoria RAM de 1GB.
- 40 pines extendidos GPIO.
- 4 puertos USB 2.0.
- 1 puerto Full HDMI.
- Ranura CSI para módulo de cámara.
- Ranura DSI para módulo LCD.
- Ranura para tarjetas MicroSD (*push-pull*).
- Núcleo de gráficos VideoCore IV 3D.
- Dimensiones de la placa de 8.5 por 5.3 cm.

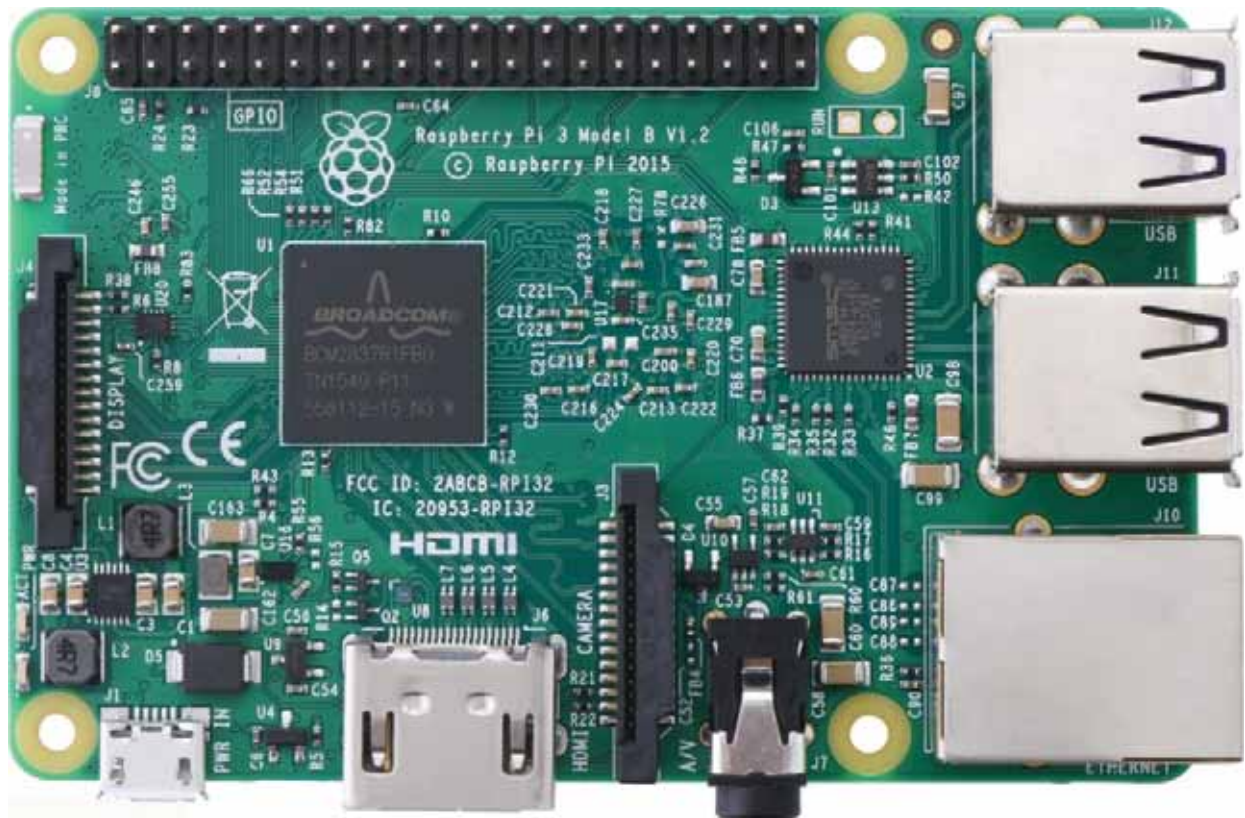


Figura 6: Placa de desarrollo Raspberry Pi 3 Modelo B.

Módulo de cámara *Raspberry Pi Camera v1*

Diseñado y fabricado por *Raspberry Pi Foundation* en el Reino Unido. El módulo se conecta a la Raspberry Pi 3, a través de un cable de cinta de 15 pines, a la interfaz serie de la cámara MIPI (CSI) de 15 pines, que fue diseñada especialmente para la interfaz con cámaras. El bus CSI es capaz de velocidades de datos extremadamente altas, y lleva exclusivamente datos de píxeles al procesador BCM43438. Algunas de sus características son las siguientes:

- Resolución 5 Megapíxeles
- Video 1080p30, 720p60 and 640 x 480p60/90
- Sensor OmniVision OV5647
- Resolución del sensor 2592 × 1944 píxeles



Figura 7: Raspberry Pi Camera V1.

6.1.2. Software

Raspbian Stretch versión 2017-09-07⁴

Raspbian es el sistema operativo oficial de la Fundación, viene preinstalado con gran cantidad de software para educación, programación y uso general; por ejemplo Python, Scratch, Sonic Pi, Java, Mathematica y más.

Raspbian es un sistema operativo gratuito basado en Debian Jessie (Debian 8.0) optimizado para el hardware Raspberry Pi, este proporciona más que un sistema operativo puro: viene con más de 35,000 paquetes, software precompilado incluido en un formato agradable para una fácil instalación en su Raspberry Pi, además cualquier software de código abierto puede ser recompilado en la propia Raspberry Pi 3 para una arquitectura armhf que pueda ser utilizado en el propio dispositivo en caso de que el desarrollador no proporcione una versión ya compilada para esta arquitectura. Esta distribución, como la mayoría, contiene repositorios donde el usuario puede descargar multitud de programas como si se tratase de una distribución de GNU/Linux para equipos de escritorio.



Figura 8: Logotipo oficial de Raspbian.

OpenCV 3⁵

OpenCV (*Open Source Computer Vision Library*) es una biblioteca de software de visión abierta y software de aprendizaje automático. OpenCV fue construido para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto con licencia de BSD, OpenCV facilita a las empresas utilizar y modificar el código.



Figura 9: Logotipo oficial de OpenCV.

La biblioteca cuenta con más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos de visión artificial y de aprendizaje automático tanto clásicos como avanzados. Estos algoritmos se pueden usar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en vídeos, rastrear movimientos de la cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D desde cámaras estéreo, unir imágenes para producir una alta resolución imagen de una escena completa, entre otras más funcionalidades.

El 27 de febrero del año en curso se presentó la versión 3.4.1 con un módulo DNN (*Deep Neural Networks*) optimizado, muchas otras mejoras y corrección de errores.

⁴Detalles de la instalación en el Apéndice A

⁵Detalles de la instalación en el Apéndice B

OpenCV cuenta con interfaces C ++, Python, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS. [13]

Python 3.3.0

6.2. Diseño del Sistema

El desarrollo de este proyecto está basado en 5 etapas fundamentales, algunas de ellas se vieron a detalle en la Sección 5.6.1. En la Figura 10 se aprecia el algoritmo de manera modular.

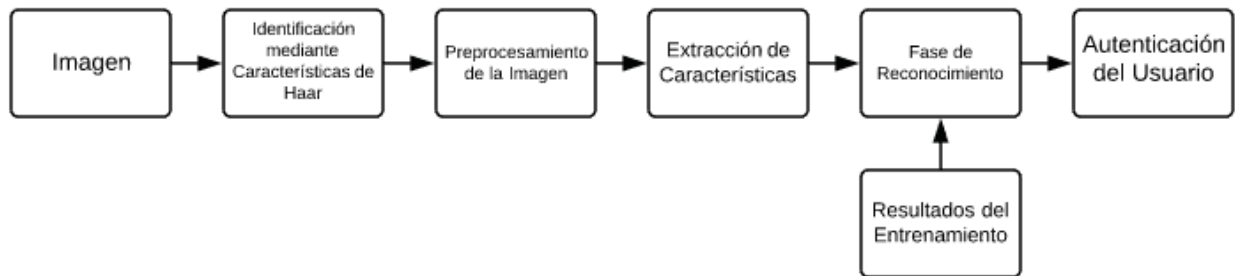


Figura 10: Sistema de reconocimiento modular en Raspberry Pi 3.

7. Resultados

7.1. Identificación mediante características de Haar

En este módulo, *"rostros_positivos.py"*, se hace uso de la adquisición de las imágenes positivas, las imágenes negativas se encuentran en el directorio *negativo* y se tomaron de la base de datos de rostros de AT&T ⁶. Cada imagen está en formato *pnm*, que puede ser visualizado en editores como *GIMP*, el tamaño de cada imagen es de 92x112 píxeles, con niveles de gris de 256 por píxel, éstas están organizadas en 40 directorios con 10 imágenes cada uno, siendo un total de 400 imágenes de rostros con variaciones de luz, expresiones faciales y detalles como lentes y vello facial.

Siguiendo con las imágenes del conjunto positivo, el nombre de la imagen se guarda con el prefijo *"img_positiva_"* seguida de su ID con extensión *pnm* (*Portable Gray Map*), en caso de que no exista el directorio se crea uno nuevo llamado *positivo*.

El sistema despliega una ventana donde se puede ver en directo la imagen de la cámara (conectada al puerto *CSI* del Raspberry Pi 3), esto con el fin de mejorar resultados al momento de capturar el rostro. El usuario presiona la letra *"f"* seguida de *Enter* para tomar una captura y se convierte a escala de grises, después, se identifican los rostros mediante las características de Haar para un único rostro con la menor distancia calculada, es importante ya que pueden detectarse varios rostros en el fondo (Figura 11).

```
(cv) pi@raspberrypi:~/Desktop/PTMario $ sudo python rostros_positivos.py
Capturando imagenes positivas para el conjunto de entrenamiento.
Presione f seguido de Enter para tomar una foto.
Presione Control+C para salir.
f
Capturando imagen
Se ha encontrado un rostro, se guardara en el conjunto de entrenamiento. ./entrenamiento/positivo/img_positiva_011.pnm
f
Capturando imagen
Se ha encontrado un rostro, se guardara en el conjunto de entrenamiento. ./entrenamiento/positivo/img_positiva_012.pnm
f
Capturando imagen
Se ha encontrado un rostro, se guardara en el conjunto de entrenamiento. ./entrenamiento/positivo/img_positiva_013.pnm
f
Capturando imagen
Se ha encontrado un rostro, se guardara en el conjunto de entrenamiento. ./entrenamiento/positivo/img_positiva_014.pnm
f
Capturando imagen
No se detecto ningun rostro, revise muestra.jpg para observar la imagen resultante.
```

Figura 11: Captura de imágenes para el conjunto positivo de entrenamiento.

⁶Disponible en: <http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>

7.2. Preprocesamiento de la imagen

En caso de que se reconozca un rostro, se elimina la información adicional al rostro, se redimensiona la imagen y se ecualiza el histograma para mejorar la calidad de la imagen.

Se despliega un letrero diciendo que se ha encontrado un rostro, y se guardará en el conjunto de entrenamiento del directorio "positivo".

En caso contrario se desplegará un letrero diciendo que no se encontró ningún rostro y se solicita revisar la imagen *muestra.jpg* para observar el resultado, esta imagen se encuentra en el directorio "positivo".

Y se lleva a cabo el proceso de acondicionamiento y normalización geométrica de las imágenes recibidas del módulo anterior como se puede observar en la Figura 12.



Figura 12: (a) Rostros capturados para el conjunto positivo de entrenamiento. (b) Rostros capturados para el conjunto negativo de entrenamiento.

7.3. Extracción de características

Este módulo consiste en extraer información asociada al rostro donde se seleccionan regiones como los ojos, nariz, boca, entre otros y se clasifican mediante deformaciones del rostro caracterizadas por cambios de forma y textura, utilizando las imágenes acondicionadas y normalizadas del módulo anterior.

El sistema busca todas las imágenes en los directorios del conjunto de entrenamiento "negativo" y "positivo" y da una cuenta de ellas.

Se prepara la imagen en escala de grises y comienza la fase de entrenamiento, donde se crean las Eigenfaces de los conjuntos negativos y positivos (ver Figura 13) y se guardan los resultados en el archivo "resultEntrenamiento.xml".

```
(cv) pi@raspberrypi:~/Desktop/PTMario $ sudo python entrenamiento.py
Leyendo imagenes del conjunto de entrenamiento, espere.
Se leyeron 22 imagenes positivas y 400 imagenes negativas.
Se esta entrenando el modelo, esta operacion puede tardar un poco.
Entrenamiento exitoso, datos guardados en resultEntrenamiento.xml
```

Figura 13: Ventana de entrenamiento del sistema.

Para observar los resultados del aprendizaje supervisado, se guardan tres imágenes, donde la primera corresponde a la media de todas las imágenes tanto positivas como negativas, la segunda y tercera, mediante Eigenvectores se obtienen las Eigenfaces del conjunto positivo de entrenamiento y las Eigenfaces del conjunto negativo, todas en formato *png* con dimensiones de 92x112 como se puede ver en la Figura 14.

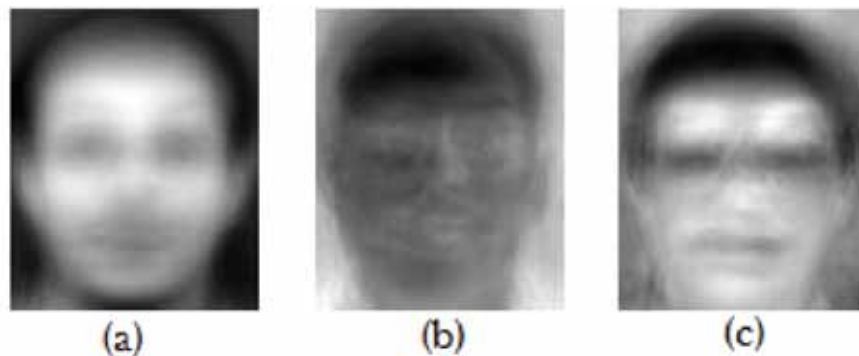


Figura 14: (a) Eigenface del promedio de todos los rostros capturados. (b) Eigenface de rostros del conjunto positivo. (c) Eigenface de rostros del conjunto negativo.

7.4. Fase de reconocimiento

En este módulo, una vez obtenido el vector de características, se leen los resultados del conjunto de entrenamiento, se inicia la cámara y al igual que el módulo de identificación mediante características de Haar se inicia una ventana tipo espejo para mejorar los resultados al momento de la obtención de la imagen.

El usuario presiona la letra "f" seguida de *Enter* para tomar una captura y se convierte a escala de grises, después, se identifican los rostros mediante las características de Haar; si no se detecta un rostro despliega un mensaje y se pide revisar la imagen resultante, si sí se detecta un rostro, es mapeado, se lleva a cabo un proceso de normalización sobre la imagen y se compara con el archivo generado por el módulo de entrenamiento .

7.5. Autenticación del usuario

Del módulo anterior, se toma una decisión de acuerdo al umbral definido en archivo "*configuracion.py*" donde se tiene lo siguiente:

Detectado un rostro positivo/negativo y el cálculo de su veracidad. Si es positivo el rostro y la veracidad es menor que 3000 entonces el rostro habrá sido reconocido, en caso de que el rostro sea positivo pero la veracidad sea mayor que 3000 entonces el rostro no habrá sido reconocido.

Si se detecta un rostro negativo (cualquier otro, independiente del conjunto de rostros negativos), se desplegará un letrero indicando su veracidad y que el rostro no fue reconocido, ver Figura 16

Para probar el alcance y mostrar una variedad de resultados, las imágenes (40 imágenes por entrenamiento) del conjunto positivo de entrenamiento fueron tomadas con una exposición de luz baja, media y alta, posteriormente procesadas por el conjunto de entrenamiento (en ese orden) y se observa que a mayor exposición de luz los resultados son mejores al momento de autenticar al usuario.

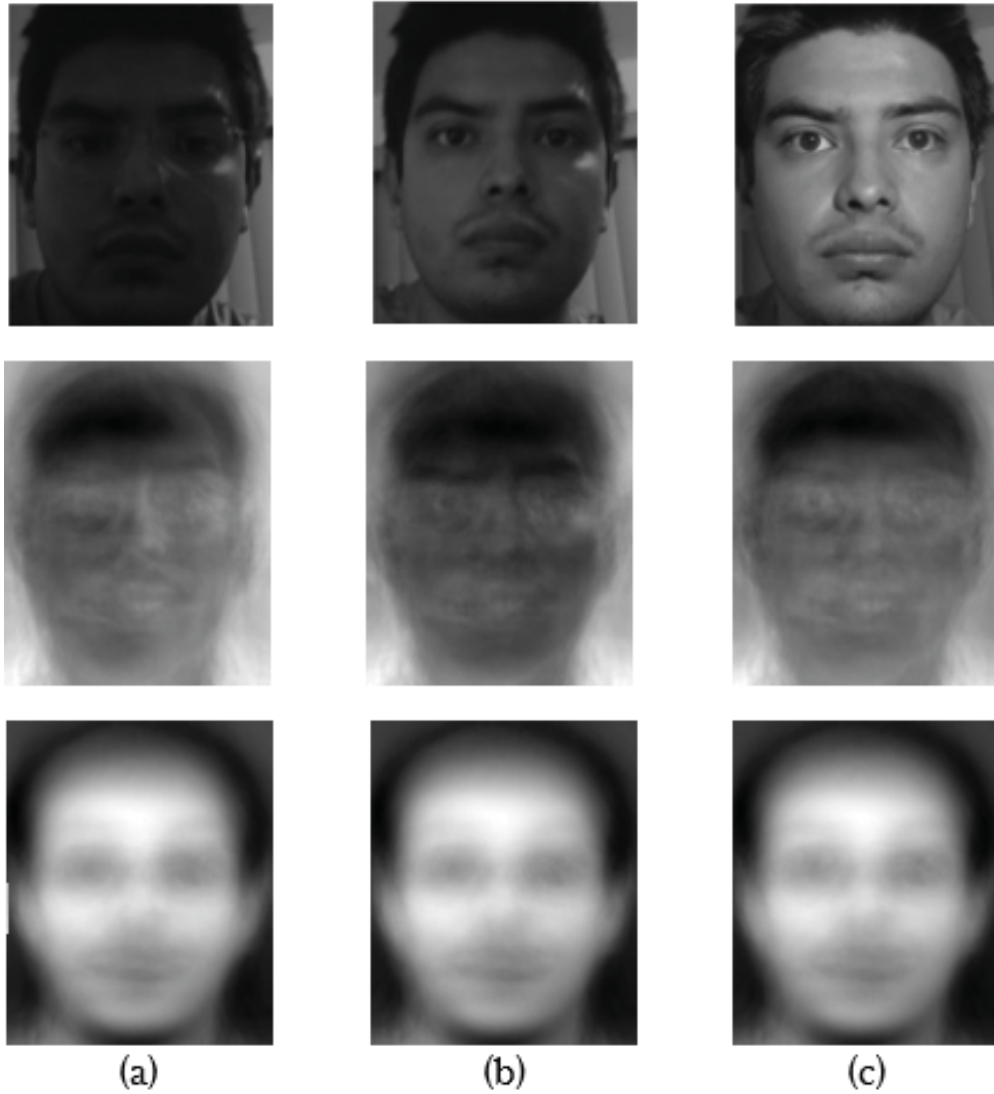


Figura 15: Rostro capturado, Eigenface positiva y media de: (a) Capturas con poca exposición de luz. (b) Capturas con exposición de luz media. (c) Capturas con alta exposición de luz.

```
(cv) pi@raspberrypi:~/Desktop/PTMario $ sudo python reconocimiento.py
Cargando datos del entrenamiento, espere.
Datos del entrenamiento cargados.
Presione f seguido de Enter para tomar una foto.
Presione Control+C para salir.
f
Detectado un rostro negativo con veracidad de 3602.36034744
Rostro no reconocido, intente de nuevo.
f
Detectado un rostro positivo con veracidad de 3897.15316423
Rostro no reconocido, intente de nuevo.
f
Detectado un rostro positivo con veracidad de 2554.09297586
Rostro reconocido.
```

(a)

```
(cv) pi@raspberrypi:~/Desktop/PTMario $ sudo python reconocimiento.py
Cargando datos del entrenamiento, espere.
Datos del entrenamiento cargados.
Presione f seguido de Enter para tomar una foto.
Presione Control+C para salir.
f
Detectado un rostro positivo con veracidad de 1530.21198947
Rostro reconocido.
f
Detectado un rostro positivo con veracidad de 1665.35051973
Rostro reconocido.
f
Detectado un rostro positivo con veracidad de 1472.27060551
Rostro reconocido.
```

(b)

```
(cv) pi@raspberrypi:~/Desktop/PTMario $ sudo python reconocimiento.py
Cargando datos del entrenamiento, espere.
Datos del entrenamiento cargados.
Presione f seguido de Enter para tomar una foto.
Presione Control+C para salir.
f
Detectado un rostro positivo con veracidad de 1015.2970462
Rostro reconocido.
f
Detectado un rostro positivo con veracidad de 1264.15811586
Rostro reconocido.
f
Detectado un rostro positivo con veracidad de 1522.29286995
Rostro reconocido.
```

(c)

Figura 16: (a) Resultados con poca exposición de luz. (b) Resultados con exposición de luz media. (c) Resultados con alta exposición de luz.

8. Conclusiones

El reconocimiento facial ha ido ganando presencia a lo largo de los años, siendo su uso principal en sistemas de seguridad y en teléfonos inteligentes.

A lo largo del desarrollo del proyecto, se observa una gran variación de resultados debido a diferentes exposiciones de luz, distancias, ángulo de posicionamiento, detalles faciales, etcétera. Por lo que dar resultados precisos es difícil.

Si bien existen algoritmos mucho más complejos y más difíciles de quebrantar, el algoritmo de Eigenfaces se presenta como un paso introductorio al entendimiento de la visión por computadora y el reconocimiento facial.

Referencias

- [1] Unam - facultad de ingeniería, biometría informática. [En línea]. Disponible: <http://redyseguridad.fi-p.unam.mx/proyectos/biometria/fundamentos/antecedentes.html>
- [2] Así ha avanzado la tecnología de reconocimiento facial. [En línea]. Disponible: <https://blogthinkbig.com/asi-ha-avanzado-la-tecnologia-de-reconocimiento-facial>
- [3] Biometría: acceso seguro a la movilidad. [En línea]. Disponible: <https://www.forbes.com.mx/biometria-acceso-seguro-la-movilidad/>
- [4] Hipólito Morales, Christian, “Cerradura Electrónica con Reconocimiento Facial.” Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2014.
- [5] Alducin Castillo, Javier, “Sistema de Identificación de Rostros Humanos a través de Redes Neuronales.” Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2008.
- [6] Vargas Chávez, J. E., “Procesamiento de Imágenes con Raspberry.” Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Iztapalapa, México, 2014.
- [7] T. R. Medak y S. K. Ranga, “Image processing platform on raspberry pi for face recognition,” *Global Journal of Advanced Engineering Technologies*, vol. 3, núm. 4, pp. 441–444, Jan 2014.
- [8] e. a. Miguel, “Sistema biométrico facial para el acceso a un área de trabajo implementado en diferentes minipc’s comparando su rendimiento,” *Congreso Internacional de Ingeniería Electrónica. Memoria Electro*, vol. 38, núm. 1, pp. 229–234, Ago 2016.
- [9] S. Marijeta y J. Dubravka, “Face recognition using eigenface approach,” *Serbian Journal of Electrical Engineering*, vol. 9, núm. 1, pp. 121–130, Feb 2012.
- [10] Reconocimiento de caras: Eigenfaces y fisherfaces. [En línea]. Disponible: https://eva.fing.edu.uy/file.php/514/ARCHIVO/2010/TrabajosFinales2010/informe_final_ottado.pdf
- [11] C. Messom y A. Barczak, “Stream processing for fast and efficient rotated haar-like features using rotated integral images,” *International Journal of Intelligent Systems Technologies and Applications (IJISTA)*, vol. 7, núm. 1, pp. 40–57, May 2009. [En línea]. Disponible: <https://pdfs.semanticscholar.org/6c95/6a107ba358d873a169a8678080145dcc3d5b.pdf>
- [12] Lorente Giménez, L., “Representación de Caras Mediante Eigenfaces.” Projectista del Departamento de Teoría de Señal y Comunicaciones, Grupo de Procesado de Imagen, Universidad Politécnica de Catalunya, España, 1998.
- [13] Opencv, about. [En línea]. Disponible: <https://opencv.org/about.html>

A. Instalación de Raspbian

Descarga de Raspbian.

El primer paso es descargar el Sistema Operativo, en este proyecto se usó la versión 2017-09-07. La imagen de Raspbian con escritorio viene contenida en un archivo ZIP que cuenta con más de 4GB de tamaño, lo que significa que estos archivos usan funciones que no son compatibles con herramientas de descompresión anteriores en algunas plataformas.

Si la descompresión del archivo presenta problemas, la página oficial de Raspberry sugiere intentar utilizar *7Zip* (Windows) o *Unarchiver* (Macintosh). Ambos son gratuitos y se han probado para descomprimir la imagen correctamente.

La versión oficial más reciente de Raspbian puede descargarse del siguiente enlace: <https://www.howtoforge.com/tutorial/howto-install-raspbian-on-raspberry-pi/>

Descarga e Instalación de la Imagen en un Grabador de Archivos de Imagen.

Para grabar el Sistema Operativo en una tarjeta micro SD es necesario un Grabador de Archivos de Imagen, en este proyecto se usó *Win32 Disk Imager*, un programa de código abierto desarrollado por los usuarios de la plataforma *SourceForge* **gruemaster** y **tuxinator2009**, y la versión oficial más reciente puede descargarse del siguiente enlace: <https://sourceforge.net/projects/win32diskimager/>

Una vez descargado *Win32 Disk Imager* se debe insertar la tarjeta micro SD en la laptop/pc y ejecutar el programa. Ya abierto, seleccionar el archivo de imagen Raspbian descargado. Seleccionar el dispositivo correcto, que es el disco que representa la tarjeta micro SD. Si la unidad (o dispositivo) seleccionada es diferente de la tarjeta SD, la otra unidad seleccionada se dañará.

Después de eso, hacer clic en el botón "Write" en la parte inferior.

Una vez completa la escritura, expulsar la tarjeta micro SD, insertarla en el Raspberry Pi 3 y encenderlo, al hacer esto, el sistema iniciará.

Configuración del Sistema Operativo Raspbian.

Pueden presentarse situaciones donde se pidan credenciales como el nombre del usuario y la contraseña. Raspbian viene con un nombre y contraseña por defecto, que son:

- *login* : *pi*
- *password* : *raspberrypi*

Cuando el sistema de inicio por primera vez, aparecerá una ventana de configuración llamada "*RaspberryPiSoftwareConfigurationTool(raspi-config)*", que debe ser como la de la Figura 17

Si se llega a cerrar la ventana de configuración, se puede acceder mediante el siguiente comando:

```
$ sudo raspi-config
```

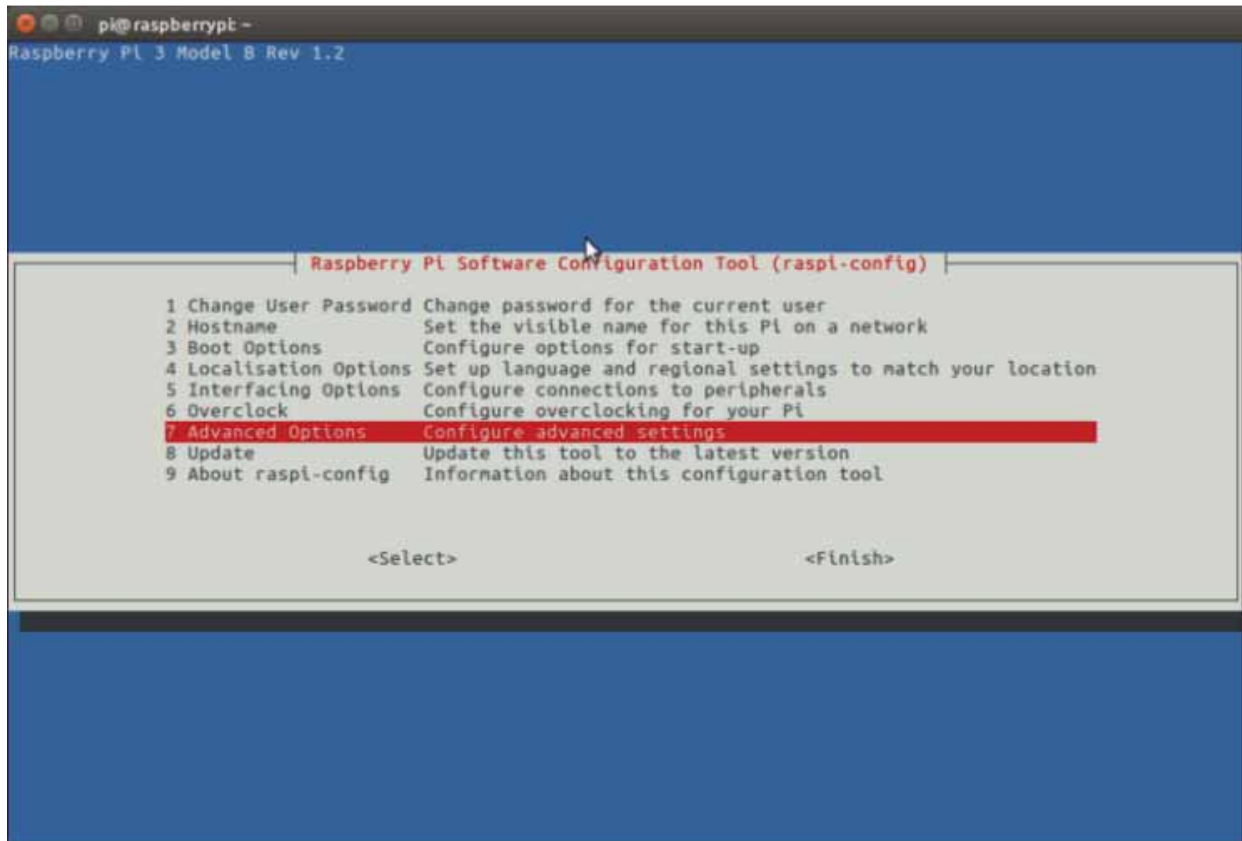


Figura 17: Ventana de configuración de Raspbian.

La primera cosa que se debe hacer en la ventana de configuración es seleccionar la opción de "AdvancedOptions", seguido de "ExpandFilesystem" (Figura 18). Se hace esto para usar todo el espacio de la tarjeta micro SD como una partición completa, es decir, expandir el Sistema Operativo para que ocupe todo el espacio en la tarjeta y que pueda ser usado como memoria de almacenamiento por la Raspberry Pi 3.

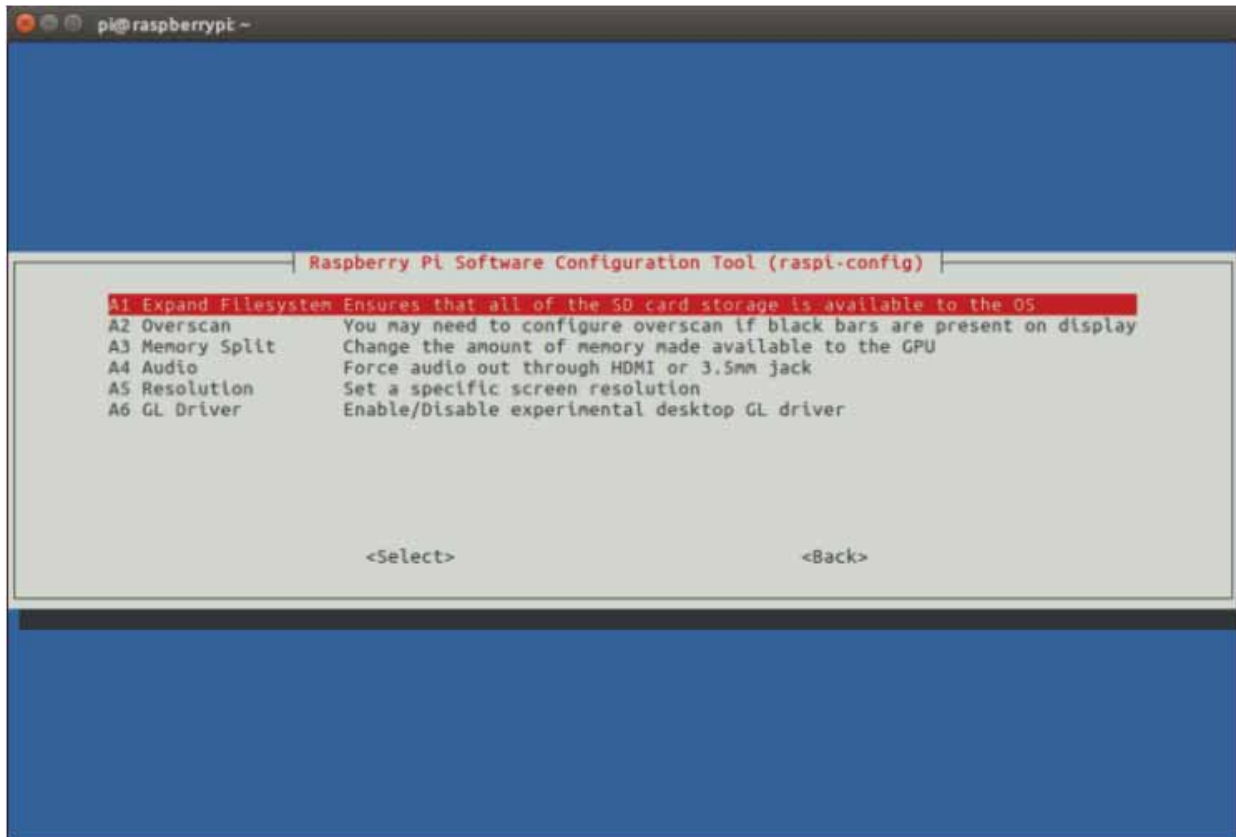


Figura 18: Ventana de configuración de Raspbian para elegir el tipo de Inicio.

Una vez que se hayan completado los dos pasos, seleccionar el botón "Finish" en la parte inferior de la página y la Raspberry Pi 3 debería reiniciarse automáticamente. Si no lo hace, se puede utilizar el siguiente comando en la terminal para reiniciar:

```
$ sudo reboot
```

Para este paso la instalación estará completa y podremos ver la interfaz gráfica de usuario de Raspbian (Figura 19).

Un dato para considerar es que el Sistema Operativo utiliza cerca de 4.2GB, si se utiliza una memoria micro SD de 8GB se habrá ocupado más del 50% de almacenamiento, y adicional a esto, la librería de visión por computadora OpenCV necesita más de 4GB de espacio libre en disco para compilar todo su código, por lo que es **necesario** contar con memorias micro SD mayores a 8GB.

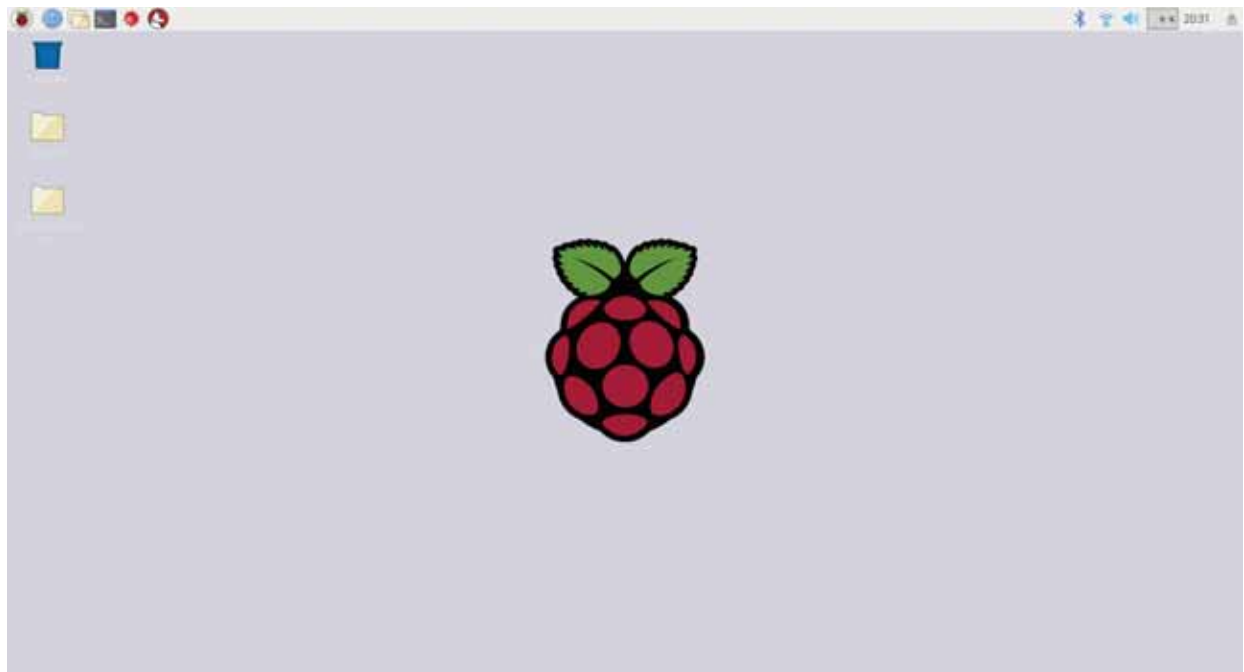


Figura 19: Escritorio de Raspbian Stretch.

B. Instalación de OpenCV

Habiendo completado la instalación de Raspbian Stretch (Apéndice A) procedemos a instalar OpenCV 3 en el Sistema Operativo con una ventana de terminal abierta.

El primer paso es actualizar todos los paquetes existentes:

```
$ sudo apt-get update && sudo apt-get upgrade
```

Se necesitan instalar algunas herramientas de desarrollador, incluyendo *CMake*, que nos ayudará a configurar el proceso de construcción de OpenCV:

```
$ sudo apt-get install build-essential cmake pkg-config
```

Después, se necesitan instalar algunos paquetes de imagen de E/S que permitan cargar distintos formatos de imagen del disco (p.e. JPEG, PNG, etc.):

```
$ sudo apt-get install libjpeg-dev libpng12-dev
$ sudo apt-get install libtiff5-dev libjasper-dev
```

Al igual que paquetes de imagen de E/S se necesitan paquetes de vídeo de E/S. Estas librerías permiten leer varios formatos de vídeo desde el disco, como también trabajar directamente con secuencias de vídeo:

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
```

La librería de OpenCV viene con un sub-módulo llamado *highgui* que es usado para mostrar imágenes en la pantalla y construir interfaces gráficas de usuario básicas. Para poder compilar el módulo *highgui*, se necesita instalar la librería de desarrollo *GTK*:

```
$ sudo apt-get install libgtk2.0-dev libgtk-3-dev
```

La mayoría de las aplicaciones dentro de OpenCV (llamadas operaciones de matrices) pueden ser optimizadas instalando algunas dependencias adicionales. Son importantes para dispositivos que cuentan con recursos limitados, tal es el caso del Raspberry Pi 3.

```
$ sudo apt-get install libatlas-base-dev gfortran
```

Ahora, se debe instalar Python 2.7 y Python 3, Raspbian 2017-09-07 ya los trae instalados por defecto, en caso contrario, habrá que descargar los archivos cabecera para poder compilar OpenCV con enlaces de Python:

```
$ sudo apt-get install python2.7-dev python3-dev
```

Es recomendable no saltarse este paso a pesar de ya tener Python instalado, se podría observar más adelante un error relacionado a que el archivo cabecera *python.h* no se encuentra al momento de hacer *make* para compilar OpenCV.

Una vez que las dependencias fueron instaladas se procederá a descargar el archivo 3.3.0 de OpenCV desde el repositorio oficial de OpenCV. Esta versión incluye el módulo *dnn* (*Deep Neural Network*):

```
$ cd ~
$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.0.zip
$ unzip opencv.zip
```

Se requiere una instalación completa de OpenCV 3 (para tener acceso a herramientas como *SIFT* y *SURF*), así que se necesita descargar el repositorio *opencv_contrib*:

```
$ wget -O opencv_contrib.zip
$ wget https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip
$ unzip opencv_contrib.zip
```

Las versiones de *opencv* y *opencv_contrib* deben ser las mismas (en este caso 3.3.0). Si son diferentes pueden presentarse varios errores al momento de compilar.

Antes de empezar a compilar OpenCV en la Raspberry Pi 3, se debe instalar *pip*, un manejador de paquetes de Python:

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo python3 get-pip.py
```

El mensaje de que ya está actualizado *pip* puede presentarse, pero es mejor hacerlo para la compilación de OpenCV.

Es una buena práctica en Python usar ambientes virtuales de algún tipo, es importante entender que es una herramienta especial que permite mantener las dependencias requeridas por diferentes proyectos en lugares separados al crear entornos en Python, estos se encuentran aislados e independientes:

```
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/.cache/pip
```

Ahora que *virtualenv* y *virtualenvwrapper* han sido instaladas, se necesita actualizar el archivo perfil *~/.profile* para incluir las siguientes líneas al final del archivo:

```
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh

$ echo -e "\n#_virtualenv_and_virtualenvwrapper" >> ~/.profile
$ echo "export_WORKON_HOME=$HOME/.virtualenvs" >> ~/.profile
$ echo "source_/usr/local/bin/virtualenvwrapper.sh" >> ~/.profile
```

Ya que el *~/.profile* ha sido actualizado, se necesita reiniciar para asegurar que los cambios tengan efecto. La mejor forma de reiniciar *~/.profile* es:

- Cerrar sesión, e iniciar nuevamente sesión.
- Cerrar la instancia de la terminal y abrir una nueva.
- O usar el comando *source*:

```
$ source ~/.profile
```

Se debe crear el ambiente virtual en Python, que se usará para el desarrollo de Visión por Computadora:

En caso de utilizar Python 2:

```
$ mkvirtualenv cv -p python2
```

En caso de utilizar Python 3:

```
$ mkvirtualenv cv -p python3
```

Cada vez que se reinicia el Raspberry Pi 3, necesitamos ingresar al ambiente virtual *cv* mediante el comando *workon* (es innecesario utilizar nuevamente *mkvirtualenv*, pues es de uso único):

```
$ source ~/.profile  
$ workon cv
```

Para asegurarnos que estamos en el ambiente virtual, observamos en la terminal las palabras (*cv*) antes del prompt, como se observa en la Figura:

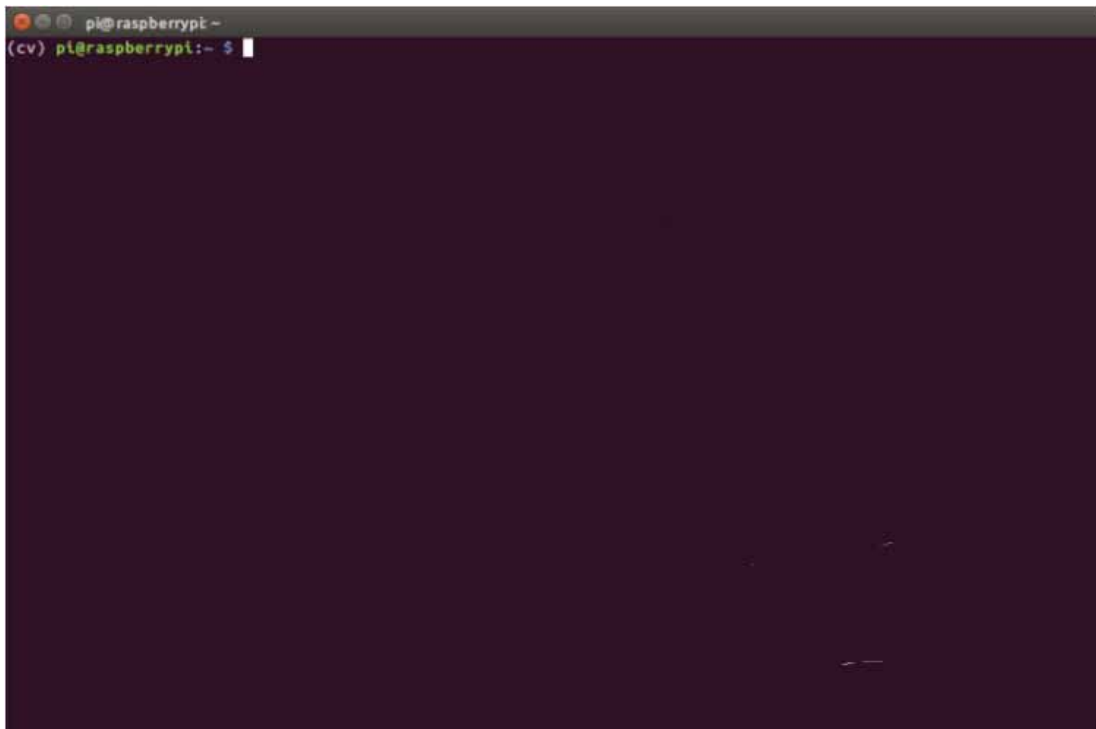


Figura 20: Ambiente Virtual en la terminal de Raspbian.

Cuando se pueda acceder al ambiente virtual (se debe permanecer por el resto de la instalación), debemos instalar *NumPy*, la única dependencia en Python usada para el procesamiento numérico:

```
$ pip install numpy
```

El proceso de descarga, instalación y compilación puede ser tardado, por lo que se debe ser paciente. Para verificar la instalación basta introducir el comando *top*, donde se verán los ciclos del CPU usados para compilar *NumPy*.

Ahora, ya se puede compilar e instalar OpenCV (recordar que debemos estar en el ambiente virtual), se puede configurar la construcción usando *CMake*:

```
$ cd ~/opencv-3.3.0/
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
-D BUILD_EXAMPLES=ON ..
```

Se iniciará el proceso de compilación de OpenCV para Python 3 por lo que es importante revisar que haya rutas válidas para el Intérprete, Librerías, *numpy* y ruta de paquetes. En caso de usar Python 2 se sigue el mismo procedimiento:

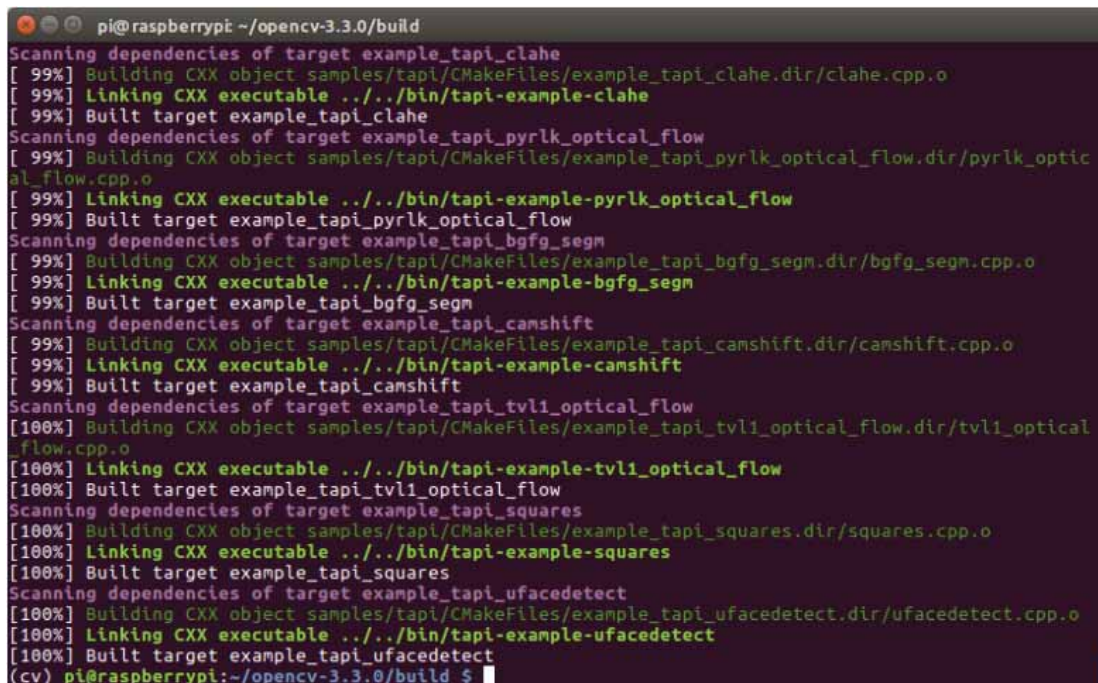
```
$ make
```

Se puede utilizar el comando:

```
$ make -j4
```

para utilizar los cuatro núcleos del Raspberry Pi 3, lo que reduce el tiempo de compilación bastante (de 5 horas aproximadamente, a 1 hora y media), pero no es recomendable ya que el proceso de compilación de OpenCV podría congelarse y es necesario borrar y volver a compilar la librería.

Una vez finalizado el proceso de compilación debería de verse como en la Figura 21



```
pi@raspberrypi: ~/opencv-3.3.0/build
Scanning dependencies of target example_tapi_clahe
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_clahe.dir/clahe.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-clahe
[ 99%] Built target example_tapi_clahe
Scanning dependencies of target example_tapi_pyrlk_optical_flow
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_pyrlk_optical_flow.dir/pyrlk_optical_flow.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-pyrlk_optical_flow
[ 99%] Built target example_tapi_pyrlk_optical_flow
Scanning dependencies of target example_tapi_bgfg_segm
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_bgfg_segm.dir/bgfg_segm.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-bgfg_segm
[ 99%] Built target example_tapi_bgfg_segm
Scanning dependencies of target example_tapi_camshift
[ 99%] Building CXX object samples/tapi/CMakeFiles/example_tapi_camshift.dir/camshift.cpp.o
[ 99%] Linking CXX executable ../../bin/tapi-example-camshift
[ 99%] Built target example_tapi_camshift
Scanning dependencies of target example_tapi_tv1_optical_flow
[100%] Building CXX object samples/tapi/CMakeFiles/example_tapi_tv1_optical_flow.dir/tv1_optical_flow.cpp.o
[100%] Linking CXX executable ../../bin/tapi-example-tv1_optical_flow
[100%] Built target example_tapi_tv1_optical_flow
Scanning dependencies of target example_tapi_squares
[100%] Building CXX object samples/tapi/CMakeFiles/example_tapi_squares.dir/squares.cpp.o
[100%] Linking CXX executable ../../bin/tapi-example-squares
[100%] Built target example_tapi_squares
Scanning dependencies of target example_tapi_ufacedetect
[100%] Building CXX object samples/tapi/CMakeFiles/example_tapi_ufacedetect.dir/ufacedetect.cpp.o
[100%] Linking CXX executable ../../bin/tapi-example-ufacedetect
[100%] Built target example_tapi_ufacedetect
(cv) pi@raspberrypi:~/opencv-3.3.0/build $
```

Figura 21: Instalación de OpenCV 3 en Raspbian Stretch 2017-09-07.

A partir de aquí, todo lo que se debe hacer es instalar OpenCV 3:

```
$ sudo make install
$ sudo ldconfig
```

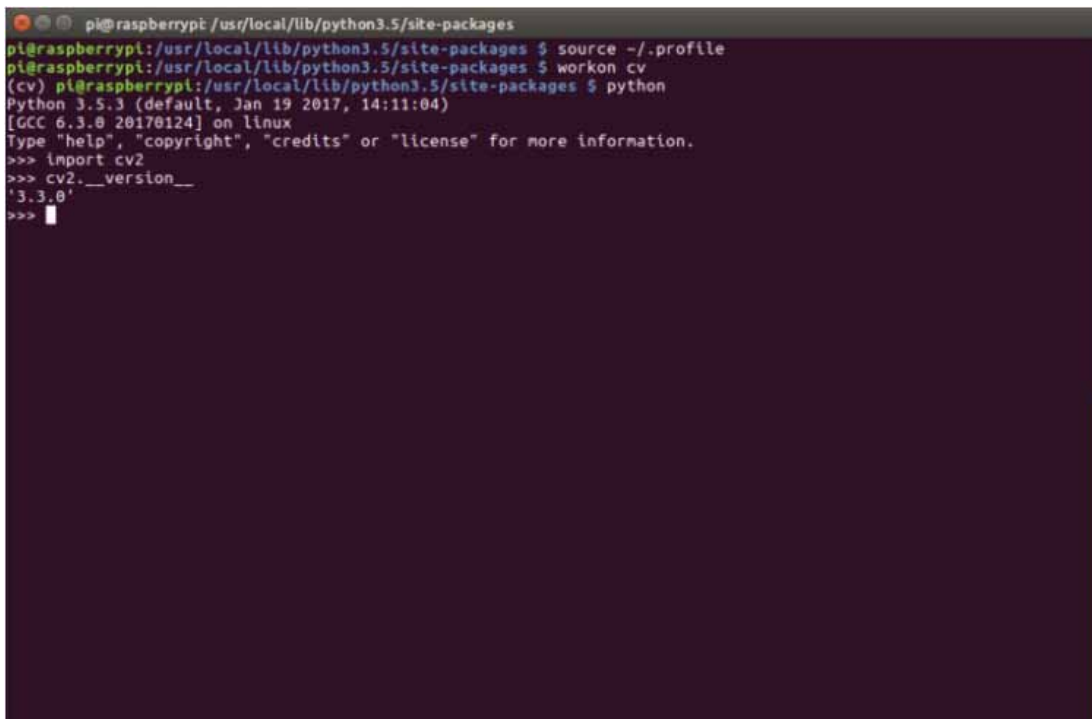
Después de usar el comando *make install*, los enlaces de OpenCV y Python deben de estar instalados en:

- Para el caso de Python 2
`/usr/local/lib/python2.7/site-packages`
- Para el caso de Python 3
`/usr/local/lib/python3.5/site-packages`

Para probar la instalación de OpenCV en Raspbian Stretch se debe abrir la terminal y ejecutar los comandos *source* y *workon* y finalmente importar los enlaces de Python y OpenCV:

```
$ source ~/.profile
$ workon cv
$ python
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>>
```

Se puede ver en la Figura 22 que OpenCV 3 ha sido instalado.



```
pi@raspberrypi: /usr/local/lib/python3.5/site-packages
pi@raspberrypi: /usr/local/lib/python3.5/site-packages $ source ~/.profile
pi@raspberrypi: /usr/local/lib/python3.5/site-packages $ workon cv
(cv) pi@raspberrypi: /usr/local/lib/python3.5/site-packages $ python
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>>
```

Figura 22: Confirmación de que OpenCV 3 ha sido instalado exitosamente en Raspbian Stretch.

C. Código

"configuracion.py"

```
import sys
import select
# Umbral de confianza de reconocimiento facial. Los valores por debajo
# de este umbral seran considerados positivos, este valor se puede modificar
# por uno mas bajo o mas alto si se es necesario.
LIMITE_POSITIVO = 3000.0

# Archivo donde se guardan los resultados del entrenamiento.
ARCH_ENTRENAMIENTO = 'resultEntrenamiento.xml'

# Directorios que contienen la imagenes que se entrenaran para el modelo.
DIRECTORIO_POS = './entrenamiento/positivo'
DIRECTORIO_NEG = './entrenamiento/negativo'

# Valor de las etiquetas que indican la confiabilidad.
ETIQUETA_POS = 1
ETIQUETA_NEG = 2

# Valor en pixeles de las imagenes que se usaran para entrenar el modelo.
ANCHO_ROSTRO = 92
ALTO_ROSTRO = 112

# Archivo de características de Haar para detectar el rostro
CAR_HAAR_ROSTRO = 'haarcascade_frontalface_alt.xml'
FACTOR_ESCALA_HAAR = 1.3
VECINO_MIN_HAAR = 4
TAM_MIN_HAAR = (30, 30)

def entrada_letra(letra):
# Funcion para introducir caracteres validos en stdin
    if select.select([sys.stdin], [], [], 0.0)[0]:
        entrada_caracter = sys.stdin.read(1)
        return entrada_caracter.lower() == letra.lower()
    return False
```

"config_rostro_haar.py"

```
import cv2
import configuracion

rostro_haar = cv2.CascadeClassifier(configuracion.CAR_HAAR_ROSTRO)

def unico_rostro(imagen):
    # Establece limites (x, y, ancho, alto) del rostro a escala de grises.
    # Si no se detectan rostros no regresa nada.
    rostros = rostro_haar.detectMultiScale(imagen,
        scaleFactor=configuracion.FACTOR_ESCALA_HAAR,
        minNeighbors=configuracion.VECINO_MIN_HAAR,
        minSize=configuracion.TAM_MIN_HAAR,
        flags=cv2.CASCADE_SCALE_IMAGE)
    if len(rostros) != 1:
        return None
    return rostros[0]

def tamMat(imagen, x, y, w, h):
    # Tamano x, y (esquina superior izquierda) y w, h (ancho y alto).
    tamMat_altura = int((configuracion.ALTO_ROSTRO / \
        float(configuracion.ANCHO_ROSTRO)) * w)
    mitad_y = y + h/2
    y1 = max(0, mitad_y-tamMat_altura/2)
    y2 = min(imagen.shape[0]-1, mitad_y+tamMat_altura/2)
    return imagen[y1:y2, x:x+w]

def redimensiona(imagen):
    # Redimensiona la imagen del rostro para el entrenamiento
    # y el reconocimiento.
    return cv2.resize(imagen,
        (configuracion.ANCHO_ROSTRO, configuracion.ALTO_ROSTRO),
        interpolation=cv2.INTER_LANCZOS4)
```

"rostros_positivos.py"

```
import glob
import os
import cv2
import picamera
import configuracion
import config_rostro_haar

# Prefijo del nombre de las imagenes capturadas positivas.
NOMBRE_ARCH_POS = 'img_positiva_'

if __name__ == '__main__':
# Inicializacion de la camara
    camera = picamera.PiCamera()

# Se crea el directorio de imagenes positivas en caso de que no exista.
    if not os.path.exists(configuracion.DIRECTORIO_POS):
        os.makedirs(configuracion.DIRECTORIO_POS)
# Se crea el ID de cada imagen capturada.
    archivos = sorted(glob.glob(os.path.join(configuracion.DIRECTORIO_POS,
        NOMBRE_ARCH_POS + '[0-9][0-9][0-9].pgm')))
    cuenta = 0
    if len(archivos) > 0:
        cuenta = int(archivos[-1][-7:-4])+1
    print 'Capturando imagenes positivas para el conjunto de entrenamiento.'
    print 'Presione f seguido de Enter para tomar una foto.'
    print 'Presione Control+C para salir.'
    camera.start_preview(fullscreen=False, window = (560,360,240,120))

    while True:
        if configuracion.entrada_letra('f'):
            print 'Capturando imagen'
# Aqui se introduce la ruta de la raiz del proyecto
            imagen = camera.capture('*RUTA*/entrenamiento/positivo/muestra.jpg')
            imagen = cv2.imread('*RUTA*/entrenamiento/positivo/muestra.jpg')
# Convierte la imagen capturada a escala de grises.
            escala_gris = cv2.cvtColor(imagen, cv2.COLOR_RGB2GRAY)
# Obtiene las coordenadas de un unico rostro reconocido.
            resultado = config_rostro_haar.unico_rostro(escala_gris)
            if resultado is None:
                print 'No se detecto ningun rostro, revise muestra.jpg \
                    \ para observar la imagen resultante.'
                continue
            x, y, w, h = resultado
# Se corta unicamente el rostro.
            tamMat = config_rostro_haar.tamMat(escala_gris, x, y, w, h)
# Guarda la imagen en un archivo.
            nombre_archivo = os.path.join(configuracion.DIRECTORIO_POS, \
                NOMBRE_ARCH_POS + '%03d.pgm' % cuenta)
```

```
cv2.imwrite(nombre_archivo, tamMat)
print 'Se ha encontrado un rostro, se guardara en el' \
'conjunto de entrenamiento.', nombre_archivo
cuenta += 1
```

"entrenamiento.py"

```
import fnmatch
import os
import cv2
import numpy as np
import configuracion
import config_rostro_haar

ARCHIVO_PROM_ENTREN = 'promedio_rostro.png'
ARCHIVO_EIGEN_POSITIVO = 'eigenface_positiva.png'
ARCHIVO_EIGEN_NEGATIVO = 'eigenface_negativa.png'

def busca_dir(directorio, match='*'):
    # Funcion iterativa que busca todos los archivos en un directorio
    for root, dirs, archivos in os.walk(directorio):
        for nombre_archivo in fnmatch.filter(archivos, match):
            yield os.path.join(root, nombre_archivo)

def prepara_imagen(nombre_archivo):
    # Funcion que lee la imagen en escala de grises y la redimensiona para
    # el conjunto de entrenamiento.
    return config_rostro_haar.redimensiona(cv2.imread(nombre_archivo,\
        cv2.IMREAD_GRAYSCALE))

def normaliza(X, bajo, alto, dtype=None):
    # Funcion que lleva a cabo el proceso de preprocesamiento de la imagen dado
    # un vector X entre un valor bajo y alto.
    X = np.asarray(X)
    minX, maxX = np.min(X), np.max(X)
    # Normaliza de [0...1].
    X = X - float(minX)
    X = X / float((maxX - minX))
    X = X * (alto-bajo)
    X = X + bajo
    # "dtype" devuelve informacion sobre un tipo de elemento especificado
    if dtype is None:
        return np.asarray(X)
    return np.asarray(X, dtype=dtype)

if __name__ == '__main__':
    print "Leyendo imagenes del conjunto de entrenamiento, espere."
    rostros = []
    etiquetas = []
    cuenta_positivos = 0
    cuenta_negativos = 0
    # Lee todas la imagenes positivas
    for nombre_archivo in busca_dir(configuracion.DIRECTORIO_POS, '*.pgm'):
```

```

        rostros.append(prepara_imagen(nombre_archivo))
        etiquetas.append(configuracion.ETIQUETA_POS)
        cuenta_positivos += 1
# Lee todas las imagenes negativas
    for nombre_archivo in busca_dir(configuracion.DIRECTORIO_NEG, '*.pgm'):
        rostros.append(prepara_imagen(nombre_archivo))
        etiquetas.append(configuracion.ETIQUETA_NEG)
        cuenta_negativos += 1
    print 'Se leyeron', cuenta_positivos, 'imagenes positivas', \
        cuenta_negativos, 'imagenes negativas.'

# Entrenamiento del modelo
    print 'Se esta entrenando el modelo, esta operacion puede tardar un poco.'
    lee_modelo = cv2.face.EigenFaceRecognizer_create()
    lee_modelo.train(np.asarray(rostros), np.asarray(etiquetas))

# Se guardan los resultado del modelo en resultEntrenamiento.xml
    lee_modelo.write(configuracion.ARCH_ENTRENAMIENTO)
    print 'Entrenamiento exitoso, datos guardados en', \
        configuracion.ARCH_ENTRENAMIENTO

# Guarda imagenes del promedio y las Eigenfaces resultantes.
    promedio = lee_modelo.getMean().reshape(rostros[0].shape)
    cv2.imwrite(ARCHIVO_PROM_ENTREN, normaliza(promedio, 0, 255, dtype=np.uint8))
    eigenvectors = lee_modelo.getEigenVectors()
    pos_eigenvector = eigenvectors[:,0].reshape(rostros[0].shape)
    cv2.imwrite(ARCHIVO_EIGEN_POSITIVO, normaliza(pos_eigenvector, 0, 255, \
        dtype=np.uint8))
    neg_eigenvector = eigenvectors[:,1].reshape(rostros[0].shape)
    cv2.imwrite(ARCHIVO_EIGEN_NEGATIVO, normaliza(neg_eigenvector, 0, 255, \
        dtype=np.uint8))

```

"reconocimiento.py"

```
import cv2
import configuracion
import config_rostro_haar
import picamera

if __name__ == '__main__':
# Carga los datos del entrenamiento
    print 'Cargando datos del entrenamiento, espere.'
    lee_modelo = cv2.face.EigenFaceRecognizer_create()
    lee_modelo.read(configuracion.ARCH_ENTRENAMIENTO)
    print 'Datos del entrenamiento cargados.'
# Inicializacion de la camara
    camera = picamera.PiCamera()

    print 'Presione f seguido de Enter para tomar una foto.'
    print 'Presione Control+C para salir.'
    camera.start_preview(fullscreen=False, window = (560,360,240,120))

    while True:
        if configuracion.entrada_letra('f'):
# Aqui se introduce la ruta de la raiz del proyecto
            imagen = camera.capture('*RUTA*/muestra.jpg')
            imagen = cv2.imread('*RUTA*/muestra.jpg')
# Convierte la imagen capturada a escala de grises.
            escala_gris = cv2.cvtColor(imagen, cv2.COLOR_RGB2GRAY)
# Obtiene las coordenadas de un unico rostro reconocido.
            resultado = config_rostro_haar.unico_rostro(escala_gris)
            if resultado is None:
                print 'No se detecto ningun rostro, revise muestra.jpg \
                    \ para observar la imagen resultante.'
            else:
                x, y, w, h = resultado
# Se corta unicamente el rostro.
                tamMat = config_rostro_haar.redimensiona(config_rostro_haar.tamMat\
                    (escala_gris, x, y, w, h))
# Compara la imagen capturada con el modelo de entrenamiento.
                etiqueta, veracidad = lee_modelo.predict(tamMat)
                print 'Detectado un rostro {0} con veracidad de {1}'.format(
                    'positivo' if etiqueta == configuracion.ETIQUETA_POS \
                    else 'negativo',
                    veracidad)
                if etiqueta == configuracion.ETIQUETA_POS and veracidad < \
                    configuracion.LIMITE_POSITIVO:
                    print 'Rostro reconocido.'
                else:
                    print 'Rostro no reconocido, intente de nuevo.'
```