



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería Informática

**SISTEMA DE RECONOCIMIENTO FACIAL PARA DISPOSITIVOS  
MÓVILES Y ROBOT DE TELEPRESENCIA**

D. Alonso Rodrigo Serrano Alvarez

Dirigido por: Dr. Ángel Pérez de Madrid y Pablo

Codirigido por: Dra. Carolina Mañoso Hierro

Curso: 2017/2018





## SISTEMA DE RECONOCIMIENTO FACIAL PARA DISPOSITIVOS MÓVILES Y ROBOT DE TELEPRESENCIA

Proyecto de Fin de Grado en Ingeniería Informática  
de modalidad *específica*

Realizado por: D. Alonso Rodrigo Serrano Alvarez

Dirigido por: Dr. Ángel Pérez de Madrid y Pablo

Codirigido por: Dra. Carolina Mañoso Hierro

Fecha de lectura y defensa: .....



## **Resumen**

Desde su aparición, la tecnología de reconocimiento facial se ha empleado tradicionalmente en sistemas de seguridad por parte de empresas y agencias gubernamentales. La aparición de varios servicios *online* permite que cualquier ciudadano con conexión a Internet pueda acceder a esta herramienta tan valiosa. Sin embargo, la novedad de la tecnología hace que no existan demasiadas aplicaciones sencillas para el usuario común.

Este proyecto busca crear una biblioteca de funciones y un conjunto de aplicaciones que facilite la tarea del reconocimiento facial. Así mismo, este proyecto busca que el *software* desarrollado sea instalado en una plataforma de *hardware* portátil y así poder formar parte de un robot de telepresencia, ser accedido por dispositivos móviles mediante comunicación inalámbrica y ser empleado como equipo de escritorio.

Tras la elección de la Raspberry Pi como plataforma de *hardware* portátil, se ha desarrollado una biblioteca de funciones en Python llamada FCModule que se encarga de la captura, detección y recorte de caras, comunicación con el servicio en la nube Microsoft Azure para el envío de los recortes y obtención del identificador del sujeto más probable, etiquetado de los rostros y, finalmente, devolución de los resultados en forma de texto o imagen. Así mismo, se han desarrollado tres aplicaciones que emplean esta biblioteca. FaceRecon es un ejemplo de uso de la biblioteca y sirve de base para el desarrollo de las otras dos aplicaciones. FaceBT es una aplicación que funciona en segundo plano y permite la conexión de otros dispositivos a la Raspberry Pi para que empleen las funciones de reconocimiento facial. La aplicación FacePi dispone de una interfaz gráfica para el uso de las funciones de reconocimiento facial de una manera sencilla para el usuario común. Finalmente, se ha desarrollado una aplicación para dispositivos Android, llamada FacePal, que permite agregar personas y caras al sistema de reconocimiento facial.

## **Palabras clave**

Visión artificial, reconocimiento facial, Android, Raspberry Pi, DIY, IoT, Python, Azure, Java

## Title

FACIAL RECOGNITION SYSTEM FOR MOBILE DEVICES AND TELEPRESENCE ROBOTS

## Abstract

Since its first appearance, facial recognition technology has been traditionally used by corporations and governmental agencies mainly as security system. The appearance of diverse online services allows any user with an Internet connection to access this valuable tool. However, the novelty of this technology hinders the development of friendly apps available to the common user.

This project seeks the development of a function library and a set of applications that facilitates facial recognition tasks. Also, this project pursues that the developed software will be installed in a mobile hardware platform, therefore allowing it to be a part of a telepresence robot, accessible by mobile devices via wireless connection and its usage as a desktop device.

After selecting Raspberry Pi as the mobile hardware platform, a function library named FCModule has been developed in Python. This library tasks include: image capture, facial detection and crop, communication with Microsoft Azure's cloud service to send the cropped images and recover of the identification of most probable candidates, face tagging and, finally, the return of the results as text or image. Also, three applications that use this library have been developed. FaceRecon is an example of the usage of the library and serves as reference for the other applications. FaceBT is an application that works in the background and allows the connection of other devices to the Raspberry Pi, allowing the use of the facial recognition functions. The FacePi application has a graphic user interface providing the user an easy access to the facial recognition functions. Finally, an Android application named FacePal has been developed that allows the addition of new persons and faces to the facial recognition system.

## Keywords

Computer vision, facial recognition, Android, Raspberry Pi, DIY, IoT, Python, Azure, Java

# Índice

<b>1. Introducción .....</b>	<b>13</b>
1.1. Introducción .....	13
1.2. Objetivos .....	14
1.3. Metodología .....	16
1.4. Estructura de la memoria.....	18
<b>2. Análisis de herramientas para el desarrollo de las aplicaciones .....</b>	<b>19</b>
2.1. <i>Hardware</i> .....	19
2.2. Servicios en la nube.....	23
2.3. Bibliotecas de visión artificial.....	26
2.4. Lenguajes de programación .....	27
2.5. Elección de herramientas.....	28
2.5.1. <i>Hardware</i> .....	28
2.5.2. Servicio en la nube .....	29
2.5.3. Bibliotecas de visión artificial.....	36
2.6. Elección del lenguaje.....	37
<b>3. Diseño y desarrollo .....</b>	<b>39</b>
3.1. FCModule .....	39
3.1.1. Face .....	40
3.1.2. FaceCrop.....	41
3.1.3. FCTools .....	42
3.2. FaceRecon .....	43
3.3. FacePi .....	46
3.4. FaceBT .....	48
3.5. FacePal .....	49
3.5.1. Consideraciones previas.....	49
3.5.2. Código.....	52
3.5.3. Funcionamiento .....	54
<b>4. Pruebas .....</b>	<b>59</b>
4.1. Comprobación con imágenes existentes .....	59
4.1.1. Prueba Bill Gates vs. Bill Gates.....	59
4.1.2. Prueba Ben Linus vs. Ben Linus .....	62
4.1.3. Prueba Bill Gates vs. Ben Linus.....	65
4.1.4. Prueba Bill Gates vs. Bill Gates Impersonator.....	66

4.2.	Prueba con imágenes existentes y reconocimiento con FaceRecon .....	67
4.3.	Entrenamiento y reconocimiento en FacePal.....	69
4.4.	Entrenamiento y reconocimiento FacePal+FaceRecon.....	71
4.5.	Reconocimiento múltiple .....	73
4.6.	Reconocimiento múltiple con FacePi .....	76
4.7.	Reconocimiento y comunicación mediante FaceBT.....	77
<b>5.</b>	<b>Presupuesto .....</b>	<b>81</b>
5.1.	Estimación de costes de desarrollo.....	81
5.2.	Estimación de costes de adquisición y uso .....	82
<b>6.</b>	<b>Conclusiones y trabajos futuros .....</b>	<b>85</b>
6.1.	Conclusiones.....	85
6.2.	Futuras ampliaciones .....	88
	<b>Bibliografía.....</b>	<b>91</b>
	<b>Anexo A – Contenido del PFG .....</b>	<b>95</b>
	<b>Anexo B – Manual de instalación .....</b>	<b>97</b>
	<b>Anexo C – Manual de usuario .....</b>	<b>101</b>
	<b>Anexo D – Materiales empleados .....</b>	<b>105</b>

## **Lista de figuras**

Figura 1. Cronograma de fases.....	17
Figura 2 Sistema de telepresencia con iPad (Fuente juguetronica.com).....	20
Figura 3. Raspberry Pi 3 Modelo B (Fuente raspberry.org).....	22
Figura 4. Cámara de Raspberry Pi (Fuente reichelt.com).....	29
Figura 5. Panel de Azure al darse de alta en el servicio QeQ.....	32
Figura 6. Ejemplo de organización de entidades (Fuente Microsoft).....	33
Figura 7. Diagrama de clases de FCModule.....	39
Figura 8. Ejemplo de esquinas del rectángulo.....	40
Figura 9. Diagrama de clases de FaceRecon.....	44
Figura 10. Descripción de FaceRecon.....	45
Figura 11. Ejecución de FaceRecon.....	46
Figura 12. Pantalla principal de FacePi.....	47
Figura 13. Ejemplo de conexión con FaceBT.....	48
Figura 14. Esquema general de FacePal.....	51
Figura 15. Pantalla principal de FacePal.....	55
Figura 16. Menú de configuración.....	55
Figura 17. Pantalla de configuración.....	56
Figura 18. Retratos de Bill Gates (Fuente Google).....	60
Figura 19. Imagen de prueba de Bill Gates (Fuente Microsoft).....	60
Figura 20. Ejemplo de ejecución durante la prueba Bill vs. Bill.....	61
Figura 21. Gráfica Bill vs. Bill.....	61
Figura 22. Caras elegidas de Ben Linus (Fuente Google).....	62
Figura 23. Imagen de prueba de Ben Linus (Michael Emerson) (Fuente Wikipedia)....	63
Figura 24. Prueba de los diez rostros de Ben Linus.....	63
Figura 25. Ejecución con la imagen de prueba de Ben Linus antes del entrenamiento.	63
Figura 26. Ejecución con imagen de prueba de Ben Linus tras el entrenamiento.....	64
Figura 27. Gráfica Ben vs. Ben.....	64
Figura 28. Imitador de Bill Gates (Fuente lookalike.com).....	66
Figura 29. Ejecución de la prueba con Bill Gates Impersonator.....	67
Figura 30. Fotos de entrenamiento de Admin.....	67

Figura 31. Primer reconocimiento de Admin.....	68
Figura 32. Imagen de Admin con iluminación alternativa.....	68
Figura 33. Prueba de identificación y creación del sujeto Akbar Espaillat.....	69
Figura 34. Creación de persona Bastian Schiffthaler .....	70
Figura 35. Cara agregada a Bastian Schiffthaler.....	70
Figura 36. Creación del sujeto Javier Alvarez e imagen añadida. ....	71
Figura 37. Reconocimiento de Akbar y Admin.....	72
Figura 38. Reconocimiento de Javi y Admin.....	72
Figura 39. Reconocimiento de Bastian.....	72
Figura 40. Nuevas personas.....	73
Figura 41. Foto grupal 1.....	74
Figura 42. Foto grupal 2.....	75
Figura 43. Imagen adicional de Oihane. ....	75
Figura 44. Foto grupal 3.....	76
Figura 45. Ejecución de FacePi. ....	77
Figura 46. Raspberry Pi con batería externa. ....	78
Figura 47. Comunicación con FaceBT.....	79
Figura 48. Objetos identificados como rostros. ....	87
Figura 49. Captura de pantalla de Win32 Disk Imager.....	97
Figura 50. Conexión de la cámara con la Raspberry Pi.....	98
Figura 51. Conexión a red Wifi. ....	98
Figura 52. Menú de configuración de FacePal. ....	100
Figura 53. Pantalla principal de FacePi.....	101
Figura 54. Archivo de configuración FaceBT. ....	102
Figura 55. Pantalla principal de FacePal .....	104
Figura 56. Pantalla de entrenamiento de FacePal.....	104

## **Lista de tablas**

Tabla 1. Comparación entre plataformas de hardware.....	23
Tabla 2. Comparativa entre servicios de reconocimiento facial.....	26
Tabla 3. Resultado de la prueba Bill Gates vs. Ben Linus.....	66
Tabla 4. Resultados de la prueba Bill Gates vs. Bill Gates Impersonator.....	67
Tabla 5. LOC del proyecto.....	81
Tabla 6. Coste inicial.....	83
Tabla 7. Coste mensual.....	84

## Listado de siglas, abreviaturas y acrónimos

**API** (*Application Programming Interface*): interfaz de programación de aplicaciones. Interfaz para emplear el *software* de una biblioteca sin entrar en los detalles de funcionamiento de la misma.

**GPIO** (*General Purpose Input/Output*): pin genérico con comportamiento personalizable.

**GB** (*Gigabyte*): unidad de almacenamiento de información, equivale a  $10^9$  bytes.

**Ghz** (*Gigaherzio*): unidad de medida de frecuencia, equivale a  $10^9$  hercios.

**GPU** (*Graphics Processing Unit*): unidad de procesamiento gráfico. Procesador dedicado al procesamiento de gráficos.

**GPS** (*Global Positioning System*): sistema de posicionamiento global. Sistema para determinar la posición de un objeto en la superficie de la Tierra.

**HDD** (*Hard Disk Drive*): unidad de disco duro. Elemento para el almacenamiento estable de información.

**HDMI** (*High-Definition Multimedia Interface*): interfaz multimedia de alta definición. Interfaz para la transmisión de audio/video.

**IDE** (*Integrated Development Environment*): entorno de desarrollo integrado. Aplicación para facilitar la labor de desarrollo de los programadores.

**JSON** (*JavaScript Object Notation*): notación de objetos JavaScript. Formato de texto para el intercambio de datos.

**LOC** (*Lines of Code*): líneas de código. Métrica de estimación del tamaño de un programa.

**Open source**: código abierto. Modelo de desarrollo basado en el acceso libre al código fuente y la colaboración abierta.

**RAM** (*Random Access Memory*): memoria de acceso aleatorio. Memoria empleada por los ordenadores para almacenar información e instrucciones.

**RFCOMM** (*Radio frequency communication*): protocolo de Bluetooth para transporte de datos.

**SDK** (*Software development kit*): kit de desarrollo de *software*. Conjunto de herramientas que permiten el desarrollo de aplicaciones.

**USB** (*Universal Serial Bus*): bus universal en serie. Estándar que define los cables, conectores y protocolos usados para conectar dispositivos electrónicos.

**WIFI**: tecnología de transmisión de datos inalámbrica local.

# 1. Introducción

En este capítulo se describe la situación actual con respecto a los sistemas de reconocimiento facial. Asimismo, se describen los objetivos que persigue este proyecto y la metodología que se ha empleado para conseguirlos. Por último, se realiza una descripción de esta memoria para facilitar la lectura de la misma.

## 1.1. Introducción

Hasta hace unos pocos años, el reconocimiento facial parecía más propio de las películas de ciencia ficción que una realidad y para algunas personas sigue pareciendo una innovación tecnológica.

Sin embargo, la historia del reconocimiento facial comenzó hace varias décadas. Entre 1964 y 1966 Woodrow “Woody” Bledsoe, Helen Chan y Charles Bisson, comenzaron el desarrollo de los primeros programas de ordenador para reconocer rostros de personas. El problema que querían resolver era que dada una colección de imágenes tomadas a personas arrestadas y otra fotografía, un programa de ordenador pudiese devolver un subconjunto de la colección de imágenes con las caras con mayor parecido. Los primeros resultados medían el cociente del número de caras devueltas por el sistema frente al número de caras de la colección (1).

La investigación progresaba, hasta que en 1968 en pruebas en las que a personas y al *software* desarrollado se les daban las mismas tareas de identificación, el *software* superó al ser humano, concluyendo que las pruebas fueron exitosas (2).

Desde entonces, el reconocimiento facial ha avanzado discretamente, pero sin pausa. Uno de los principales motivos por el que este tipo de técnicas han pasado bastante desapercibidas, es debido a que algunas agencias gubernamentales y grandes corporaciones han investigado e invertido en estos sistemas para su uso propio. De hecho, la investigación inicial de Woodrow Bledsoe, fue financiada por una agencia de inteligencia cuyo nombre se desconoce y buena parte de su trabajo no fue nunca publicada (3).

El uso más frecuente en la actualidad es como sistema de vigilancia en agencias gubernamentales, seguridad en los aeropuertos y estaciones de transporte público. El primer sistema instalado fue en 2011 en el aeropuerto de Tocumen, Panamá (4), y desde entonces más aeropuertos están instalando estos sistemas.

Con respecto a seguridad, también se utiliza el reconocimiento facial en ciencia forense, en la identificación de cuerpos. El reconocimiento facial fue uno de los métodos empleados en la identificación del cuerpo de Osama Bin Laden en 2011 (5).

Un uso muy extendido de los sistemas de reconocimiento facial es la detección de jugadores conflictivos en los casinos. En Las Vegas existe una lista negra pública denominada *Black Book* (6) mantenida por las autoridades gubernamentales de control de juego, en la que se guardan los perfiles de los jugadores tramposos. Las fotos y nombre de los miembros de esa lista se emplean con sistemas de reconocimiento facial para evitar a los jugadores tramposos en los casinos.

Con respecto a las redes sociales, recientemente Facebook adquirió DeepFace, un sistema de reconocimiento facial que emplea *deep learning* para realizar el reconocimiento facial. Este sistema fue entrenado con 4 millones de fotos que los usuarios habían subido a sus cuentas. Según sus autores, la precisión en sus pruebas es del 97.35% (7). DeepFace se emplea dentro de Facebook para etiquetar e identificar personas en las fotos que sus usuarios publican.

En general, se puede concluir que los sistemas de reconocimiento facial son casi de uso exclusivo de las agencias gubernamentales y grandes empresas. Acercar más al usuario común esta tecnología fue una de las motivaciones para realizar este proyecto.

## 1.2. Objetivos

El objetivo principal de este proyecto es desarrollar un sistema de reconocimiento facial que pueda ser empleado por dispositivos móviles, robots de telepresencia o como equipo de escritorio.

Además, este proyecto busca conseguir que el sistema sea rápido, eficaz, fácil de usar, fácil de incorporar, fácil de mantener y de bajo coste.

Para conseguir estos objetivos, en primer lugar, se va a escoger una plataforma *hardware* que permita conseguir los objetivos descritos anteriormente.

A continuación, se va a desarrollar una biblioteca de funciones que realice al menos las siguientes tareas:

- gestión de capturas de imágenes.
- detección y recorte de caras.
- comunicación con un servicio existente de reconocimiento facial en la nube.
- etiquetado de las caras en la imagen original.
- devolución de resultados.

Para el uso de esta biblioteca de funciones, se van a desarrollar tres aplicaciones que emplean dicha biblioteca:

- una primera aplicación funcionará como ejemplo de uso de esta biblioteca. Esta aplicación servirá además como base para realizar las pruebas de verificación del correcto funcionamiento de la biblioteca.
- una segunda aplicación dispondrá de una interfaz gráfica para que el usuario que lo prefiera pueda emplear las funciones de reconocimiento facial usando la Raspberry Pi como equipo de escritorio.
- una aplicación de comunicación que permita a otros dispositivos comunicarse inalámbricamente con este sistema y aprovechar sus funciones de reconocimiento facial sin necesidad de que el usuario tenga que manejar directamente el dispositivo donde este sistema esté instalado.

Finalmente, se va a desarrollar una aplicación de entrenamiento para dispositivos móviles que sirva de complemento a las demás aplicaciones, facilitando así la incorporación de nuevas personas y caras al sistema sin tener que hacer uso directo de la plataforma *hardware* donde se instalará la biblioteca.

La idea principal de este proyecto es que se pueda emplear el sistema de reconocimiento facial que se va a desarrollar conjuntamente a un robot de telepresencia o de forma autónoma, de tal manera que el robot o dispositivo móvil acceda a las

funciones del sistema mediante la aplicación de comunicación desarrollada para obtener datos de identificación de individuos. La aplicación de entrenamiento permitirá una adición sencilla de nuevas personas al sistema.

De esta forma se obtendrán varias ventajas:

- la adición de nuevas personas se puede realizar mediante un dispositivo móvil con total independencia del sistema de reconocimiento facial. no siendo necesario que el usuario tenga encendido o apagado el sistema, ni que esté físicamente presente.
- el sistema puede ser acoplado o desacoplado fácilmente del robot de telepresencia y pasar a usarse como equipo de escritorio, si así se prefiere. Al mismo tiempo podrá ser usado de forma totalmente independiente.
- el sistema podrá ser emplearlo por usuarios con distintos perfiles, no siendo necesario tener conocimientos de programación para su uso. Sin embargo, el sistema ofrecerá una vía a los desarrolladores para emplear las funciones del sistema fácilmente en sus propias aplicaciones.

Entre los posibles usos están:

- asistencia en reuniones, reconociendo a los participantes.
- uso conjunto con robots de patrulla para realizar vigilancia.
- sistemas de seguridad caseros.
- etc.

### **1.3. Metodología**

El desarrollo del proyecto se dividió en distintas fases:

- Análisis (Octubre - Diciembre): durante esta fase se buscaron y evaluaron las herramientas existentes adecuadas para realizar este proyecto. Esta fase incluyó la lectura de documentación sobre las distintas herramientas, búsqueda de ejemplos de uso de bibliotecas de *software* y pruebas comparativas entre servicios en la nube. Además, se empleó este periodo para

la adquisición de la base teórica necesaria para poder realizar el resto del proyecto.

- Desarrollo (Diciembre - Abril): una vez elegidas las herramientas, se comenzó con el desarrollo de la biblioteca de funciones y las aplicaciones de este proyecto. También se comenzó la realización de pruebas para verificar el correcto funcionamiento de la comunicación con el servicio en la nube. Durante estas pruebas, surgieron problemas con el servicio en la nube elegido inicialmente que provocaron un cambio en el desarrollo de las aplicaciones; los detalles del cambio del servicio en la nube se explican en el Capítulo 2.
- Evaluación (Marzo - Junio): durante esta fase, se realizaron pruebas de las aplicaciones, la corrección de errores y se codificaron nuevas funciones.
- Redacción (Abril - Junio): en esta fase se redactaron los capítulos de la presente memoria y los anexos con los manuales de instalación y uso.

La Figura 1 muestra en un diagrama de Gantt la duración de las distintas fases de este proyecto y las actividades realizadas.

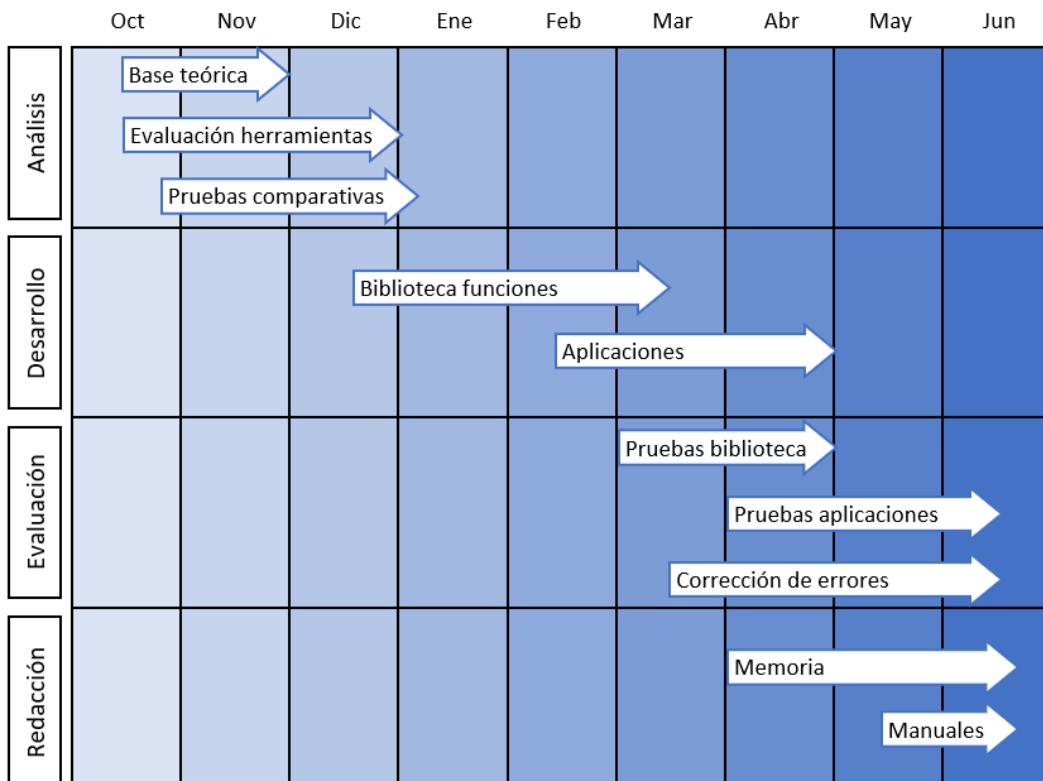


Figura 1. Cronograma de fases.

## 1.4. Estructura de la memoria

La memoria de este proyecto se estructura en seis capítulos y cuatro anexos que se describen a continuación:

### Capítulos:

1. Introducción: capítulo introductorio para facilitar la lectura de la memoria. Recoge los antecedentes, objetivos, metodología empleada y estructura de la memoria.
2. Análisis de herramientas para el desarrollo de las aplicaciones: en este capítulo se analizan diversas herramientas existentes para la elaboración de este proyecto y la justificación de uso de aquellas elegidas.
3. Diseño y desarrollo: este capítulo se dedica al desarrollo de la biblioteca de funciones y de las aplicaciones de este proyecto, así como sus características más relevantes a nivel de código.
4. Pruebas: a lo largo de este capítulo se detallan las distintas pruebas realizadas al conjunto de aplicaciones desarrolladas y se presentan evidencias de su funcionamiento.
5. Presupuesto: este capítulo propone una estimación de costes de desarrollo de las aplicaciones y un ejemplo de coste de instalación y uso en empresa.
6. Conclusiones y trabajos futuros: en este capítulo se exponen las conclusiones obtenidas a lo largo del proyecto y se presentan posibles ampliaciones, futuros usos y oportunidades de mejora del sistema desarrollado en este proyecto.

### Anexos:

- A. Contenido del PFG: explica la estructura de directorios del material creado en este proyecto y su contenido.
- B. Manual de instalación: contiene la información de instalación de aplicaciones y puesta en marcha de la Raspberry Pi.
- C. Manual de uso: contiene información sobre cómo emplear las aplicaciones desarrolladas en este proyecto.
- D. Materiales empleados: describe los materiales empleados para la elaboración de esta memoria y del *software* desarrollado.

## 2. Análisis de herramientas para el desarrollo de las aplicaciones

En este capítulo se hace un análisis de las distintas herramientas existentes que pueden servir para el desarrollo de este proyecto.

En primer lugar, se analizan las opciones disponibles y se citan sus características más relevantes. Se indican las ventajas y desventajas de las diversas opciones.

El capítulo finaliza con un apartado donde se muestran y justifican las herramientas finalmente elegidas para realizar este proyecto.

### 2.1. *Hardware*

Se comenzó con la búsqueda de la plataforma de *hardware* adecuada para la albergar las aplicaciones de este proyecto. Se buscaron dos plataformas: una para albergar la biblioteca de funciones y las aplicaciones que la emplean y otra para albergar la aplicación de entrenamiento.

Las características deseables son las siguientes:

- Bajo coste.
- Recursos (RAM, HDD) suficientes para cumplir las demandas del *software*.
- Tamaño adecuado para cada parte del sistema.
- Documentación amplia.
- Soporte de la comunidad.
- Integración de componentes.

A continuación, se describen las plataformas analizadas.

#### Equipo de escritorio

Se trata de la opción más versátil, los equipos de escritorio pueden presentarse de muchas formas, tamaños y características.

Son altamente modulares. Estos equipos permiten configuración a la carta para las características del sistema. Sin embargo, aunque este tipo de equipo tiene los recursos adecuados, su tamaño excesivo lo hace inviable para su acoplamiento a un

sistema de telepresencia u otro tipo de robot. Además, el coste de este tipo de equipo tampoco es un factor favorable para su elección.

### Equipos portátiles

Comparado con los equipos de sobremesa, el portátil es más fácil de acoplar a un robot de telepresencia. Por el contrario, se sacrifica la modularidad, los portátiles tienen menos capacidad de cambiar componentes. Si bien un portátil de gama media tiene recursos suficientes para los requisitos del proyecto, su precio y su consumo energético dependiente del uso de baterías y/o cable de conexión a red eléctrica no lo convierten tampoco en un candidato viable.

### Tablets

En algunos robots de telepresencia se han empleado *tablets* como el iPad con el fin de realizar un *streaming* constante (Figura 2). Los *tablets* tipo iPad son de poco peso y volumen aceptable, pero presentan ciertas limitaciones como el espacio de almacenamiento o su menor versatilidad para crear y usar aplicaciones frente a otro tipo de *tablets*.



Figura 2 Sistema de telepresencia con iPad (Fuente juguetronica.com).

Existen otros tipos de *tablet* como los basados en Android, con alta versatilidad para crear aplicaciones, o los Surface, con vastos recursos de *hardware* para

implementar el sistema pudiendo usar Windows y Linux. Sin embargo, este tipo de *tablets* tienen un precio poco asequible y el consumo energético suele ser parecido al de un portátil.

### Teléfonos móviles

Su pequeño tamaño les hace más viables que los sistemas anteriores y tienen gran cantidad de componentes integrados (cámara, Wifi, Bluetooth, 4G, etc.). Sin embargo, según el tipo de móvil la programación es limitada a un tipo de lenguaje y el *hardware* puede ser insuficiente salvo que se adquiera un móvil de gama alta.

Existen plataformas basadas en tres sistemas: iOS, Android y Windows Phone.

No se pudo valorar iOS para este proyecto puesto que no se disponía de ningún dispositivo para realizar pruebas.

Windows Phone es un sistema con poca presencia en el mercado y en breve estará obsoleto, puesto que Microsoft abandona la plataforma en favor de sistemas más extendidos (8).

Android es el sistema para móviles más extendido, dispone de herramientas que facilitan la programación y numerosa documentación, ejemplos y amplio soporte de la comunidad de programadores.

### Raspberry Pi 3

La familia de plataformas Raspberry Pi (9) son un conjunto de diferentes modelos de ordenadores de pequeño tamaño con gran cantidad de componentes integrados, Bluetooth, Wifi, HDMI, USB... (Figura 3).

A todos los efectos es un ordenador y puede usarse como tal para navegar por Internet, emplear programas de ofimática, ver recursos multimedia y muchas más aplicaciones.

La principal ventaja de este sistema es su pequeño tamaño (cabe en la palma de la mano) que permite su fácil acoplamiento a casi cualquier tipo de robot o llevarlo de forma autónoma. También destaca su bajo consumo energético, que permite acoplarlo a la batería del robot sin que suponga un consumo excesivo.

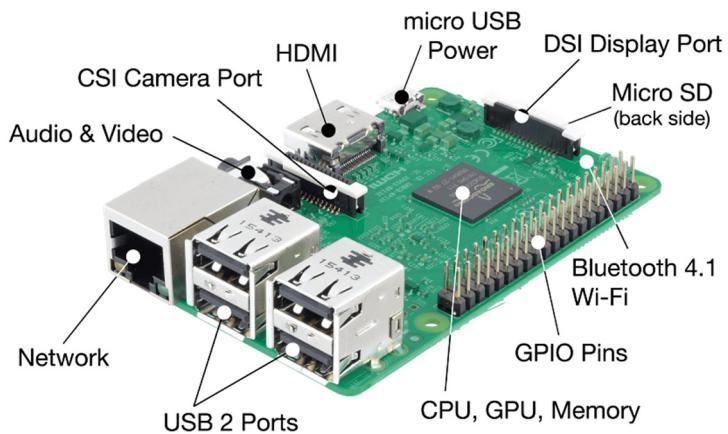


Figura 3. Raspberry Pi 3 Modelo B (Fuente [raspberry.org](http://raspberry.org)).

Todo el *software* que emplea es *open source*, lo que permite la instalación de diferentes sistemas operativos y programas. Incluye el uso de una versión de Linux llamada Raspbian muy versátil y de fácil uso. Además, dispone de ciertos intérpretes de lenguaje y compiladores ya integrados para su uso (Python, C, etc).

El coste de esta plataforma es muy bajo, en torno a 40 €, lo cual la hace muy asequible.

Existe una gran cantidad de documentación y soporte de la comunidad de desarrolladores, además de gran cantidad de ejemplos sobre proyectos que pueden realizarse con esta plataforma, desde servidores web hasta cajas fuertes bloqueadas por GPS.

Otra ventaja es que dispone de varios puertos USB para poder conectar distintos componentes, una conexión para una cámara exclusiva de la plataforma y pines GPIO que le permiten conectar con otras plataformas o componentes como *chips* de radio RF, pantallas externas, sensores, efectores, etc.

El principal inconveniente es su limitación de recursos. El modelo B tiene 1 Gb de RAM compartido con la GPU y un procesador de 1.2 Ghz.

Tabla 1 se muestra una comparativa de las distintas plataformas analizadas.

	<b>Equipo escritorio</b>	<b>Portátil</b>	<b>Tablet</b>	<b>Teléfono móvil</b>	<b>Raspberry Pi 3 B</b>
<b>Precio</b>	ALTO	ALTO	ALTO	MEDIO	BAJO
<b>Tamaño</b>	GRANDE	MEDIANO	MEDIANO	PEQUEÑO	PEQUEÑO
<b>Documentación</b>	VARIADA	VARIADA	ESCASA	VARIADA	ABUNDANTE
<b>Recursos</b>	ALTO	ALTO	ALTO	MEDIO	MEDIO
<b>Soporte</b>	ALTO	MEDIO	BAJO	MEDIO	ALTO
<b>Consumo</b>	ALTO	ALTO	MEDIO	MEDIO	BAJO
<b>Modularidad</b>	ALTA	BAJA	BAJA	BAJA	MEDIA
<b>Integración</b>	MEDIA	ALTA	ALTA	ALTA	ALTA
<b>Conectividad</b>	ALTA	ALTA	BAJA	BAJA	ALTA

Tabla 1. Comparación entre plataformas de hardware.

### Otras plataformas

Se han consultado otras fuentes para buscar alternativas. (10) (11). Existen en el mercado varias opciones interesantes:

- ASUS Tinker Board.
- Banana Pi M3.
- BeagleBone Black
- Intel Joule.
- NanoPC-T3.
- NanoPi Neo Plus 2.
- ODroid XU4.
- Orange Pi Plus 2E.
- Pine A64.
- Pocket Beagle

Estas opciones varían en cuanto a precio, memoria, velocidad de procesador y componentes integrados. No obstante, ninguno de los citados anteriormente tiene tanta documentación y soporte como la Raspberry Pi.

## 2.2. Servicios en la nube

En esta sección se detallan los distintos servicios en la nube analizados que disponen de aplicaciones a las que enviar imágenes de rostros que realicen una

comparación de los rostros con clasificadores entrenados y que finalmente devuelvan los valores e identificadores de los candidatos más relevantes.

Las características deseables para elegir el servicio son:

- Facilidad de uso.
- Documentación madura sobre el servicio.
- Compatibilidad con los lenguajes y tecnologías de este proyecto.
- Soporte de la comunidad.
- Plan de estudiante o plan gratuito durante un periodo de tiempo.

A continuación, se describen los servicios en la nube analizados y se citan sus características principales.

### **Amazon Rekognition**

Amazon Rekognition (12) forma parte del ecosistema AWS, lo cual permite su fácil integración con otras aplicaciones que se desarrollen empleando dicho ecosistema. Dispone de un plan gratuito durante 12 meses con limitaciones. El plan incluye reconocimiento facial con limitación de número de caras a reconocer al día.

El problema de este servicio es que la documentación es bastante limitada y casi no existen ejemplos.

### **Face++**

Además del servicio basado en la nube, Face++ (13) dispone de un servicio *offline* para Android. En principio esta opción es interesante, pero la opción *offline* solo incluye detección de caras y no el reconocimiento de las mismas.

Incluye un plan gratuito, aunque las limitaciones del plan solo permiten realizar una transacción por segundo, lo que en ocasiones puede ralentizar el sistema.

No obstante, al intentar crear un usuario para probar la cuenta gratuita y poder ver las condiciones de uso, la web se quedó bloqueada en todos los intentos.

Adicionalmente, debido al nuevo Reglamento General de Protección de Datos de la UE, Face++ suspende sus servicios dentro la Unión Europea.

### **Kairos**

El servicio Kairos (14) ofrece un valor añadido como son la detección de edad, género, estado emocional y análisis de video en tiempo real.

El plan gratuito ofrece acceso a todas las características con limitaciones. Una de ellas es un máximo de 25 transacciones por minuto y 1500 en total al día. Parece insuficiente para un reconocimiento múltiple de caras y menos adecuado para análisis en tiempo real de video.

### **Google Vision**

El servicio integrado con Google Cloud (15) ofrece muchas posibilidades, pero el reconocimiento facial no es una de ellas. Por tanto, se descarta para este proyecto.

### **IBM Watson**

Aunque explícitamente Watson (16) no tiene un servicio de reconocimiento facial, dispone del servicio Visual Recognition, que permite crear clasificadores específicos para poder realizar el reconocimiento facial.

El plan gratuito incluye solo la clasificación de 250 imágenes al día; sin embargo, el plan de estudiante permite un acceso mayor durante 6 meses.

La documentación es mejorable y tiene pocos ejemplos, pero el material publicado es adecuado.

### **Microsoft Azure**

Azure (17) incluye un servicio de reconocimiento facial con posibilidad de creación de grupos (amigos, trabajo) y opciones de valor añadido como detección de emociones, gafas, género, estimación de la edad, etc.

La documentación es bastante completa pero los ejemplos en Android no son adecuados.

Microsoft ofrece un plan gratuito de 30 días para cuentas nuevas y hasta 12 meses para estudiantes. Existen ciertas limitaciones como el empleo de 10 llamadas por segundo a la API, pero son más que suficientes para el desarrollo de este proyecto.

En la Tabla 2 se muestra una comparativa de los diversos servicios en la nube analizados.

	 Rekognition	 Face++ 旷视 Cognitive Services	 KAIROS	 Google	 IBM Watson	 Microsoft Azure
<b>Nombre</b>	Amazon Rekognition	Face++	Kairos	Google vision	IBM Watson	Microsoft Azure
<b>Servicio online</b>	SI	SI	SI	SI	SI	SI
<b>Servicio offline</b>	NO	SI	NO	NO	NO	NO
<b>Reconocimiento facial exclusivo</b>	SI	SI	SI	NO	NO	SI
<b>SDK</b>	SI	NO	NO	SI	SI	SI
<b>API</b>	SI	SI	SI	SI	SI	SI
<b>Plan estudiante</b>	NO	NO	NO	NO	SI	SI
<b>Plan gratuito</b>	Limitado	Limitado	Limitado	Limitado	Limitado	Limitado
<b>Servicios adicionales (almacén, bases de datos)</b>	SI	SI	SI	SI	SI	SI
<b>Documentación</b>	MEDIA	ESCASA	ESCASA	BUENA	MEDIA	BUENA
<b>Valor añadido (detección de emociones, rasgos, sexo)</b>	SI	SI	SI	NO	NO	SI

Tabla 2. Comparativa entre servicios de reconocimiento facial.

## 2.3. Bibliotecas de visión artificial

En esta sección se evalúan las bibliotecas de funciones que puedan emplearse para la detección facial. Existen varias opciones, pero solo se analizan las dos más extendidas.

### **OpenCV**

OpenCV (18) es una biblioteca *open source* que dispone de algoritmos que pueden ser empleados en proyectos de visión artificial y aprendizaje automático. Su principal atractivo es la gran cantidad de algoritmos muy robustos, revisados y comprobados.

Esta biblioteca además es gratuita, no siendo necesario ningún tipo de suscripción, pero no ofrece un servicio explícito de detección facial, este debe ser desarrollado empleando las funciones que ofrece.

### **Mobile Vision**

Mobile Vision (19) es un subconjunto de ML Kit (20) de Google que incluye APIs específicas para la detección facial, el escaneo de códigos de barras y el reconocimiento de texto. Su uso es gratuito, dispone de una guía de referencia bastante completa y existen muchos ejemplos para entender su funcionamiento.

## **2.4. Lenguajes de programación**

Es difícil analizar todos los lenguajes de programación que podrían emplearse debido a la gran variedad que existe actualmente. La elección del lenguaje elegido en el desarrollo de este proyecto se ha realizado tras la elección del resto de herramientas, con el fin de reducir el número de opciones a considerar.

No obstante, las características que se buscan en los lenguajes a emplear son las siguientes:

- Lenguaje de propósito general.
- Documentación abundante.
- Existencia de ejemplos.
- Sintaxis que permita la fácil lectura del código.
- Existencia de bibliotecas para crear una interfaz gráfica.
- Multiplataforma.
- Sintaxis reconocida por generadores de documentación.

## 2.5. Elección de herramientas

A continuación, se detallan las herramientas finalmente elegidas y los motivos que han llevado a dicha elección.

### 2.5.1. *Hardware*

Tras el análisis de las distintas opciones, dos plataformas han sido elegidas: Raspberry Pi 3 y Android Phone.

#### Raspberry Pi 3

Se ha escogido esta plataforma por estar a mitad de camino entre el tamaño y consumo energético de un móvil y los recursos y versatilidad de un ordenador. Además del hecho de que existen gran número de proyectos que incluyen la Raspberry Pi.

El pequeño tamaño de la Raspberry Pi facilita su acoplamiento a un robot de telepresencia u otro tipo de robots, también permite que se pueda usar como equipo de escritorio o bien emplear de forma móvil.

La gran cantidad de documentación sobre la Raspberry Pi y su precio son una ventaja adicional apreciada.

Para la captura de imágenes se necesita una cámara digital. Se puede conectar una *webcam* mediante conexión USB o bien emplear el conector de cámara integrado en la placa base.

El uso de una *webcam* tiene como ventajas:

- existencia de *webcams* de alta definición y gran número de *megapixels*.
- incorporación de *flash* en algunos modelos.
- trípode u otro tipo de soporte incorporado.

Sus desventajas son:

- necesidad de drivers específicos que no siempre existen para Linux.
- necesidad de módulos específicos para incorporar su uso en lenguajes de programacion.

La cámara de la propia Raspberry Pi tiene como ventajas:

- conector específico dedicado en la propia placa.
- tamaño reducido.
- Raspbian incorpora drivers de serie.
- existen módulos para su fácil uso en programación.

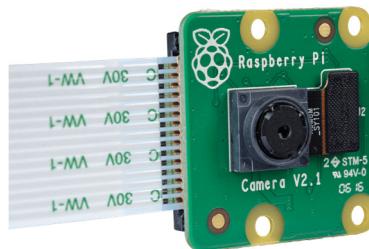


Figura 4. Cámara de Raspberry Pi (Fuente reichelt.com).

A la vista de las ventajas y desventajas de la *webcam* y la cámara de la Raspberry Pi, se decidió emplear la cámara propia de Raspberry Pi modelo Module 2 (21) cuyas características más relevantes son sus 8 *megapixels* y video en 1080p.

### Android Phone

Se ha elegido este sistema como plataforma donde instalar la aplicación de entrenamiento debido a que se buscaba que el dispositivo estuviese altamente extendido y al alcance de los usuarios. Por tanto, una vez generada la aplicación se puede instalar en diversos dispositivos realizando la misma función.

Otro de los motivos es que el IDE Android Studio permite crear fácilmente aplicaciones Android y existe gran cantidad de documentación al respecto (22).

Se ha empleado un Samsung Galaxy S3 para el desarrollo de la aplicación y la realización de pruebas.

### 2.5.2. Servicio en la nube

El servicio elegido finalmente es el de Microsoft Azure, aunque no fue la primera elección. A continuación, se describen los motivos que provocaron el cambio de Watson a Azure.

Al inicio del proyecto, se comenzó a trabajar con Azure y con Watson. Para verificar qué servicio ofrecía el mejor porcentaje de confianza en el reconocimiento facial, se crearon y aplicaron las pruebas “Bill Gates vs. Ben Linus” que se describen en el Capítulo 4.

Con Watson se obtuvieron unos porcentajes de confianza altos (hasta 82 %). Mientras que con Azure se obtuvieron unos porcentajes de confianza buenos, pero no tan buenos como los de Watson (hasta un 73 %). Azure se abandonó y se comenzaron a desarrollar las aplicaciones empleando Watson como servicio en la nube.

Aunque Watson no tiene un servicio de reconocimiento facial explícito, si permite la creación de clasificadores específicos en los que se pueden almacenar fotos, entrenar el sistema y, posteriormente, comparar una foto con cada clasificador y obtener un porcentaje de confianza. Las pruebas iniciales y facilidad de uso de la API cumplían con las expectativas para este proyecto.

El hecho de no tener un sistema específico de reconocimiento facial parecía no ser un problema al poder crear clasificadores individuales y cada uno ligarlos a una persona. Además, cada clasificador puede alimentarse también con ejemplos negativos. Se planteó que la primera foto de cada persona se subiese como ejemplo positivo al clasificador y se añadiese como ejemplo negativo a los demás clasificadores.

Tres motivos hicieron que finalmente se descartase el sistema Watson como núcleo del proyecto:

- El requisito mínimo de fotos para un clasificador es de diez. No siempre es posible disponer de ese número de imágenes. Por ejemplo, en el momento de entrenar el sistema mediante el móvil es posible que la persona a la que se fotografía no tenga la paciencia ni ganas de esperar a que se le realicen diez fotos. Emplear la misma foto diez veces no es una buena idea, no permite un buen entrenamiento del clasificador y baja el porcentaje de confianza obtenido.
- Durante el periodo de prueba de Watson, el sistema migró el tipo de cuenta que se empleaba a cuenta “Lite” con la funcionalidad muy limitada. Los

motivos de esta migración se desconocen puesto que la cuenta tenía seis meses de uso de estudiante. Durante esta transición aparentemente IBM estaba cambiando partes de su sistema y esto pudo estar relacionado con la migración de cuentas.

- El tercer motivo fue que el clasificador no acepta nuevas fotografías una vez creado el clasificador y subidas las imágenes iniciales de entrenamiento. Esto crea un problema, puesto que cada vez que se quiera añadir una fotografía al clasificador específico de una persona se tendría que eliminar completamente su clasificador, obtener las imágenes iniciales, añadir las nuevas, recuperar los ejemplos negativos, añadirlos también y subir toda la información de nuevo, lo que supone una sobrecarga de transacciones y mayor lentitud global del sistema.

Si bien los dos primeros motivos podrían remediararse, el tercero no tiene una solución factible. Por tanto, a mediados de febrero se comenzó a probar en mayor medida el sistema Azure y a principios de marzo se abandonó completamente el sistema Watson. Esto provocó un cambio en el planteamiento del proyecto y tener que comenzar a codificar de nuevo las aplicaciones.

### **Características de Azure**

Tras darse de alta en el servicio (como estudiante en el caso de este proyecto) el panel principal de Azure permite elegir un servicio para comenzar a trabajar. Para este proyecto se eligió el servicio QeQ Cognitive Services (Figura 5) que permite trabajar con la API denominada Face API (23). Una vez dado de alta, el servicio Azure entrega una clave de suscripción de 32 caracteres con la que se deben realizar todas las consultas y llamadas a la API.

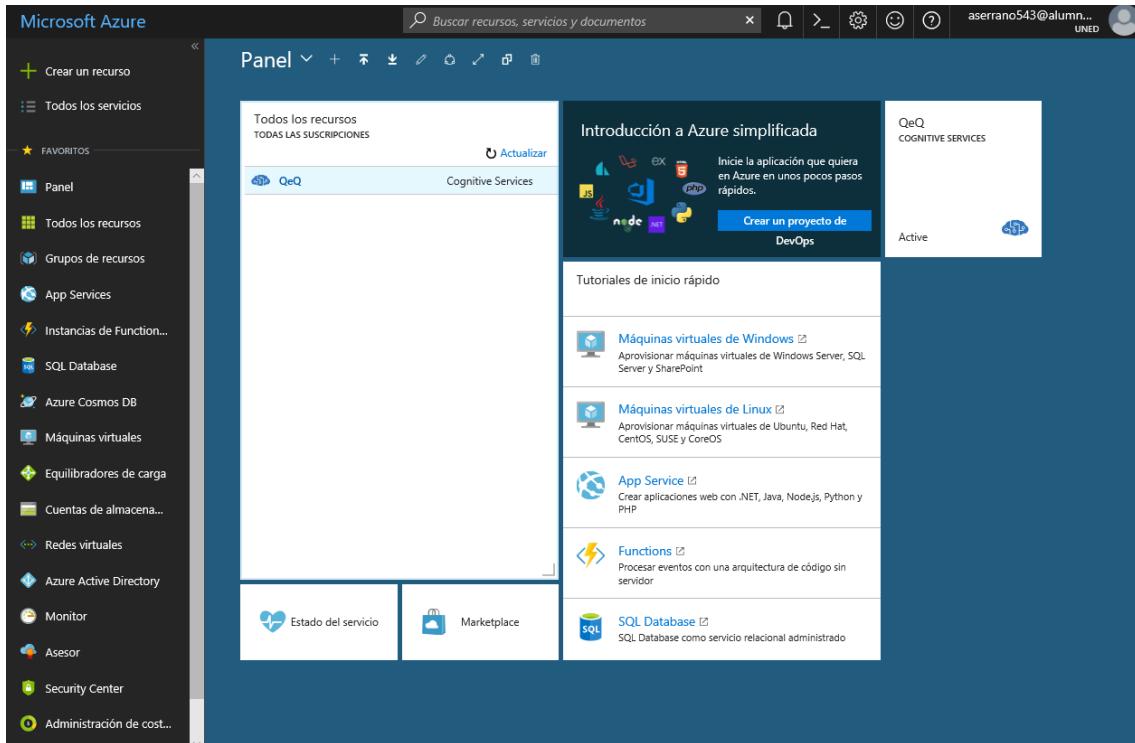


Figura 5. Panel de Azure al darse de alta en el servicio QeQ.

Para el reconocimiento facial, Azure dispone del servicio Cognitive Services. Este servicio dispone de un API bien documentado llamado Face API y de varios SDK para Android, iOS, Python y Windows de libre acceso disponibles en Github.

La base de la API es la construcción de una dirección URL que combina varias partes:

1. Un método HTTP: POST, GET o DELETE. Se usa uno u otro dependiendo de la función del API que se quiera emplear.

2. Una dirección base que indica el servidor y servicio que se quiere acceder:

<https://northeurope.api.cognitive.microsoft.com/face/v1.0/>

3. Una parte adicional que especifica qué parte del servicio se solicita, grupo de personas, persona, etc:

[/persongroups/{personGroupId}/persons/{personId}/persistedFaces\[?userData\]\[&targetFace\]](/persongroups/{personGroupId}/persons/{personId}/persistedFaces[?userData][&targetFace])

4. Una cabecera con información sobre el contenido del cuerpo del mensaje (generalmente formato JSON) y la clave de subscripción:

```
headers = {'Content-Type': 'application/json', 'Ocp-Apim-Subscription-Key': '{subscription key}'}
```

5. Un cuerpo del mensaje con distinta información que irá con la petición HTTP:
- parámetros (ej, umbral de confianza).
  - información (en formato JSON).
  - datos (imagen).

La API se compone de varias partes y tiene muchas aplicaciones como la identificación de grandes grupos de personas. Dado el objetivo de este proyecto, solo algunas partes de la API han sido empleadas, siendo estas Face, PersonGroup y Person, de las que solo se describen los métodos generales que se han empleado.

En líneas generales se pueden crear varios grupos de personas (`PersonGroup`) y cada uno de ellos es capaz de almacenar personas (`Person`). Cada `Person` a su vez puede almacenar caras (`Face`) y de esta manera los datos quedan fácilmente almacenados (Figura 6).



Figura 6. Ejemplo de organización de entidades (Fuente Microsoft).

Cada entidad tiene su identificador único que permite recuperar rápidamente la información pertinente. Los tres que se emplean son:

- `faceId`: identifica una cara.
- `personId`: identifica una persona.
- `personGroupId`: identifica un grupo.

Se describen las tres partes relevantes de la API: `Face`, `PersonGroup` y `Person`.

### **Face**

El método que se emplea es `Identify`. Este método identifica caras, para ello se introducen como parámetros de entrada:

- un `faceId` (pueden ser varios) con la cara que queremos identificar (esta cara tiene que estar en el sistema).
- un `groupPersonId` para identificar el grupo en el que se comparara la cara. Azure permite comparar una cara con un grupo entrenado por llamada.
- un número máximo de candidatos de retorno. Este parámetro es opcional, por defecto su valor es 10.
- un umbral de confianza que sirve para que no devuelva resultados cuyo porcentaje de confianza esté por debajo del valor indicado. Este parámetro es opcional.

El retorno de `Identify` son datos en formato JSON indicando los `personID` de las personas a las cuales podría pertenecer la cara objeto de identificación y la confianza de estos.

### **Person**

Dispone de métodos para agregar, recuperar y borrar personas. Se emplean principalmente tres de ellos:

`Add Face` se encarga de añadir las caras (siempre de una en una) a la persona que se le indique. Para ello, como parámetro de entrada se introduce el `personId` de la persona objetivo, el `groupPersonId` donde esté situada la persona y una URL con la

imagen que contiene la cara. Si la llamada es exitosa, se obtiene el `faceId` correspondiente.

`Create` es un método sencillo que, aportando un `groupPersonId` y un nombre para la persona que se desea crear, crea una persona y devuelve su `personId`.

`Get` se emplea para conseguir los datos de una persona a partir de su `personId`.

### **PersonGroup**

De forma análoga a `Person`, dispone de métodos para agregar, recuperar y borrar. Dispone de un método `create` que tan solo aportando el nombre del grupo, éste es creado y devuelve el `groupPersonId` correspondiente.

Existen dos métodos importantes que se emplean:

`Train`: tras subir fotos a una persona dentro de un grupo, es necesario entrenar el clasificador. Por una parte, se entrena el clasificador individual y por otra parte se entrena como grupo. Esto sirve por si se desea crear grupos con características similares para otro tipo de aplicaciones, como clasificar gente joven y gente mayor o gente con el pelo corto y pelo largo, etc.

Para este proyecto se necesita entrenar los clasificadores individuales, por tanto, cada vez que se añade una cara a una persona es necesario invocar el entrenamiento.

Como parámetro de entrada se necesita el `groupPersonId` del grupo objetivo y si el entrenamiento ha sido correcto se recibe un mensaje JSON vacío.

`Get training status`: el entrenamiento es más complejo cuantos más datos se tengan, por tanto, a medida que crece el sistema el tiempo de espera para la finalización será mayor. Es prudente usar este método para comprobar si se ha terminado el entrenamiento y es seguro continuar usando el sistema.

El parámetro de entrada es el `groupPersonId` del grupo objetivo de verificación y se obtienen cuatro cadenas de texto con información sobre el entrenamiento:

- estado del entrenamiento (no comenzado, ejecutando, terminado o fallido).
- fecha y hora de creación del grupo.
- fecha y hora de última acción realizada sobre el grupo.
- mensajes de error, si los hubiera.

Con los objetos y métodos vistos anteriormente es suficiente en principio para poder crear una implementación propia, como se describe en el capítulo 3.

### 2.5.3. Bibliotecas de visión artificial

#### OpenCV

Esta biblioteca dispone de varias funciones de interés. Con el fin de evitar la sobrecarga y conseguir el envío eficaz de imágenes, se ha empleado OpenCV en una parte del sistema.

Este proyecto pretende ser móvil, para ello las comunicaciones a Internet son inalámbricas (Wifi o uso de datos de telefonía móvil), por tanto, es importante que los envíos sean ligeros.

El empleo de OpenCV puede ayudar a resolver una serie de inconvenientes causados al tomar fotos, ya que estas, al ser de alta resolución, pueden ocupar unos cuantos *megabytes*. Los inconvenientes de emplear una imagen de gran tamaño son:

- El envío de datos mediante medios inalámbricos generalmente tiene menor velocidad que si se emplean conexiones físicas, lo que puede suponer una ralentización del sistema.
- En el caso del uso de redes de telefonía móvil, se añade el coste por *megabyte* enviado. El sistema no resultaría tan económico como se pretende.
- Al recibir imágenes grandes, el servicio de reconocimiento tiene que detectar primero las caras, separarlas y procesarlas. Esto crea más tiempo de procesamiento en el servicio, disminuyendo el rendimiento global del proyecto.

Por tanto, la solución es el envío de imágenes que:

- contengan la información justa y necesaria, es decir, solo la cara.
- solo contengan una cara, disminuyendo el tiempo de procesamiento.
- sean de menor tamaño.

Para implementar esta solución se ha desarrollado una función específica que se describe con detalle en el Capítulo 3 dedicado a la implementación. Esta función recibe la imagen capturada, recorta las caras relevantes, las almacena para su envío individual al servicio Azure y posterior identificación. La implementación de esta función de recorte de caras es ligera, rápida y no demanda demasiados recursos.

La función de recorte aprovecha el módulo `objdetect` (*Object Detection*) de OpenCV, que contiene la clase `CascadeClassifier` (24), a la que se accede para realizar la función descrita anteriormente. Esta clase permite la detección de caras basada en la extracción de características con filtros de base Haar y clasificadores en cascada (25) (26).

### **Mobile Vision**

La elección de Mobile Vision está motivada por los inconvenientes de las imágenes de gran tamaño descritas en la sección anterior y la necesidad de solucionar dichos inconvenientes en la aplicación de entrenamiento que se ha desarrollado para Android Phone. Existe una versión de OpenCV para Android con la que pudo haberse implementado la misma solución, sin embargo, Mobile Vision está diseñado para trabajar en Android y dispone de funciones de fácil uso e implementadas para facilitar la detección facial.

## **2.6. Elección del lenguaje**

### **Python**

Para el desarrollo de la biblioteca de funciones y las aplicaciones que la emplean, los posibles lenguajes de programación quedan acotados a dos opciones principalmente (Python y C++), debido a la anterior elección de Raspberry Pi, Azure y OpenCV para este proyecto.

Ambos lenguajes cumplen las características deseadas para este proyecto citadas anteriormente en este capítulo. Sin embargo, se ha escogido Python por varios motivos:

- Su sintaxis permite una fácil lectura y escritura de código.
- Está bien integrado en Raspberry Pi, es su lenguaje “oficial”.

- Existen muchas bibliotecas bien documentadas para Python que permiten un fácil uso de las características de la Raspberry Pi (cámara, GPIO, etc).
- El SDK de Azure para Python es muy sencillo de usar.
- OpenCV para Python está muy bien documentado y existen gran cantidad de ejemplos.
- La biblioteca gráfica Qt (27) tiene un *binding* para Python (PyQt) (28) que permite crear interfaces gráficos con muy pocas líneas de código.

## Java

Al elegir Android como plataforma para el desarrollo de la aplicación de entrenamiento, se pueden emplear varios lenguajes para la programación de esta aplicación (29), pero se eligió Java por varios motivos:

- Java es uno de los lenguajes oficiales de Android.
- Android dispone de un IDE gratuito llamado Android Studio que permite la programación de aplicaciones para Android empleando el lenguaje Java con muchas funciones integradas.
- Aunque existen otros lenguajes como Kotlin, la mayor parte de la documentación sobre Android sigue siendo sobre Java.
- De la misma forma, la mayor parte de ejemplos y bibliotecas disponibles para Android están escritas en Java

Para el diseño estético de la aplicación también se emplea XML. Si bien no es un lenguaje de programación como tal, es necesario entenderlo para hacer un diseño correcto de la aplicación.

## 3. Diseño y desarrollo

En este capítulo se describen los componentes *software* desarrollados.

En este proyecto se han desarrollado una biblioteca de funciones (FCModule) y tres aplicaciones que la emplean (FaceBT, FacePi, FaceRecon). Todos estos componentes se instalarán en la Raspberry Pi. Adicionalmente se ha desarrollado una aplicación para Android (FacePal).

Por cada uno de estos componentes de *software*, se describe cómo se estructura su código, cuáles son sus funciones principales y cómo se emplean. Para mayor detalle de atributos y funciones de cada componente *software*, se ha incluido un manual de clases en la documentación del proyecto. Los manuales de instalación y de usuario de las aplicaciones se encuentran en los Anexos B y C de esta memoria.

### 3.1. FCModule

Esta biblioteca contiene las funciones de detección facial, recorte de caras, identificación, y etiquetado de las caras en una imagen. Adicionalmente, contiene funciones de gestión de los datos almacenados en Azure.

Para facilitar la comprensibilidad del sistema y el mantenimiento, se divide el código en módulos/clases agrupando funciones similares. En la Figura 7Figura 9 se puede observar un esquema de las clases contenidas en el módulo.

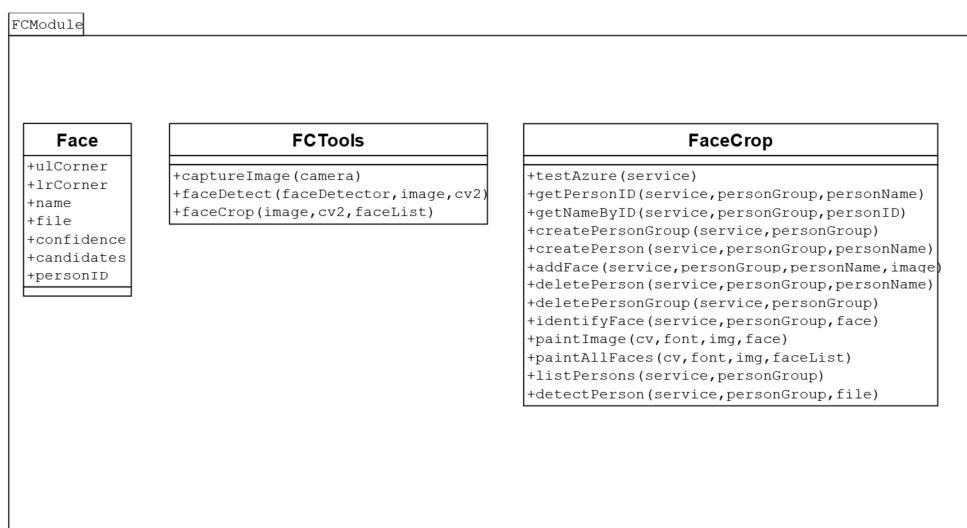


Figura 7. Diagrama de clases de FCModule.

### 3.1.1. Face

Esta clase define la estructura de datos que contiene la información de cada cara. Su constructor requiere que se le pasen dos parámetros, `ulCorner` y `lrCorner`, que corresponden con las coordenadas de las esquinas del rectángulo que contiene la cara objetivo, como se muestra en la Figura 8.

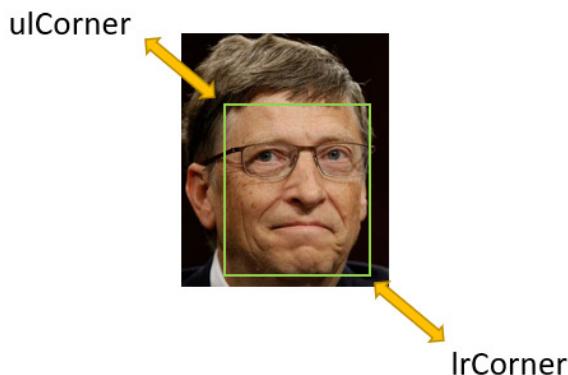


Figura 8. Ejemplo de esquinas del rectángulo.

Sus atributos son:

`ulCorner`: contiene las coordenadas X, Y de la esquina superior izquierda (*upper left corner*).

`lrCorner`: contiene las coordenadas X, Y de la esquina inferior derecha (*lower right corner*).

`name`: contiene el nombre del mejor candidato tras su identificación.

`file`: contiene el nombre del fichero que almacena el *bitmap* con la cara recortada, que es enviado a Azure.

`confidence`: el nivel de confianza del mejor candidato.

`candidates`: lista de todos los candidatos ordenados por nivel de confianza.

`personID`: identificador de Azure.

El objetivo principal de esta clase es almacenar las coordenadas del rectángulo según se van detectando rostros en una imagen, permitiendo la localización de los rostros en la imagen original. Los objetos derivados de esta clase posteriormente recibirán el nombre de archivo que albergará el rostro y por último recibirán el nombre

de la persona a la que pertenece el rostro o la etiqueta “Desconocido” si no fuesen reconocidos.

### 3.1.2. FaceCrop

Esta clase contiene las funciones de gestión de las imágenes relacionadas con el recorte de caras. A continuación, se describen sus funciones internas:

- `captureImage (camera) :`

Captura una imagen desde la cámara de la Raspberry Pi.

Parámetros:

- `camera`: objeto que contiene la cámara de la Raspberry Pi.

Retorno:

- `img`: un objeto `PiRGBArray` que contiene la información cruda de la imagen.

- `faceDetect (faceDetector, image, cv2) :`

Detecta caras en una imagen.

Parámetros:

- `faceDetector`: archivo que contiene la información sobre el filtro de Haar.
- `image`: objeto `PiRGBArray` que contiene la imagen.
- `cv2`: objeto con las funciones de OpenCV.

Retorno:

- `faceList`: un *array* de objetos `Face` detectados en la imagen.

- `faceCrop (image, cv2, faceList) :`

Recorta caras de una imagen guardando cada una en un archivo. No devuelve ningún tipo de dato.

Parámetros:

- `image`: objeto `PiRGBArray` que contiene la imagen.
- `cv2`: objeto con las funciones de OpenCV.
- `faceList`: un *array* de objetos `Face` detectados en la imagen.

### 3.1.3. FCTools

Este módulo contiene varias funciones para el entrenamiento inicial del sistema y gestión de los datos almacenados en Azure. Solo se describirán en detalle las funciones más relevantes, del resto solo se hará una descripción breve, para mayor detalle consultar el manual de clases (Anexo A).

`identifyFace(service, groupName, face) :`

Esta función conecta con Azure, identifica la cara y pasa el nombre identificado al objeto `Face` correspondiente. No devuelve ningún tipo de dato, solo modifica sobre la marcha cada objeto `Face` que esté contenido en el *array* de caras.

Parámetros:

- `service`: el objeto que contiene las funciones para conectar con Azure.
- `groupName`: nombre del grupo de identificación.
- `face`: objeto de la clase `Face`.

`paintImage(cv, font, img, face) :`

Esta función toma las coordenadas de un objeto `Face` y dibuja un rectángulo que enmarca la cara, además de escribir el nombre correspondiente a la cara.

Parámetros:

- `cv`: objeto de OpenCV con las funciones para la gestión de imágenes.
- `font`: fuente para escribir el texto sobre las imágenes.
- `img`: imagen entera.
- `face`: objeto de la clase `Face`.

`getNameByID(service, personGroup, personID) :`

Esta función cubre una necesidad no incluida en Azure, que es la relación entre el identificador asignado y el nombre real de la persona. Al identificar una cara, Azure devuelve el identificador del candidato, pero no su nombre. Para facilitar la identificación, esta función devuelve el nombre asignado por el usuario al candidato.

**Parámetros:**

- `service`: el objeto que contiene las funciones para conectar con Azure.
- `personGroup`: nombre del grupo de identificación.
- `personID`: id de una persona.

**Retorno:**

- `name`: nombre real del candidato.

A continuación, se mencionan el resto de las funciones, para más detalles consultar el manual de clases adjunto a este proyecto.

- `testAzure(service)` :  
Test de funcionamiento.
- `getPersonID(service, personGroup, personName)` :  
Obtiene el ID de una persona mediante el nombre real.
- `createPersonGroup(service, personGroup)` :  
Crea un grupo de personas.
- `createPerson(service, personGroup, personName)` :  
Crea una persona.
- `addFace(service, personGroup, personName, image)` :  
Añade una cara a una persona ya dada de alta.
- `deletePerson(service, personGroup, name)`  
Elimina una persona y todas las caras asociadas
- `deletePersonGroup(service, personGroup)` :  
Elimina un grupo de personas y toda la información asociada.
- `listPersons(service, personGroup)` :  
Devuelve información sobre las personas de un grupo.
- `detectPerson(service, personGroup, file)` :  
Identifica una persona desde una imagen de archivo.

### 3.2. FaceRecon

FaceRecon es un ejemplo de uso de FCModule. Su finalidad es demostrar la invocación de funciones de FCModule para constituir una herramienta de detección e identificación facial.

La Figura 9 muestra cómo FaceRecon usa las distintas clases de FCModule.

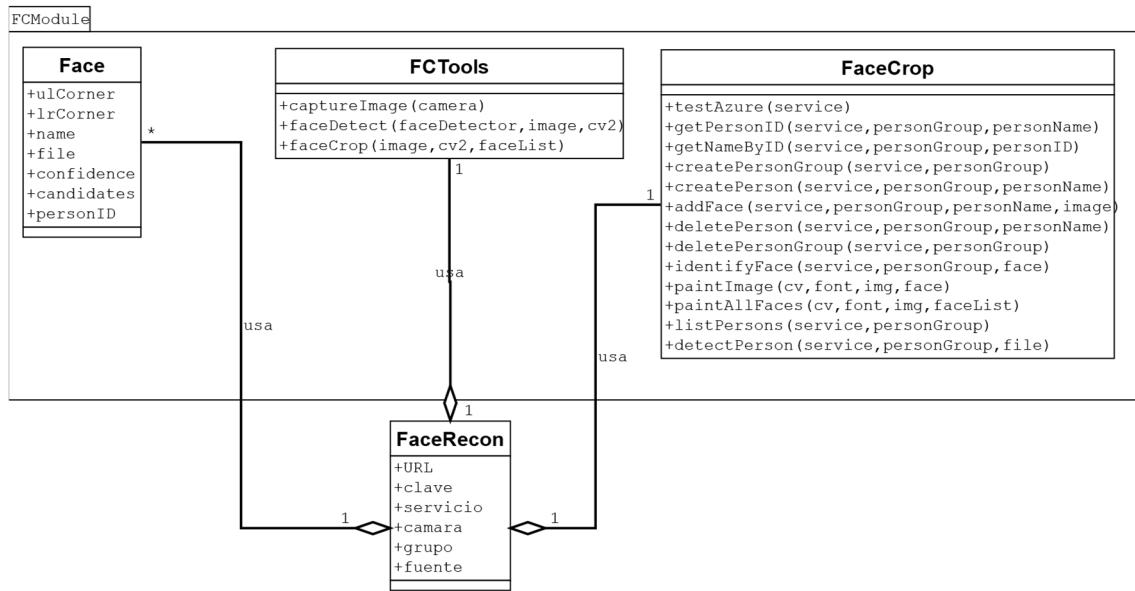


Figura 9. Diagrama de clases de FaceRecon.

FaceRecon define los siguientes atributos para el uso de FCModule:

- **sKey**: contiene la clave de suscripción de Azure.
- **server**: indica el valor que se añade a la URL base para la conexión con Azure.
- **faceDetector**: el archivo XML que contiene la información de filtros a emplear. Se encuentra en la carpeta 'utils'. Estos filtros se han obtenido del paquete OpenCV y se almacenan en este directorio por comodidad.
- **groupName**: el nombre del grupo en Azure que contiene las personas cuyos datos ya han sido subidos.
- **waitTime**: tiempo de espera entre identificaciones, para evitar pasar el límite de la versión gratuita.

La descripción genérica de la aplicación se muestra en la Figura 10. Al inicio el sistema toma una imagen; si no se detectan caras, vuelve a tomar otra imagen hasta encontrar al menos una cara. Una vez detectado uno o más rostros, se individualizan en recortes que se enviados a Azure para obtener su identificador. Si no se dispone de identificador, se etiqueta el rostro como “Desconocido”, en caso contrario, se etiqueta con el nombre asignado a ese identificador.

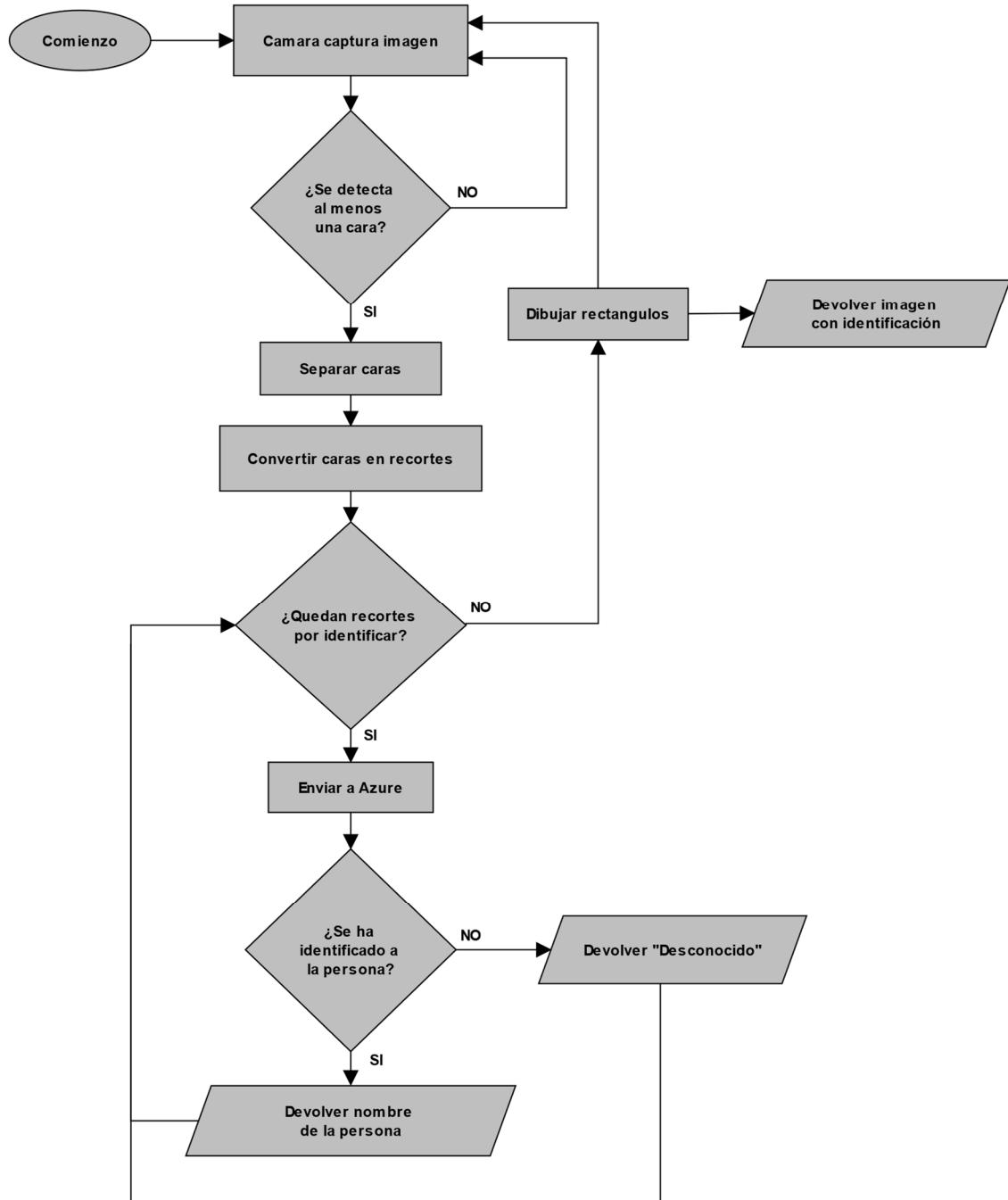


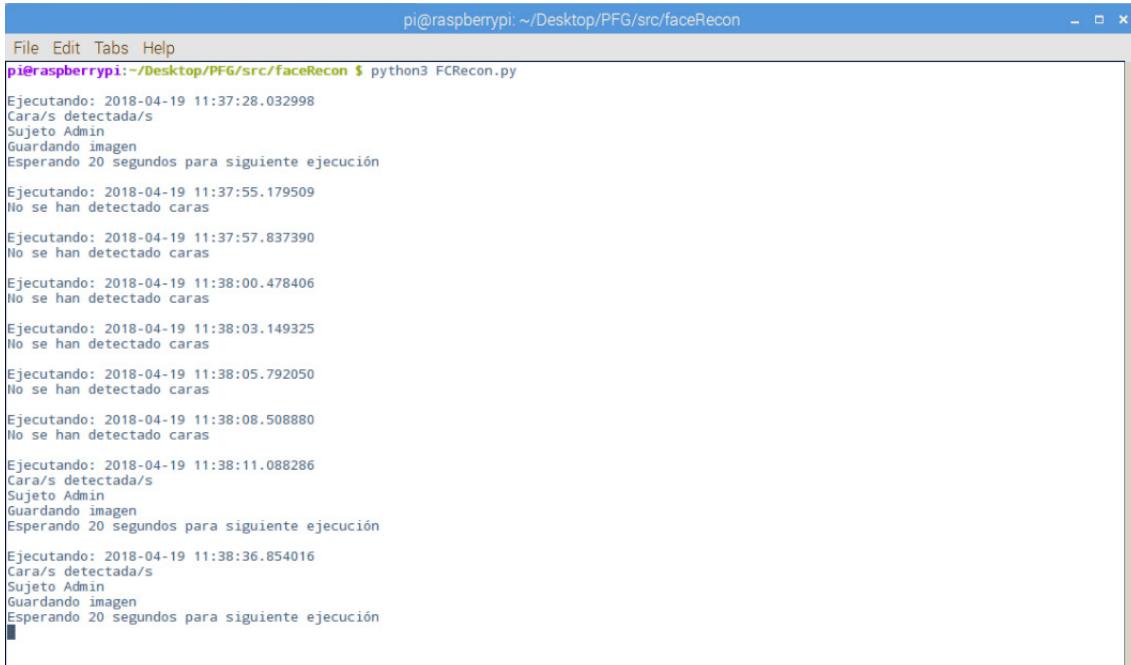
Figura 10. Descripción de FaceRecon.

La ejecución de FaceRecon (Figura 11) es un bucle que se ejecuta siempre, salvo que el usuario ejecute Control+C o el sistema se apague. El bucle realiza las siguientes acciones como se indica en el siguiente pseudocódigo:

- Imagen //variable que almacena una imagen
- Caras //array que almacena caras detectadas
- while (true)
  - o Imagen = capturar imagen

- o Caras.insertar (Imagen) //detecta, recortar
- o por cara en Caras
  - identificar cara
  - pintar cara en Imagen
- guardar Imagen

Las imágenes creadas por FaceRecon son almacenadas en el directorio img, cada una con el nombre según el siguiente patrón: image\_{marca\_temporal}.jpg.



A terminal window titled "pi@raspberrypi: ~/Desktop/PFG/src/faceRecon \$ python3 FCRecon.py". The window shows the command being run and its output. The output includes timestamped logs of face detection attempts, such as "Ejecutando: 2018-04-19 11:37:28.032998 Cara/s detectada/s Sujeto Admin Guardando imagen Esperando 20 segundos para siguiente ejecución". It also shows instances where no faces were detected, like "No se han detectado caras". The process repeats every 20 seconds.

```
File Edit Tabs Help
pi@raspberrypi:~/Desktop/PFG/src/faceRecon $ python3 FCRecon.py
Ejecutando: 2018-04-19 11:37:28.032998
Cara/s detectada/s
Sujeto Admin
Guardando imagen
Esperando 20 segundos para siguiente ejecución
Ejecutando: 2018-04-19 11:37:55.179509
No se han detectado caras
Ejecutando: 2018-04-19 11:37:57.837390
No se han detectado caras
Ejecutando: 2018-04-19 11:38:00.478406
No se han detectado caras
Ejecutando: 2018-04-19 11:38:03.149325
No se han detectado caras
Ejecutando: 2018-04-19 11:38:05.792050
No se han detectado caras
Ejecutando: 2018-04-19 11:38:08.508880
No se han detectado caras
Ejecutando: 2018-04-19 11:38:11.088286
Cara/s detectada/s
Sujeto Admin
Guardando imagen
Esperando 20 segundos para siguiente ejecución
Ejecutando: 2018-04-19 11:38:36.854016
Cara/s detectada/s
Sujeto Admin
Guardando imagen
Esperando 20 segundos para siguiente ejecución
```

Figura 11. Ejecución de FaceRecon.

FaceRecon ha servido como base para la realización de las dos aplicaciones que se describen a continuación, FacePi y FaceBT.

### 3.3. FacePi

Es una aplicación de escritorio con interfaz gráfica que permite al usuario acceder a las funciones de FCModule de manera sencilla e interactiva, como se muestra en la Figura 12.

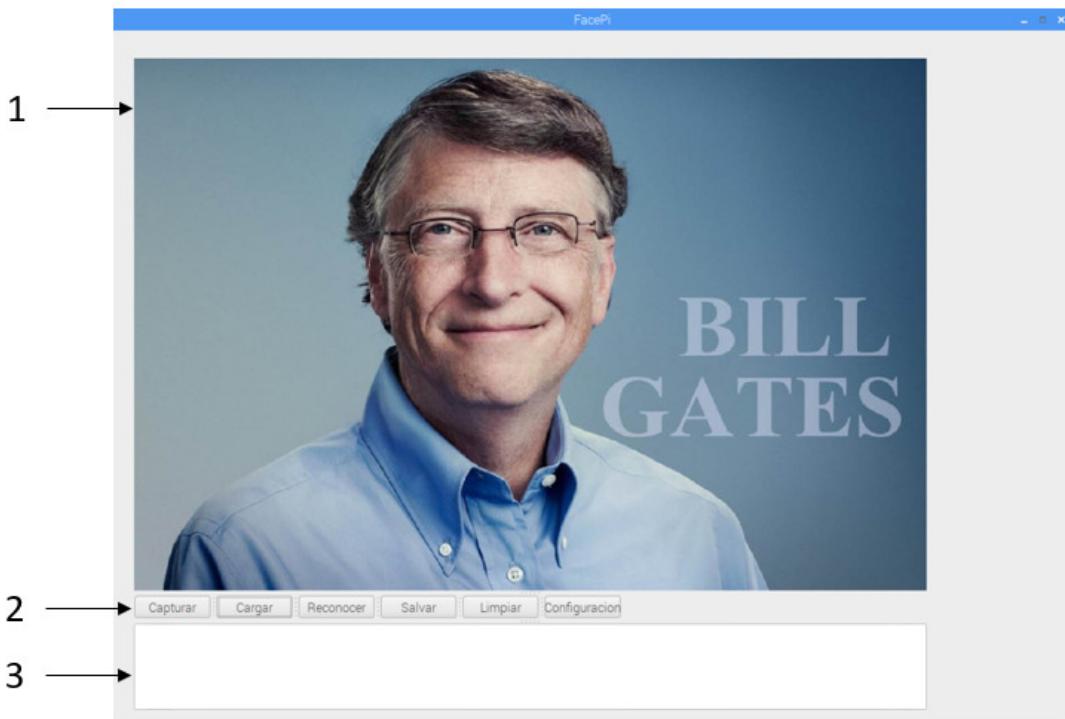


Figura 12. Pantalla principal de FacePi.

Las funciones de esta aplicación son las siguientes:

- identificación y etiquetado de caras, tanto de imágenes almacenadas como de capturas instantáneas mediante la cámara de la Raspberry Pi.
- salvar las imágenes identificadas y etiquetadas.
- mostrar por consola de texto los datos de las caras identificadas (nombre, confianza, otros candidatos).

En la Figura 12 se identifican las tres áreas de la aplicación:

- 1 Área de visualización: muestra la imagen capturada o elegida por el usuario.
- 2 Botones:
  - Capturar: captura una imagen mediante la cámara de la Raspberry Pi.
  - Cargar: busca y carga una imagen almacenada.
  - Reconocer: ejecuta la detección de caras, el envío a Azure para obtener el identificador y el etiquetado de las mismas.
  - Salvar: guarda la imagen en un archivo.
  - Limpiar: limpia la consola de texto.

- Configuración: permite la modificación de los parámetros de la aplicación:
    - Nombre del servidor.
    - Clave de suscripción a Azure.
    - Nombre del grupo de personas.
- 3 Consola de texto: muestra mensajes de texto del funcionamiento de la aplicación y de los resultados del reconocimiento.

### 3.4. FaceBT

Esta aplicación se ejecuta siempre al inicio de la Raspberry Pi en segundo plano y permite que otros dispositivos con Bluetooth puedan conectarse a la Raspberry Pi y puedan emplear las funciones de reconocimiento facial de FCModule.

Al estar siempre ejecutándose en segundo plano, le da a la Raspberry Pi de este proyecto la capacidad de ser portátil, solo necesitando una batería externa y sin requerir ninguna intervención adicional por parte del usuario. Al encender la Raspberry Pi, queda a la espera de conexión de otro dispositivo. Una vez se ha realizado el emparejamiento, se envían los comandos mediante un terminal de puerto de serie Bluetooth (como Bluetooth Serial Terminal para Windows o Serial Bluetooth Terminal para Android) y la Raspberry Pi devuelve los resultados de ejecutar los comandos, como se ve en la Figura 13.

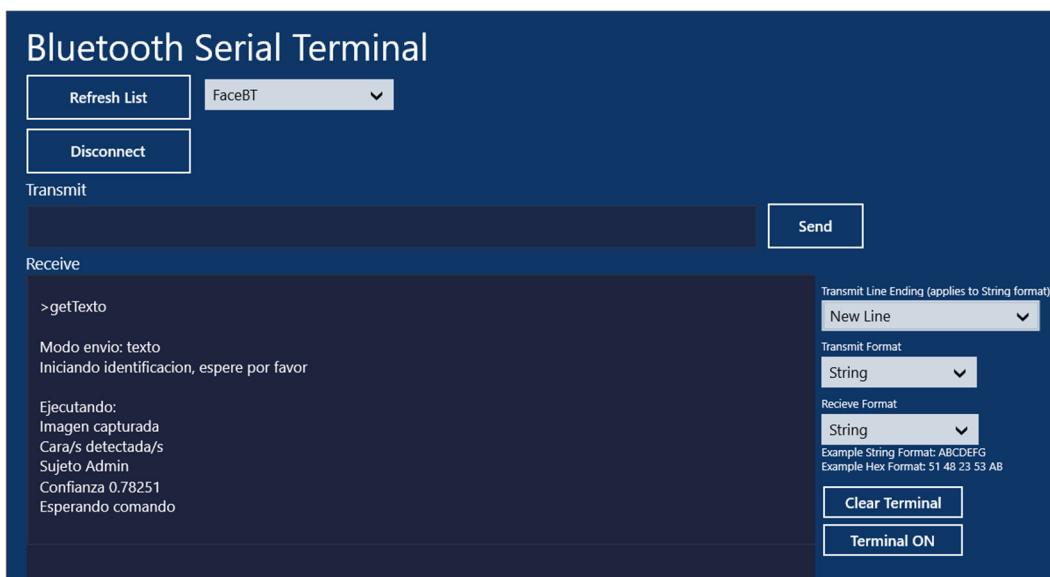


Figura 13. Ejemplo de conexión con FaceBT.

La aplicación desarrollada, aprovecha el protocolo Bluetooth RFCOMM (26) para el envío y recepción de datos.

Los comandos que acepta FaceBT son los siguientes:

- `getTexto`: devuelve información en forma de texto de la imagen capturada y de los rostros identificados.
- `getJSON`: devuelve un archivo en formato JSON con la información de la imagen capturada y de los rostros identificados. El archivo JSON está encapsulado entre los caracteres <<< al inicio del archivo y >>> al final del archivo, debido a que los archivos JSON comienzan con el carácter { y un inicio de mensaje con este carácter provoca la no transmisión del mensaje.
- `getImage`: devuelve la imagen capturada con los rostros identificados como *array de bytes*.
- `apagar`: cierra la aplicación y apaga la Raspberry Pi.

## 3.5. FacePal

Finalmente, se ha desarrollado una aplicación móvil totalmente independiente de FCModule, para su uso en dispositivos Android. Una buena descripción es la de entrenador de bolsillo. Se instala en un teléfono móvil y permite añadir nuevas personas o caras a personas conocidas de una manera cómoda. Incluye también la opción de reconocimiento de caras, independiente de la Raspberry Pi.

### 3.5.1. Consideraciones previas

#### Sobre las clases:

Las aplicaciones de Android están divididas en multitud de componentes (archivos con valores, diseños, clases, manifiestos, imágenes, etc). Queda fuera del alcance del proyecto explicar cada una de ellas, se entra en detalle en las clases más relevantes y sobre todo aquellas que están relacionadas directamente con el reconocimiento facial.

## Sobre la cámara

En principio FacePal tomaría imágenes empleando la cámara de fotos de los propios dispositivos Android. Debido a la variedad de versiones de Android y cambios en el API de la cámara, no se ha logrado una versión estable que funcione en varios dispositivos Android de la misma forma. Por ejemplo, en el dispositivo de pruebas principal (teléfono Samsung Galaxy S3) la cámara siempre toma las fotos con 90º de giro. Si la foto resultado está girada, el reconocimiento no funciona correctamente.

La excesiva complejidad de emplear la cámara junto a la aplicación resultó en un cambio del diseño original del programa. Ahora la aplicación busca fotos en la galería de imágenes del dispositivo. De esta manera se ganan tres ventajas:

- se puede tomar la foto con la aplicación preferida del usuario, recortar, girar o realizar las modificaciones que crea convenientes.
- las fotos guardadas pueden emplearse para el entrenamiento sin necesidad de que el sujeto esté delante del dispositivo esperando a ser identificado.
- pueden emplearse imágenes descargadas de internet o enviadas al dispositivo del usuario mediante aplicaciones de mensajería, redes sociales, etc.

## Sobre el SDK

Microsoft tiene disponible un SDK para aplicaciones Android que permite el uso de Face API en aplicaciones propias. Al contrario que su equivalente en Python, este SDK está pobemente documentado, los ejemplos son confusos y preparar las dependencias en Android Studio llevaron a realizar múltiples modificaciones en el código, degradando su calidad.

Por tanto, se tomó la decisión de no utilizar el SDK de Microsoft y crear funciones propias para conectar con Azure, empleando una biblioteca de cliente Http llamada `httpclient-android` (27) para realizar las solicitudes directamente a Azure. Para interpretar los resultados, se ha empleado la biblioteca JSON que incorpora Android Studio.

Una descripción genérica de la aplicación se muestra en la Figura 14. Al iniciar la aplicación, se selecciona una imagen de la galería del dispositivo. Si se detecta alguna cara, se puede enviar a Azure para su identificación. Si la identificación es positiva, la

aplicación permite añadir el rostro a la colección de imágenes del individuo identificado. En caso contrario, permite agregar una persona nueva.

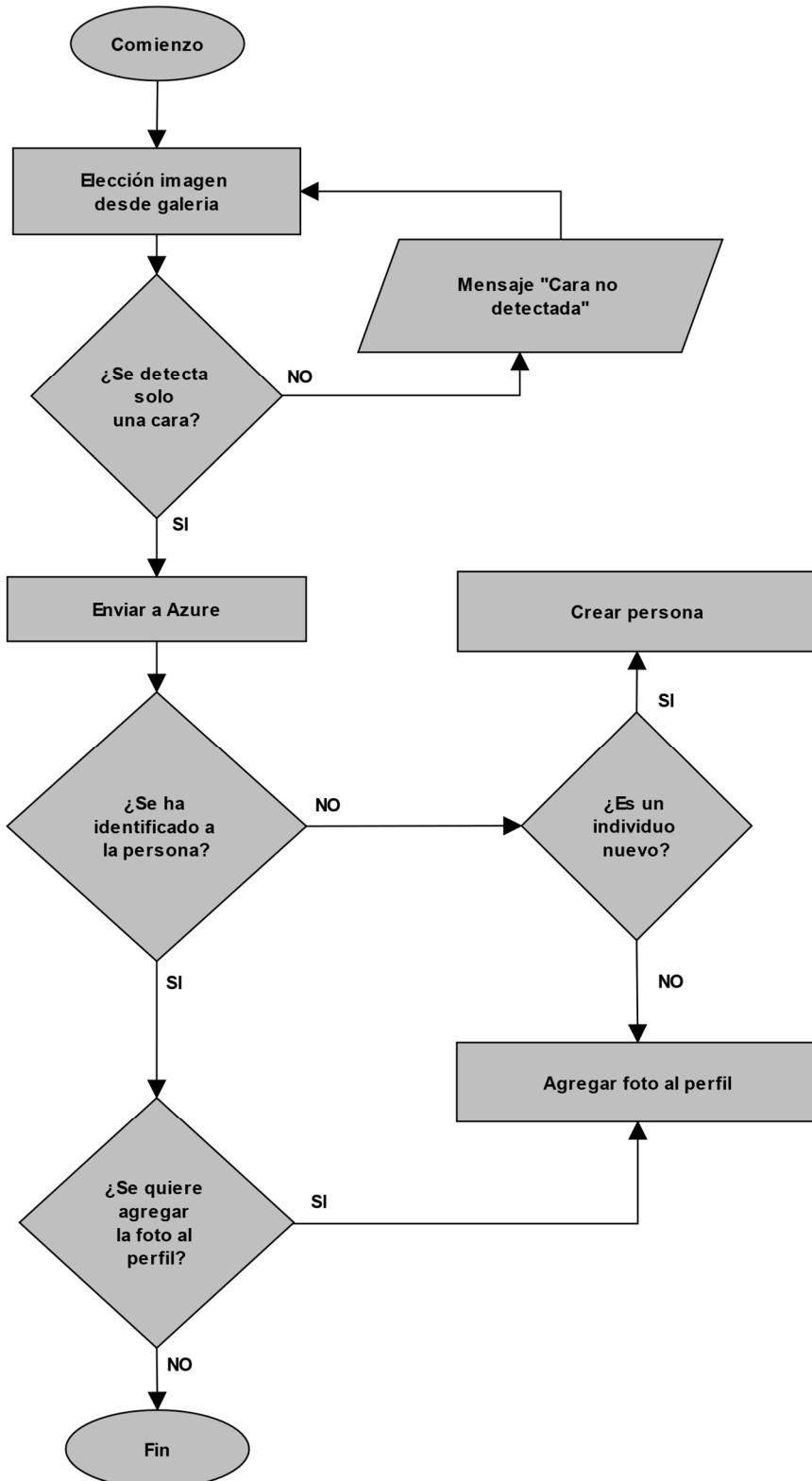


Figura 14. Esquema general de FacePal.

### 3.5.2. Código

La parte de código realizado se compone de varias clases divididas en cuatro grupos:

**Clases de Actividades:** declaran las funciones que se emplean en cada actividad. FacePal tiene varias actividades:

- MainActivity: pantalla principal.
- SettingsActivity: pantalla de configuración.
- TrainActivity: pantalla de entrenamiento de caras.
- AboutActivity: pantalla Acerca de...

**Clase con funciones Util:** contiene funciones separadas de la interfaz gráfica, de manera que son fácilmente editables y lo más modular posible. Se describe la signatura y una breve descripción de cada función que implementa esta clase estática:

- byte[] toBase64(Bitmap bm)  
Partiendo de un objeto *bitmap* se convierte en un *array* de *bytes* para el envío a Azure como parte del cuerpo del mensaje.
- String getBaseUrl()  
Recoge el nombre del servidor a emplear de la configuración de la aplicación y construye la URL base.
- String getKey()  
Recoge la clave de suscripción de la configuración de la aplicación.
- String getGroupName()  
Recoge el nombre del grupo de entrenamiento de la configuración de la aplicación.
- Bitmap detectFace(Bitmap bitmap)  
Partiendo de una imagen cargada, detecta una cara y recorta la imagen para eliminar información irrelevante. En caso de no encontrar un solo rostro, se retorna un objeto nulo.

- `String getName(String personID )`

Como se ha mencionado anteriormente, Azure no ofrece el nombre del candidato directamente al usar su servicio de reconocimiento, solo devuelve los `personID` de los posibles candidatos. Se ha creado esta función para revisar la lista de personas de un grupo y si el `personID` existe, devuelve el nombre real asociado.

- `String identiFace (Bitmap bitmap)`

Partiendo del `bitmap` recortado, este se envía a Azure y si la ejecución ha sido exitosa se recuperan los `personID` de los posibles candidatos. La función solo devuelve el `personID` del candidato más probable.

- `String trainGroup(String groupName) :`

Al agregar una cara a una persona, es necesario ejecutar el entrenamiento para que los nuevos datos sean tenidos en cuenta.

- `String addFace(String groupName, String personName, Bitmap bitmap)`

Agrega un `bitmap` que contiene una cara a una persona que pertenece a un grupo determinado. Esta función ejecuta siempre `trainGroup` al acabar de agregar la cara.

- `String getPersonID(String groupName, String name)`

Esta función devuelve el `personID` a partir de un nombre real.

- `String addPerson(String groupName, String name)`

Crea una persona nueva y la añade a un grupo determinado.

- `String catchJSONerror(String jsonString)`

Esta función se ejecuta cada vez que se recibe una respuesta desde Azure. El archivo recibido siempre está en formato JSON. Esta función comprueba si se ha devuelto un mensaje de error desde Azure y si es así, devuelve un mensaje describiendo el error; en caso contrario, devuelve el mensaje original sin alterar.

**Clases para tareas asíncronas:** heredan de la clase `AsyncTask` y sirven para poder realizar tareas en segundo plano como la comunicación con Azure mientras se pueden

presentar otras actividades como mensajes por pantalla. Para su uso se invocan por su nombre y el método `execute()`. Se sobrescriben tres métodos:

- `onPreExecute`: ejecuta antes de la tarea (mensajes).
- `doInBackGround`: codifica la tarea asíncrona.
- `onPostExecute`: ejecuta tareas tras la tarea asíncrona, como la presentación de resultados.

Las tareas asíncronas generan un problema: si se quiere devolver otro tipo de dato que no sea una cadena de caracteres, la tarea `onPreExecute` no se ejecuta y se pierde funcionalidad. Esto se ha corregido empleando la interfaz `AsyncResponde`; esta interfaz permite devolver tipos distintos de datos tras ejecutar `onPostExecute` y así poder recuperar información relevante.

En FacePal se emplean varias clases de este tipo que se describirán según corresponda.

#### **Clases auxiliares:**

- `GlobalClass`: permite acceder al contexto de la aplicación desde cualquier punto de la misma.
- `MiniSnack`: crea un objeto `Snackbar` con características adicionales (duración del mensaje y botón OK para cerrar) para la devolución de mensajes por parte de la aplicación.

### **3.5.3. Funcionamiento**

Para explicar el desarrollo de esta aplicación se describen sus distintas pantallas y se detalla cada elemento.

La pantalla principal (Figura 15) dispone de tres botones para ejecutar funciones, un área para la carga de imágenes y un menú de configuración.

La pantalla principal reside en la clase `MainActivity` y el archivo de diseño `activity_main.xml`.

La clase `MainActivity` dispone de cuatro funciones, dos de ellas relevantes:

- `onCreate` se ejecuta cuando se crea la actividad y es la responsable de crear los botones y asignar un objeto de escucha (`Listener`) cuya acción se ejecutará al pulsar el botón.
- `onActivityResult` se ejecuta tras el regreso de alguna ejecución. En esta aplicación, esta porción de código se ejecuta al seleccionar una imagen con el botón BUSCAR y el código ejecuta las funciones de recorte de la imagen.

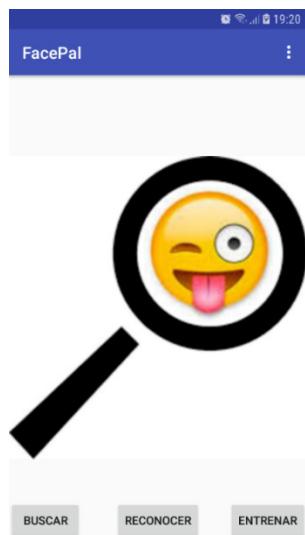


Figura 15. Pantalla principal de FacePal.

El menú de configuración (Figura 16) dispone de dos opciones: Configuración y Acerca de...

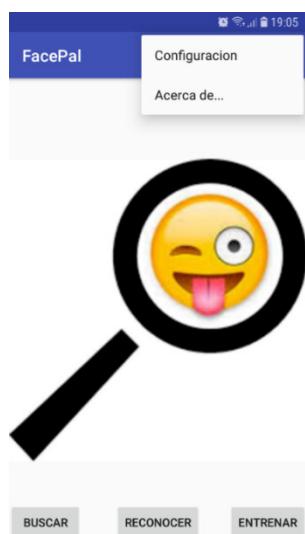


Figura 16. Menú de configuración.

La pantalla de configuración solo dispone del submenú General, permitiendo futuras ampliaciones. Dentro de este submenú se dispone de tres opciones de configuración (Figura 17):

- Nombre de grupo: grupo que almacena las caras a reconocer en el servidor.
- Clave de suscripción: clave aportada por Azure.
- Nombre del servidor: necesario para componer la URL, varía según la región.

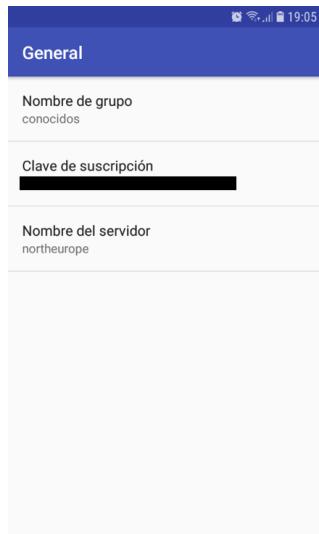


Figura 17. Pantalla de configuración.

A continuación, se describen las funciones de los botones de la pantalla principal. Se muestran varias capturas de pantalla con ejemplos de funcionamiento a lo largo del Capítulo 4.

#### Botón BUSCAR

Al pulsar este botón, se abre la galería de imágenes de la versión de Android correspondiente y permite seleccionar una imagen.

Una vez seleccionada la imagen, FacePal busca una cara. Si la detecta, recorta la imagen lo máximo posible para realizar un envío reducido y así ahorrar tiempo y datos, y muestra el recorte por pantalla.

Si la imagen contiene más de un rostro, ninguno o la rotación es incorrecta, devolverá un mensaje de error y mostrará la imagen por pantalla para que pueda comprobarse qué imagen fue cargada.

Al pulsar el botón se crea una instancia de la tarea asíncrona `AsyncFaceCroper`. Esta envía mensajes al usuario de la tarea que se ejecuta y se encarga de detectar el rostro y recortar la imagen. En esta clase se declara la interfaz `AsyncResponse` para que al finalizar la ejecución `onPostExecute` se recupere un booleano confirmando o no la detección de una cara y además un objeto `Bitmap` que contiene la cara recortada.

Si existe un rostro, los botones RECONOCER y ENTRENAR estarán disponibles. De lo contrario quedarán bloqueados informando que no se ha cargado una cara.

### **Botón RECONOCER**

Si existe una cara cargada, el botón RECONOCER ejecuta la tarea asíncrona correspondiente y conecta con Azure para emplear la función de identificación de cara.

Si el sujeto es conocido, se devuelve el nombre real. En caso contrario, devuelve un mensaje indicando que es un sujeto desconocido.

El botón RECONOCER instancia una tarea asíncrona `AsyncFaceDetector` que gestiona la detección de caras y la comunicación en segundo plano con Azure.

### **Botón ENTRENAR**

Esta tarea requiere crear una nueva actividad, que está contenida en la clase `trainActivity` y cuyo diseño está descrito en `activiy_train.xml`.

La pantalla de entrenamiento consta de cuatro partes:

- Área de texto para la introducción de nombres.
- Área de imagen para saber en todo momento con qué cara se trabaja.
- Botón AÑADIR CARA.
- Botón AÑADIR PERSONA.

Al iniciar la actividad, la cara recortada de la actividad anterior se carga en el área de imagen de esta actividad. Automáticamente se ejecuta un reconocimiento para determinar si esta cara corresponde a una persona que ya está en el sistema.

Si la cara corresponde a una persona ya conocida, el sistema bloquea el área de texto con el nombre de la persona y a su vez se bloquea el botón AÑADIR PERSONA.

En caso contrario, se bloquea el botón AÑADIR CARA y se desbloquea el área de texto y el botón AÑADIR PERSONA, para que pueda escribirse el nombre de la nueva persona.

### **Botón AÑADIR CARA**

Añade la cara cargada en el área de imagen a una persona existente en el sistema.

Al pulsar el botón se crea la tarea asíncrona `AsyncAddFace` que gestiona la comunicación en segundo plano con Azure, añadiendo la cara y ejecutando el entrenamiento del grupo.

### **Botón AÑADIR PERSONA**

Añade una nueva persona al sistema, crea la tarea asíncrona `AsyncAddPersona` que realiza tres tareas:

- Comprueba que el nombre introducido en el área de texto no existe en el sistema. Si existe, devuelve un error; en caso contrario, añade la persona al grupo de entrenamiento.
- Añade la cara a la persona anteriormente añadida.
- Ejecuta el entrenamiento del grupo para que los nuevos datos de la persona añadida formen parte del sistema.

## 4. Pruebas

En este capítulo se describen las pruebas llevadas a cabo para verificar el correcto funcionamiento del sistema, una vez desarrolladas las aplicaciones e instaladas en sus plataformas correspondientes. Las pruebas consistieron en el empleo de las aplicaciones desarrolladas de forma individual y en combinación, empleando para ello imágenes de diverso tipo.

### 4.1. Comprobación con imágenes existentes

Para comprobar que el sistema cumple los objetivos de este proyecto, cabe plantearse las siguientes hipótesis:

- H1: Al aumentar el número de imágenes, aumenta la confianza obtenida.
- H2: El sistema es capaz de distinguir personas con rasgos muy similares.

Para ello se ha empleado el módulo FCTools que forma parte de FCModule, con funciones usadas para el entrenamiento en la Raspberry Pi, aunque por comodidad se ha utilizado una versión reducida empleada en Windows 10. Esta versión se encuentra disponible en la documentación suplementaria de este proyecto.

Se han diseñado cuatro pruebas que emplean imágenes obtenida de Internet para poder confirmar ambas hipótesis.

- Bill Gates vs. Bill Gates.
- Ben Linus vs. Ben Linus.
- Bill Gates vs. Ben Linus.
- Bill Gates vs. Bill Gates Impersonator.

#### 4.1.1. Prueba Bill Gates vs. Bill Gates

En esta primera prueba se han empleado diez fotos de un personaje público del cual es fácil obtener imágenes en Internet. El personaje elegido ha sido Bill Gates, cofundador de Microsoft, edad 62 años, ojos azules, gafas, piel blanca, nariz mediana (Figura 18).



Figura 18. Retratos de Bill Gates (Fuente Google).

Para confirmar la hipótesis H1 se han ido añadiendo las caras una a una al sistema. Por cada imagen añadida se ha ejecutado una comprobación con la imagen de prueba de la Figura 19 que nunca es añadida al sistema, obteniendo el valor de confianza en cada iteración.



Figura 19. Imagen de prueba de Bill Gates (Fuente Microsoft).

La prueba puede explicarse con el siguiente pseudocódigo:

- crearPersona ("Bill Gates")
- imagenPrueba //imagen de prueba
- colecciónCaras //colección de fotos que el sistema desconoce
- por cada Cara en colecciónCaras
  - añadir Cara a Persona
  - realizar entrenamiento
  - comparar con imagenPrueba
  - imprimir candidato y confianza

En la Figura 20 se observa un ejemplo de ejecución de estas pruebas.

```
Windows PowerShell
PS E:\PFG\pyazure> python test1.py
{'persistedFaceId': '6bac7072-ac1d-4d45-b8cb-7515f019910b'}
Datos caraTest
[{"faceId": "dcbde737-5994-488b-916a-ae5eb78d9ef", "faceRectangle": {"top": 108, "left": 57, "width": 214, "height": 214}}]
Datos mejor candidato
[{"personId": "7d76c34d-f856-45e1-9c71-e74bf9e7a513", "confidence": 0.78474}]
Candidato: Bill Gates Confianza: 0.78474
PS E:\PFG\pyazure>
```

Figura 20. Ejemplo de ejecución durante la prueba Bill vs. Bill.

Con los datos obtenidos, se ha elaborado el gráfico de confianza de la Figura 21. Como puede observarse, cuantas más imágenes se añaden a la misma persona, mayor es la confianza obtenida. Por tanto, la hipótesis H1 queda confirmada.

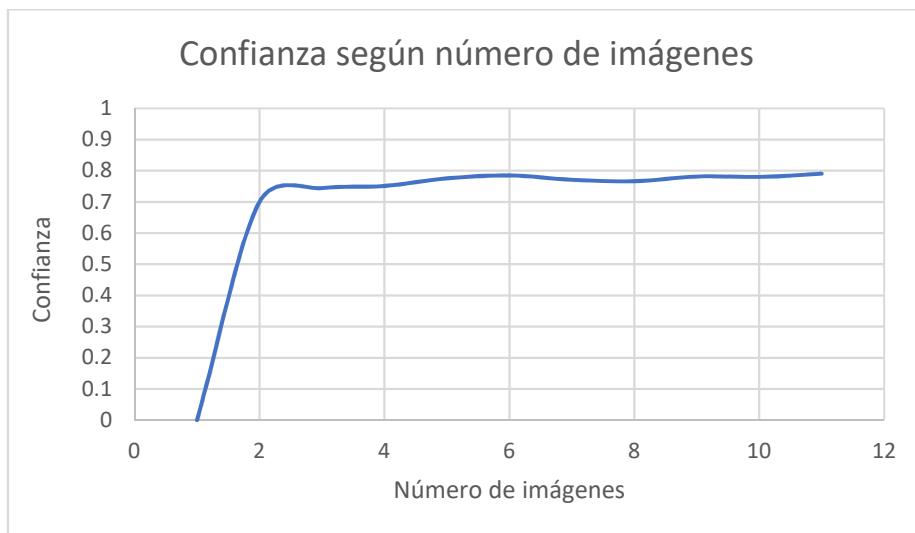


Figura 21. Gráfica Bill vs. Bill.

Aunque como se ve en el gráfico la tendencia es de crecimiento y tiende a estabilizarse, se pueden observar pequeñas fluctuaciones a las que conviene prestar atención. Estas ocurren al añadir las imágenes bill5.jpg, bill6.jpg y bill8.jpg, y puede deberse a los siguientes motivos:

- bill5.jpg:
  - o el cristal de las gafas presenta demasiado brillo.
  - o la resolución de la imagen es baja comparada con las demás.

- bill6.jpg:
  - o foto de juventud, aproximadamente 40 años, el resto de las fotos son más recientes
- bill8.jpg:
  - o la pose es distinta, la cabeza está inclinada
  - o la boca está más abierta que en el resto de la colección

#### 4.1.2. Prueba Ben Linus vs. Ben Linus

Para esta prueba, se ha añadido una nueva persona que tiene una serie de rasgos similares a Bill Gates pero sin ser idéntico. La persona elegida es Ben Linus, principal villano de la serie “Lost”, interpretado por el actor Michael Emerson, edad 63 años, ojos azules, gafas, piel blanca y nariz mediana.

Como en el caso anterior, se ha creado la persona de Ben Linus y se le han añadido las diez caras de la Figura 22.



Figura 22. Caras elegidas de Ben Linus (Fuente Google).

La diferencia con la prueba anterior es que las imágenes no son tan homogéneas en tamaño ni forma, de manera que se puede evaluar si el sistema es capaz de funcionar adecuadamente a pesar de la información adicional fuera del área de la cara, apoyando la hipótesis H1.



Figura 23. Imagen de prueba de Ben Linus (Michael Emerson) (Fuente Wikipedia).

La prueba comienza comprobando cada una de las diez caras de la Figura 22 y la imagen de prueba de la Figura 23, para observar que ninguna de ellas es detectada como Bill Gates. En la Figura 24 se muestran los resultados de ejecución de esta prueba y, en la Figura 25, el resultado de la prueba con la imagen de la Figura 23.

```
PS E:\PFG\pyazure> python test2.py
Datos caraTest, archivo: ben0.jpg
[{"faceId": "3e061cfa-2ae1-4f20-b162-8858ffcf3702", "faceRectangle": {"top": 130, "left": 591, "width": 732, "height": 732}]
Datos mejor candidato
[{"faceId": "3e061cfa-2ae1-4f20-b162-8858ffcf3702", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben1.jpg
[{"faceId": "46fb1bcc-84b3-43d5-8c1b-5d378379146d", "faceRectangle": {"top": 265, "left": 210, "width": 322, "height": 322}]
Datos mejor candidato
[{"faceId": "46fb1bcc-84b3-43d5-8c1b-5d378379146d", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben2.jpg
[{"faceId": "f6c462b8-10fb-4e16-8309-48f3c258181a", "faceRectangle": {"top": 86, "left": 34, "width": 135, "height": 135}]
Datos mejor candidato
[{"faceId": "f6c462b8-10fb-4e16-8309-48f3c258181a", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben3.jpg
[{"faceId": "72c553ba-9871-4212-a83f-bd343cd06b62", "faceRectangle": {"top": 60, "left": 253, "width": 138, "height": 138}]
Datos mejor candidato
[{"faceId": "72c553ba-9871-4212-a83f-bd343cd06b62", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben4.jpg
[{"faceId": "ff6617bf-2f77-4cla-a85f-633e53c4dd0c", "faceRectangle": {"top": 457, "left": 343, "width": 634, "height": 634}]
Datos mejor candidato
[{"faceId": "ff6617bf-2f77-4cla-a85f-633e53c4dd0c", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben5.jpg
[{"faceId": "199d5c01-91e8-4954-b1ec-4b23e8e66a60", "faceRectangle": {"top": 69, "left": 50, "width": 81, "height": 81}]
Datos mejor candidato
[{"faceId": "199d5c01-91e8-4954-b1ec-4b23e8e66a60", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben6.jpg
[{"faceId": "c12fe8d7-ddb2-4b81-a1a3-c1dbdf4b5994", "faceRectangle": {"top": 35, "left": 74, "width": 44, "height": 44}]
Datos mejor candidato
[{"faceId": "c12fe8d7-ddb2-4b81-a1a3-c1dbdf4b5994", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben7.jpg
[{"faceId": "0378e7e5-a334-484d-9399-a400e625e6f7", "faceRectangle": {"top": 52, "left": 151, "width": 130, "height": 130}]
Datos mejor candidato
[{"faceId": "0378e7e5-a334-484d-9399-a400e625e6f7", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben8.jpg
[{"faceId": "8f5c36c1-69bb-47ba-a77d-b40bd245ec72", "faceRectangle": {"top": 49, "left": 141, "width": 66, "height": 66}]
Datos mejor candidato
[{"faceId": "8f5c36c1-69bb-47ba-a77d-b40bd245ec72", "candidates": []}]
No existe un candidato que concuerde
Datos caraTest, archivo: ben9.jpg
[{"faceId": "2eb0c4c9-57f4-4760-83ac-ad9f95742dda", "faceRectangle": {"top": 117, "left": 241, "width": 234, "height": 234}]
Datos mejor candidato
[{"faceId": "2eb0c4c9-57f4-4760-83ac-ad9f95742dda", "candidates": []}]
No existe un candidato que concuerde
PS E:\PFG\pyazure>
```

Figura 24. Prueba de los diez rostros de Ben Linus.

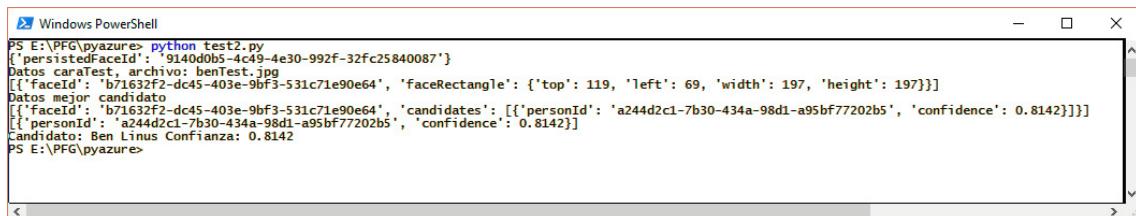
---

```
PS E:\PFG\pyazure> python test2.py
Datos caraTest
[{"faceId": "8b81a686-4adb-4d42-90d5-2e9f60bd8d72", "faceRectangle": {"top": 119, "left": 69, "width": 197, "height": 197}]
Datos mejor candidato
[{"faceId": "8b81a686-4adb-4d42-90d5-2e9f60bd8d72", "candidates": []}]
No existe un candidato que concuerde
PS E:\PFG\pyazure>
```

Figura 25. Ejecución con la imagen de prueba de Ben Linus antes del entrenamiento.

En ningún caso el sistema reconoció el rostro como perteneciente a Bill Gates, verificando así la hipótesis H2. Se debe tener en cuenta que Azure no devuelve a ningún candidato cuyo umbral de confianza esté por debajo de 0.5. Esto no significa que no exista algún tipo de similitud, sino que no es suficiente como para ser fiable. Por debajo de ese umbral (0.5) prácticamente sería como acertar la similitud tirando una moneda a cara o cruz.

En la segunda parte de esta prueba se han añadido cada una de las diez caras de Ben Linus y se han contrastado con la imagen de prueba de la Figura 23. En todos los casos se obtuvo una identificación satisfactoria. En la Figura 26, se muestra la ejecución con la última cara.



```

Windows PowerShell
PS E:\PFG\pyazure> python test2.py
{'persistedFaceId': '9140db5-4c49-4e30-992f-32fc25840087'}
Datos caraTest, archivo: benTest.jpg
[{"faceId": "b71632f2-dc45-403e-9bf3-531c71e90e64", "faceRectangle": {"top": 119, "left": 69, "width": 197, "height": 197}}]
Datos mejor candidato
[{"faceId": "b71632f2-dc45-403e-9bf3-531c71e90e64", "candidates": [{"personId": "a244d2c1-7b30-434a-a98d1-a95bf77202b5", "confidence": 0.8142}]}
Candidato: Ben Linus Confianza: 0.8142
PS E:\PFG\pyazure>

```

Figura 26. Ejecución con imagen de prueba de Ben Linus tras el entrenamiento.

Con los resultados obtenidos se ha generado el gráfico de la Figura 27. El resultado es el esperado: a medida que aumenta el número de imágenes, aumenta la confianza que devuelve el sistema al usar la imagen de prueba y en todos los casos se detecta la imagen de prueba como Ben Linus, reforzando así la hipótesis H1.

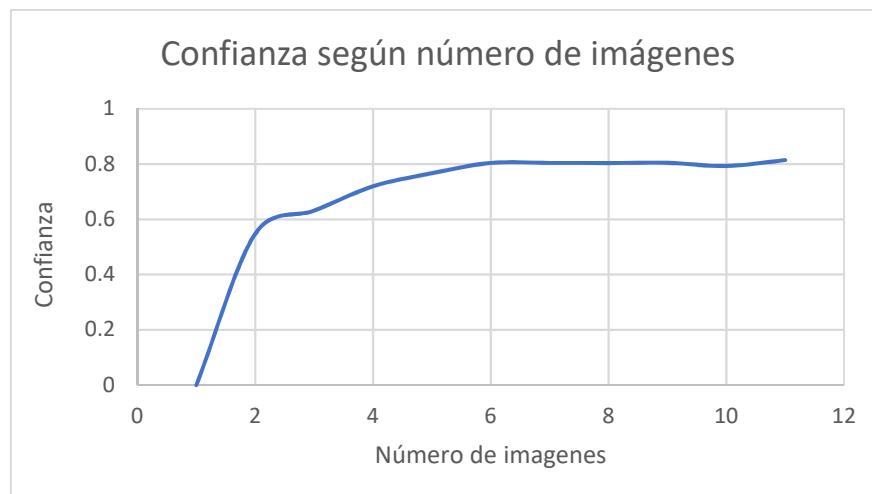


Figura 27. Gráfica Ben vs. Ben.

Un análisis más preciso del gráfico también revela otros datos de interés. El sistema no comienza a estabilizarse hasta la sexta imagen añadida, mientras que en la prueba Bill Gates vs. Bill Gates con la segunda imagen ya se conseguía un valor alto y casi estable de confianza. El motivo de este arranque tardío es debido a que la colección de imágenes de Ben Linus es más heterogénea. La imagen ben0.jpg (la primera que se sube al sistema) es oscura y Ben no tiene gafas, mientras que la imagen de prueba es luminosa y con gafas, lo que explica que el primer dato de confianza (0.54765) sea poco mejor que adivinar.

Pero incluso con una colección tan heterogénea (gafas, sin gafas, con sombrero, mucha luz, poca luz), el sistema al final del entrenamiento consigue una buena confianza (mayor del 80%) con la imagen de prueba.

#### 4.1.3. Prueba Bill Gates vs. Ben Linus

Esta prueba está diseñada para poder verificar la hipótesis H2.

Para esta prueba se han usado como imágenes de prueba todas y cada una de las imágenes de entrenamiento empleadas anteriormente. De esta manera se puede saber si alguna de las caras de Bill es reconocida como Ben y viceversa.

Los datos de confianza obtenidos se reflejan en la Tabla 3.

Archivos	Bill Gates	Ben Linus
bill0.jpg	0.83524	0
bill1.jpg	0.80866	0
bill2.jpg	0.85215	0
bill3.jpg	0.82128	0
bill4.jpg	0.87035	0
bill5.jpg	0.88662	0
bill6.jpg	0.82094	0
bill7.jpg	0.81845	0
bill8.jpg	0.80564	0
bill9.jpg	0.87751	0
ben0.jpg	0	0.80195
ben1.jpg	0	0.72930
ben2.jpg	0	0.76416

ben3.jpg	0	0.81979
ben4.jpg	0	0.83860
ben5.jpg	0	0.85980
ben6.jpg	0	0.60980
ben7.jpg	0	0.86963
ben8.jpg	0	0.79360
ben9.jpg	0	0.76265

Tabla 3. Resultado de la prueba Bill Gates vs. Ben Linus.

Tras la ejecución del test se puede observar que ninguna cara de Bill Gates ha sido clasificada como Ben Linus, ni como segunda opción con peor confianza. Del mismo modo, ninguna imagen de Ben Linus ha sido identificada como Bill Gates.

Por tanto, gracias a esta prueba se puede corroborar la hipótesis H2.

#### 4.1.4. Prueba Bill Gates vs. Bill Gates Impersonator

Por último, se decidió hacer una prueba con una imagen curiosa encontrada en <http://www.lookalike.com>.

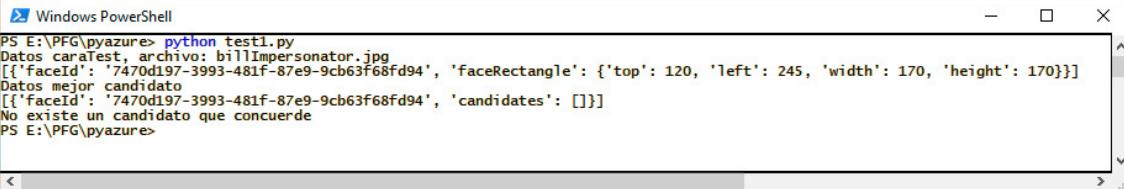
Esta web ofrece el servicio de contratación de personas muy parecidas físicamente a personajes conocidos (actores, políticos, empresarios), que son entrenados para comportarse, vestirse y citar frases de dichos personajes.

En el catálogo, se ha encontrado a un imitador de Bill Gates bastante convincente, como se puede apreciar en la Figura 28. En esta prueba, se ha sometido la foto del imitador al método de identificación.



Figura 28. Imitador de Bill Gates (Fuente lookalike.com).

Como se observa en la Figura 29 y se refleja en la Tabla 4 el clasificador no encuentra a ningún candidato que tenga similitudes con el imitador. Por tanto, la hipótesis H2 queda reforzada.



```
PS E:\PFG\pyazure> python test1.py
Datos caratest, archivo: billImpersonator.jpg
[{"faceId": "7470d197-3993-481f-87e9-9cb63f68fd94", "faceRectangle": {"top": 120, "left": 245, "width": 170, "height": 170}}
Datos mejor candidato
[{"faceId": "7470d197-3993-481f-87e9-9cb63f68fd94", "candidates": []}]
No existe un candidato que concuerde
PS E:\PFG\pyazure>
```

Figura 29. Ejecución de la prueba con Bill Gates Impersonator.

Archivos	Bill Gates	Ben Linus
billImpersonator.jpg	0	0

Tabla 4. Resultados de la prueba Bill Gates vs. Bill Gates Impersonator.

## 4.2. Prueba con imágenes existentes y reconocimiento con FaceRecon

Para esta prueba se ha creado mediante el uso de FCTools la persona Admin a la cual se han añadido suministran fotos del propio autor de este proyecto para que sea reconocido por la aplicación FaceRecon en situaciones reales.

Se han empleado 9 fotos del autor con distintas edades, luminosidad, barba y corte de pelo (Figura 30).

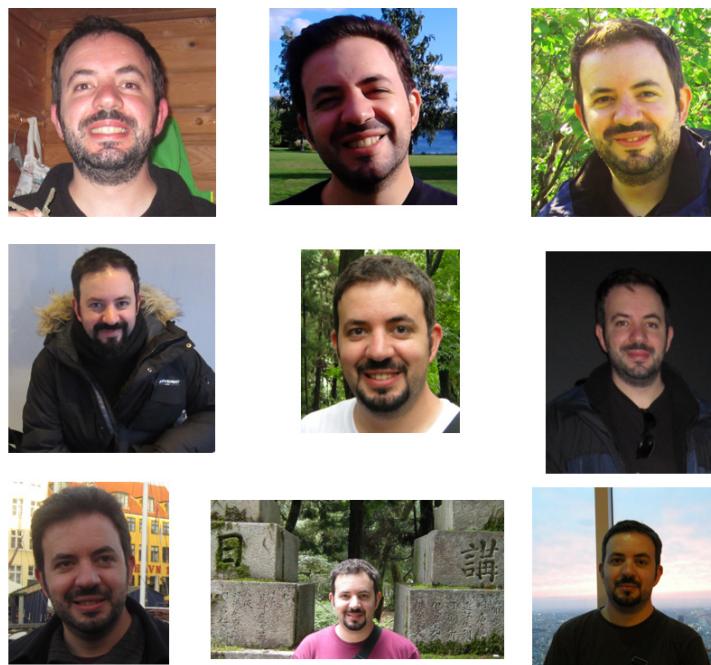


Figura 30. Fotos de entrenamiento de Admin.

Al contrario que en los casos anteriores, no se empleó una imagen de prueba para verificar el grado de confianza ni si el sistema devolvía a la persona correcta, es decir, esta prueba se realizó a ciegas para verificar si la aplicación FaceRecon funcionaba correctamente. Para ello se ejecutó FaceRecon empleando la cámara de la Raspberry Pi. El sistema reconoció un rostro en la imagen y procedió a su identificación, ofreciendo como resultado la identificación de Admin y devolviendo la imagen capturada con el rostro reconocido y la etiqueta correspondiente (Figura 31)

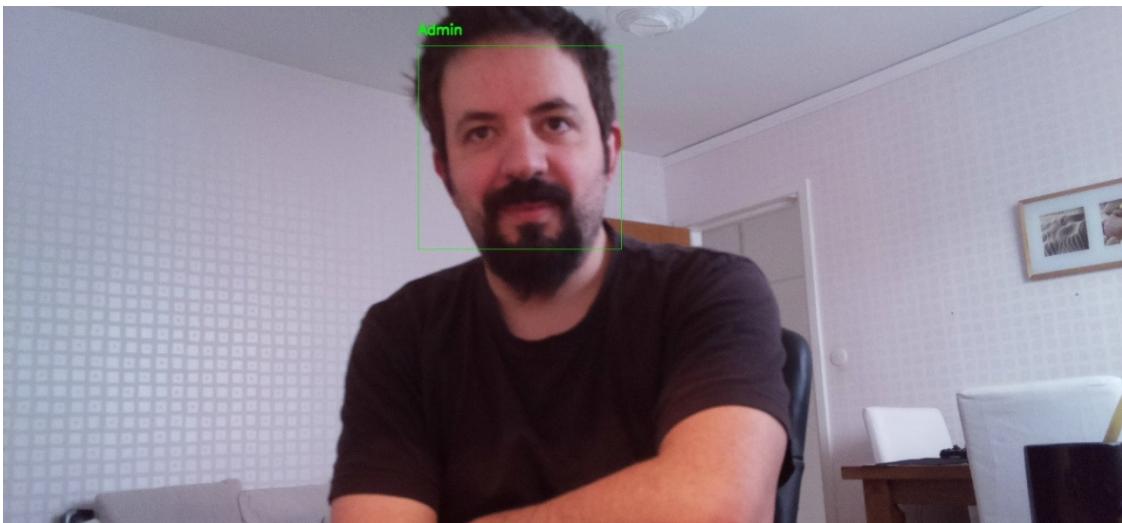


Figura 31. Primer reconocimiento de Admin.

En una segunda prueba, se adquirió una nueva imagen desde un ángulo distinto y con iluminación artificial frontal y natural trasera (Figura 32). FaceRecon volvió a reconocer a Admin sin problemas.



Figura 32. Imagen de Admin con iluminación alternativa.

### 4.3. Entrenamiento y reconocimiento en FacePal

Para esta prueba se han tomado retratos de tres individuos con la cámara de un teléfono móvil y se han sometido a la función de reconocimiento de FacePal. En primer lugar, se confirmó que estos individuos no eran reconocidos al no existir ninguna imagen previa en la base de datos. A continuación, estas personas y sus imágenes se añadieron a la lista de personas conocidas. Una nueva ejecución de la aplicación de reconocimiento resultó en la satisfactoria identificación de los individuos, demostrando así el correcto funcionamiento de FacePal.

#### Sujeto 1: Akbar Espaillat

En la Figura 33 se puede observar primero la imagen tomada con la cámara, luego la verificación de que el sujeto es desconocido y, por último, la confirmación de que la persona se agregó al sistema.

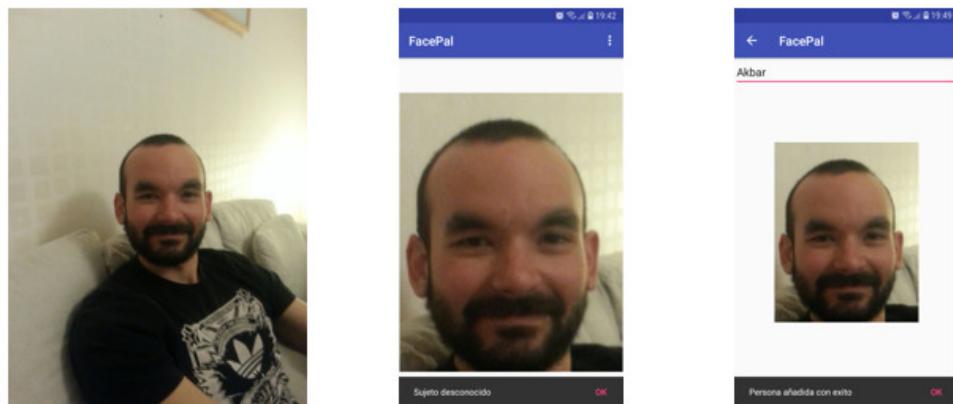


Figura 33. Prueba de identificación y creación del sujeto Akbar Espaillat.

#### Sujeto 2: Bastian Schiffthaler

La Figura 34 muestra cómo se agregó a Bastian Schiffthaler al sistema. En la foto empleada el sujeto tiene gafas puestas. En la Figura 35 se comprueba que el sistema reconoció a Bastian sin gafas y se añadió una nueva cara a esta persona.

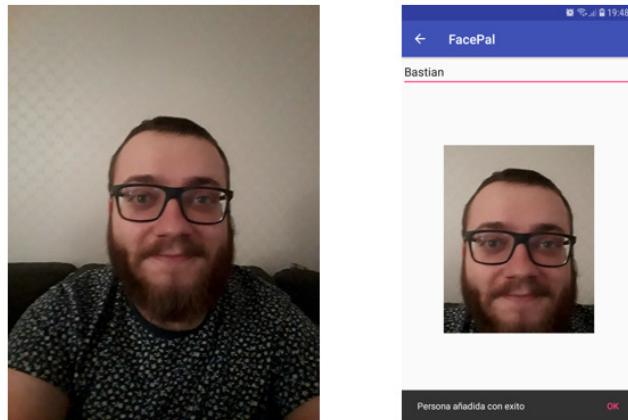


Figura 34. Creación de persona Bastian Schiffthaler.

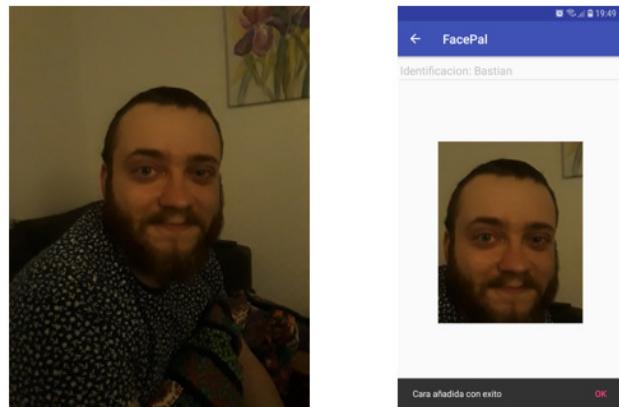


Figura 35. Cara agregada a Bastian Schiffthaler.

### Sujeto 3: Javier Alvarez

En el caso de Javier Alvarez se han tomaron dos retratos distintos: uno con su aspecto habitual y otro con gafas y gorro. El propósito era saber si mediante FacePal se identificaba al mismo individuo a pesar de tener un aspecto distinto.

La Figura 36 muestra cómo se agregó la primera imagen al sistema, cómo el sistema reconoció la segunda foto como “Javi” a pesar de que las gafas reflejan la luz y las cejas y el pelo no son visibles, y la tercera foto confirma que se agregó la segunda cara a los datos de la persona.

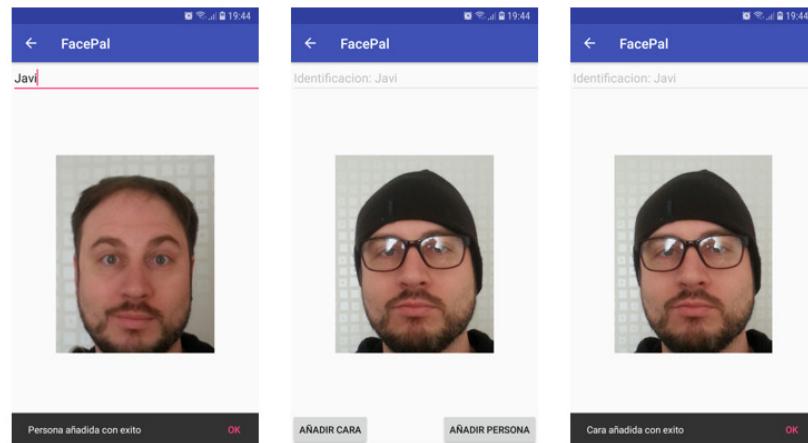


Figura 36. Creación del sujeto Javier Alvarez e imagen añadida.

Tras estas pruebas se pudo comprobar el buen funcionamiento de la aplicación, cómo no se confunde con otros individuos cargados anteriormente y, además, muestra la precisión de los algoritmos de Azure al solo necesitar una foto para reconocer a un individuo a pesar de que las fotos presentan elementos distintos.

#### 4.4. Entrenamiento y reconocimiento FacePal+FaceRecon

Llegados a este punto, el grupo de personas contenía varias personas almacenadas y en algunos casos varias fotos por persona, que habían sido añadidas durante las anteriores pruebas de FaceRecon o mediante el uso de la aplicación FacePal. Para comprobar el funcionamiento conjunto de ambas aplicaciones, se decidió emplear FaceRecon durante un día y comprobar si reconocía individuos en una situación real. Se tomaron aproximadamente 300 fotos durante el funcionamiento continuo. La mayoría de estas imágenes eran del autor (Admin) frente al ordenador, siendo poco relevantes. El resto contenían a más de una persona. En esta sección, se muestran tres imágenes representativas para demostrar el correcto funcionamiento.

La Figura 37 muestra a Akbar y Admin reconocidos durante una demostración de la aplicación.

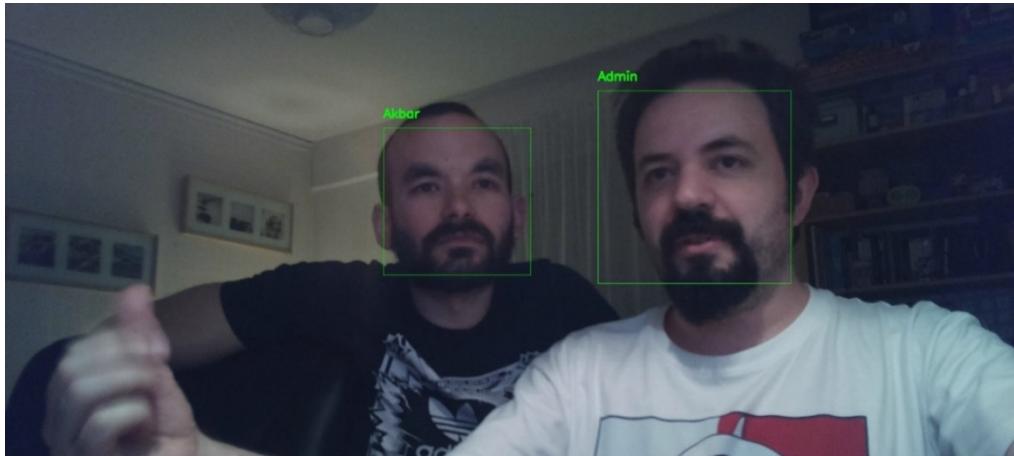


Figura 37. Reconocimiento de Akbar y Admin.

La Figura 38 muestra a Javi y Admin.



Figura 38. Reconocimiento de Javi y Admin.

La Figura 39 muestra a Bastian reconocido, pero no a Admin, porque no todo el rostro fue detectado. En el fondo de la imagen el sistema señala a un desconocido, un sujeto que no fue entrenado en el sistema.

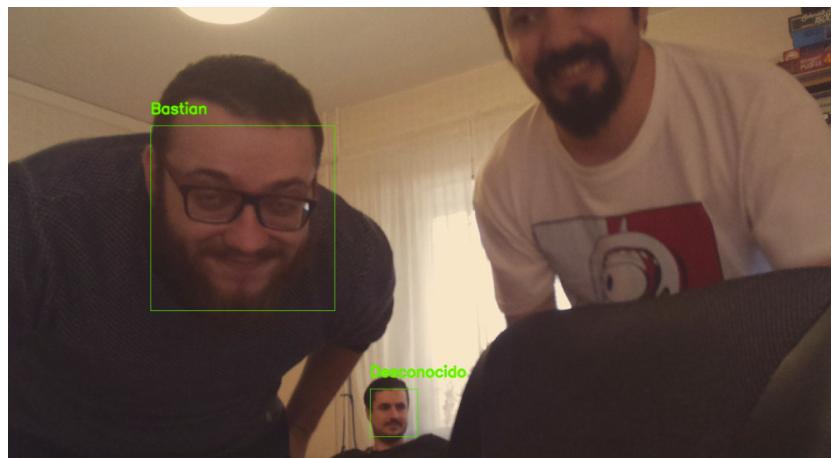


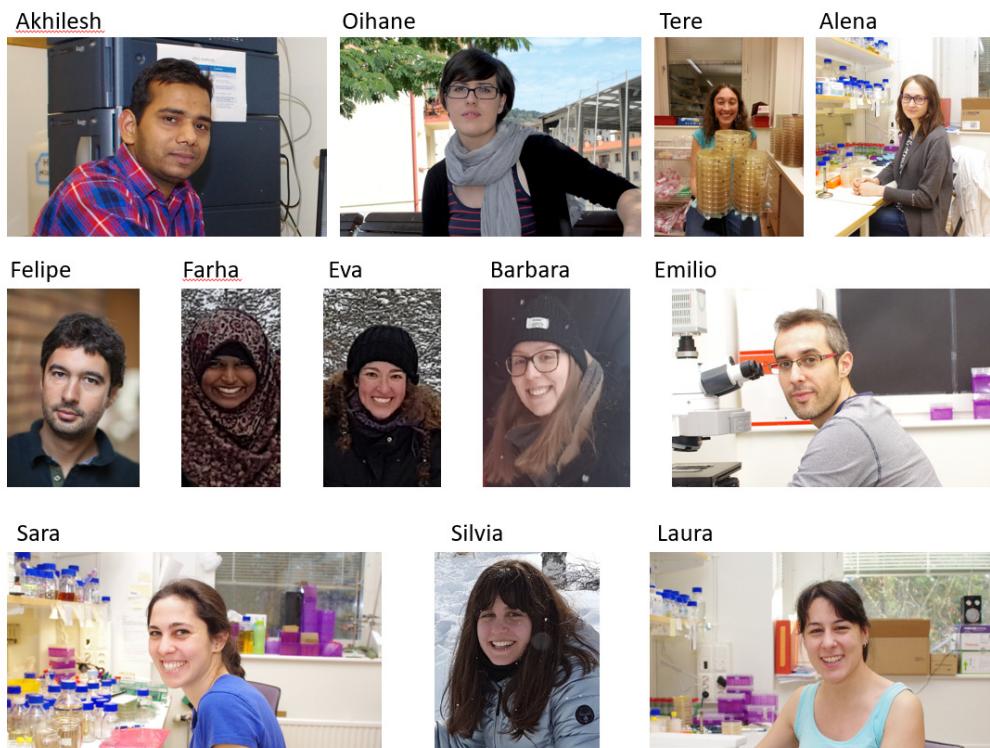
Figura 39. Reconocimiento de Bastian.

Con los datos de entrenamiento de las pruebas anteriores, y con los resultados obtenidos en esta prueba, se puede concluir que FaceRecon reconoció los rostros que se esperaban y que la combinación FacePal y aplicaciones basadas en FCModule funciona adecuadamente.

## 4.5. Reconocimiento múltiple

En esta prueba se trasladó el sistema a otra ubicación, gracias al uso de una batería portátil y una pantalla pequeña.

Se entrenó el sistema con personas adicionales mediante el uso de FacePal, empleando fotos de distintos tamaños y calidades como se observa en la Figura 40. Tras el entrenamiento, las personas implicadas posaron para distintas fotos, se seleccionaron las más significativas y se analizaron los resultados obtenidos.



*Figura 40. Nuevas personas.*

Los resultados obtenidos verifican el correcto funcionamiento del sistema incluso con capturas con varias personas. Sin embargo, existen algunas observaciones que deben ser analizadas en cada imagen.

Como se observa en la Figura 41, 12 de 13 personas fueron reconocidas con éxito, en un tiempo total de 15 segundos. Oihane (esquina superior derecha) fue marcada como “Desconocido”. Esto pudo deberse a que no se ve completamente su cara.



Figura 41. Foto grupal 1.

En la Figura 42, se buscó tomar una foto similar cambiando la posición de algún miembro para observar si esto afectaba a la detección. En este caso, las mismas 12 personas fueron detectadas en un tiempo de 16 segundos. La persona marcada como “Desconocido” fue de nuevo Oihane, como en el caso anterior, aunque en esta foto su rostro no estaba parcialmente tapado, borroso o torcido. Al ser la misma persona la que no había sido reconocida, se añadió la imagen de la Figura 43 para intentar conseguir mejores resultados.



Figura 42. Foto grupal 2.



Figura 43. Imagen adicional de Oihane.

En la Figura 44, se puede observar que, de 12 personas en total, dos rostros no se reconocieron como tal y otros dos se marcaron como “Desconocido”, todo en un tiempo de 13 segundos. Una vez más Oihane (segunda por la izquierda) fue marcada como “Desconocido”. La otra persona marcada como “Desconocido” (camiseta blanca) no había sido entrenado en el sistema, por tanto, el resultado es el esperado. Los rostros de Silvia y Barbara no fueron reconocidos en esta imagen. Entre los posibles motivos se encuentran la lejanía de los rostros, el hecho de que sea una foto a contraluz y, en el caso de Barbara, la posición torcida de su cabeza.



Figura 44. Foto grupal 3.

El caso de Oihane llama la atención por varias cuestiones. En primer lugar, se pensó que no la reconocía debido a que no estaba usando gafas en ese momento, pero las pruebas con Bastian Schiffthaler (Figura 35) y Javier Alvarez (Figura 36) demuestran que no es motivo suficiente para la no detección. Posteriormente se pensó en la calidad de las fotos de entrenamiento, pero otros sujetos se entrenaron con una sola foto de inferior calidad. Esto implica que no siempre el sistema puede funcionar con una foto. En algunos casos, los atributos faciales serán más difíciles de detectar y para que el clasificador funcione de manera correcta habrá que añadir más fotos.

Nota adicional: esta prueba grupal se realizó empleando la versión de pago del servicio Azure, dado que, al incrementar el número de personas, incrementa el número de transacciones y en la versión gratuita se supera fácilmente el límite de transacciones por minuto. Esto puede evitarse creando una pausa entre reconocimientos, pero a cambio de incrementar el tiempo de ejecución total.

#### 4.6. Reconocimiento múltiple con FacePi

El funcionamiento de FaceRecon y FacePi es muy similar, ambos usan las funciones de FCModule para ejecutar las tareas reconocimiento de rostros. Para demostrar su correcto funcionamiento, se realizó una prueba con FacePi.

Para esta prueba se empleó la Figura 18, que contiene diez caras de Bill Gates. en primer lugar, se eliminó el texto de la figura. A continuación, se cargó la imagen en FacePi y se pulsó el botón RECONOCER.

La Figura 45 muestra el resultado de la ejecución. Todas y cada una de las diez caras fueron identificadas como Bill Gates, demostrando que el programa funciona como se esperaba.

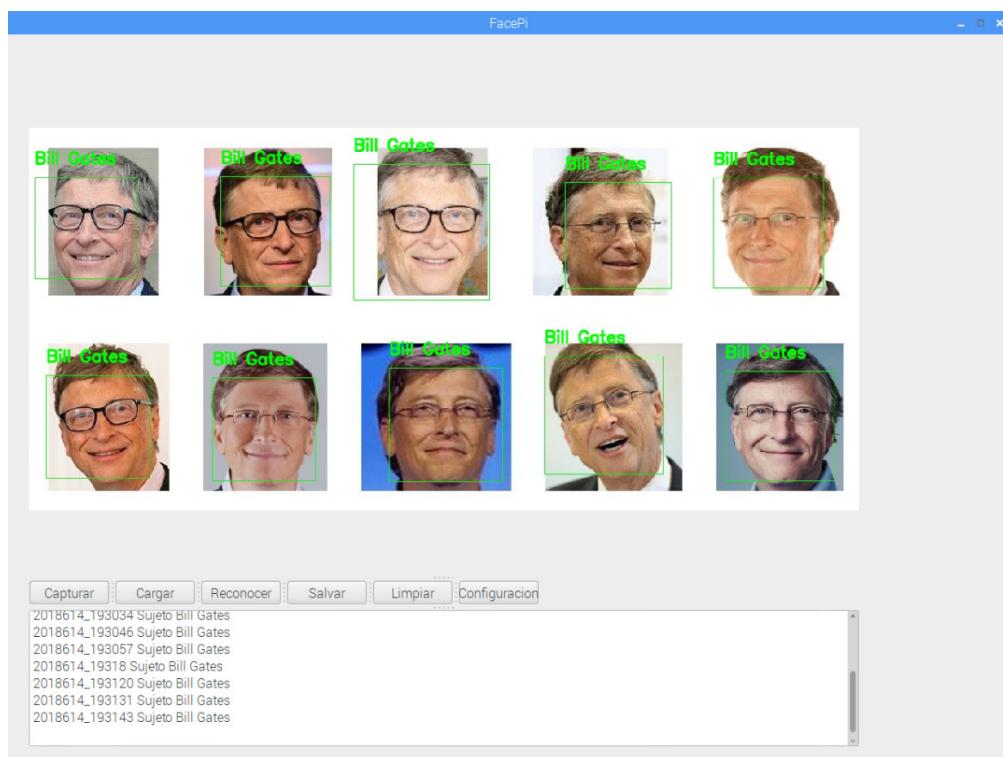


Figura 45. Ejecución de FacePi.

#### 4.7. Reconocimiento y comunicación mediante FaceBT

Finalmente, se diseñó una prueba para emplear el sistema mediante FaceBT. Para esta prueba se empleó solo la Raspberry Pi con su cámara y una batería externa, como se ve en la Figura 46 y un dispositivo para comunicarse con la Raspberry Pi.



Figura 46. Raspberry Pi con batería externa.

En la prueba se conectó la Raspberry Pi con la batería y se empleó una *tablet* Surface con conexión Bluetooth y el programa Bluetooth Serial Terminal (29), que permite la comunicación entre ambos dispositivos.

Para esta prueba se empleó al autor (Admin) (Figura 30) y a Laura (Figura 40), que ya estaban en el grupo de personas de Azure. A través de la aplicación de terminal, se solicitó a FaceBT un archivo JSON con la información de la imagen capturada.

En la Figura 47 se observa que el programa se conectó con el dispositivo llamado FaceBT (Raspberry Pi del proyecto) y que se envió la orden `getJSON`. FaceBT al recibir la orden, dio la respuesta con el mensaje “Modo envío: JSON”, indicando que se entendió bien la orden. A continuación, FaceBT devolvió una serie de mensajes sobre el estado de la ejecución y posteriormente devolvió la información en formato JSON entre los símbolos <<< y >>> por los motivos explicados en el Capítulo 3.

De la información obtenida se extrae que se ha detectó a Laura (Laurita) con una confianza de 0.7236 y a Admin con una confianza de 0.8344, concluyendo que la prueba había sido satisfactoria.

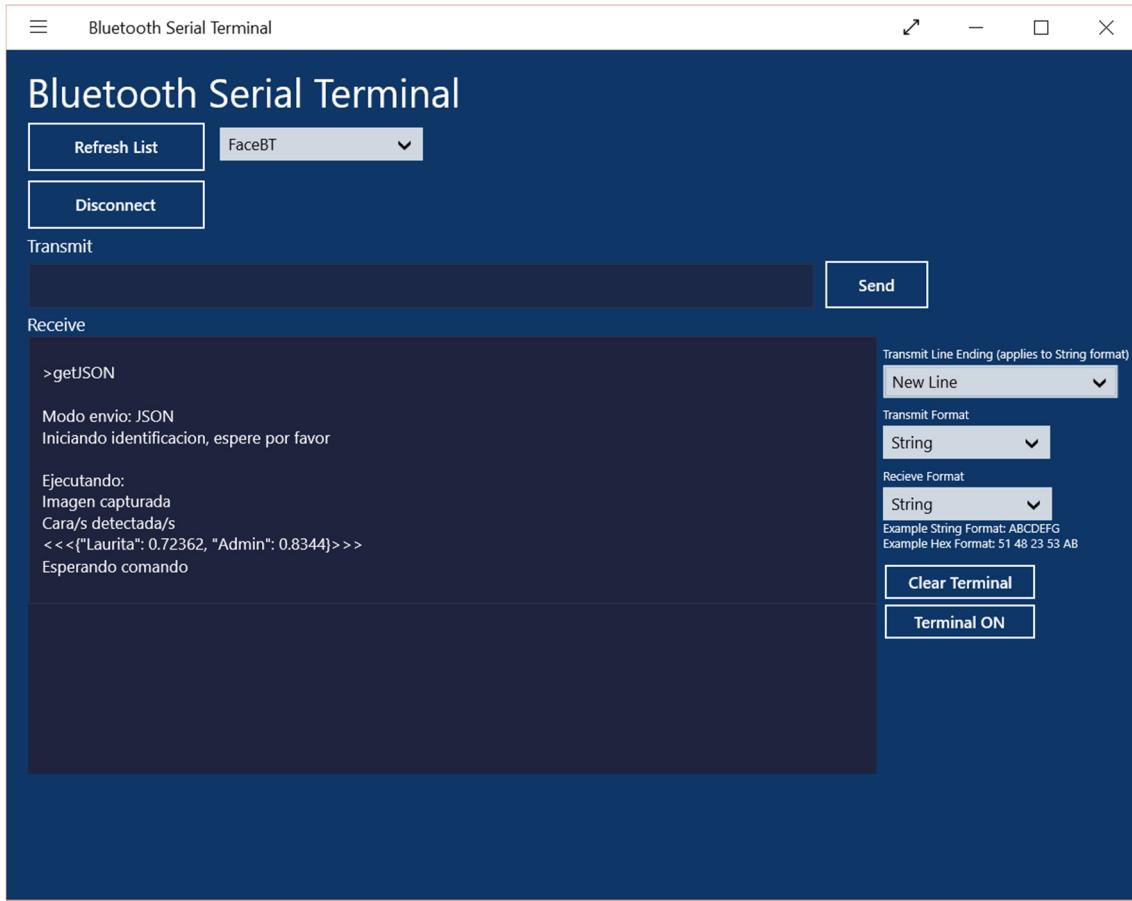


Figura 47. Comunicación con FaceBT.



## 5. Presupuesto

En este capítulo se realizarán estimaciones de los costes de desarrollo del conjunto de aplicaciones desarrollados para este proyecto y una estimación de costes de adquisición y uso.

### 5.1. Estimación de costes de desarrollo

Para estimar los costes de desarrollo de este conjunto de aplicaciones se realiza un cálculo del esfuerzo en persona-mes y del coste económico del desarrollo. Una manera de calcular los costes es determinar primero los puntos de función y con ellos obtener una estimación de las LOC (líneas de código) de las que se compondrá el proyecto. Como se dispone del dato de LOC (Tabla 5), no es necesario calcular los puntos de función y se puede calcular directamente el esfuerzo.

Lenguaje	LOC	Porcentaje
Java	1510	61.43 %
Python	948	38.57 %
TOTAL	2458	100 %

Tabla 5. LOC del proyecto.

Para el cálculo del esfuerzo se emplean ecuaciones derivadas de la ecuación de software (30 pág. 611):

$$E = \left( \frac{LOC \times B^{0.333}}{P} \right)^3 \times \frac{1}{t^4}$$

De la cual se derivan dos ecuaciones para el cálculo de tiempo mínimo y el esfuerzo. La ecuación del tiempo mínimo de desarrollo en meses es la siguiente:

$$t_{min} = 8.14 \times \left( \frac{LOC}{P} \right)^{0.43}$$

Con el tiempo mínimo se puede aplicar la ecuación derivada del esfuerzo cuyo resultado será el esfuerzo en personas-mes:

$$E = 180 \times B \times t^3$$

Para el cálculo del tiempo mínimo, el valor de P será 10000, como se recomienda para un *software* de telecomunicaciones y sistemas recomendado en (30 pág. 611).

$$t_{min} = 8.14 \times \left( \frac{2458}{10000} \right)^{0.43} = 4.45 \text{ meses} = 0.37 \text{ años}$$

Para el cálculo del esfuerzo se escoge el valor de B = 0.28 recomendado en (30 pág. 611) y el tiempo mínimo en años.

$$E = 180 \times 0.28 \times 0.37^3 = 2.55 \text{ personas - mes}$$

El coste de personal se obtiene con la siguiente ecuación:

$$\text{Coste personal} = \text{esfuerzo} \times \text{tarifa de mano de obra}$$

La tarifa se ha determinado empleando la calculadora salarial que ofrece la empresa Hays en su página web (31). Los criterios para calcular la tarifa han sido los de un desarrollador de Python con 5 años de experiencia.

El salario medio bruto mensual de un desarrollador según Hays es de 3000€. Por tanto, el coste de personal sería:

$$\text{Coste personal} = 2.55 \text{ p.m.} \times 3000 \text{ €} = 7650 \text{ €}$$

## 5.2. Estimación de costes de adquisición y uso

A continuación, se realiza una estimación de los costes que le supondría a una empresa la adquisición y uso del sistema de reconocimiento facial.

El plan de Azure empleado para la elaboración de este proyecto es gratuito; sin embargo, las limitaciones del plan gratuito no son recomendables para un uso empresarial, lo que implica unos gastos por uso.

Existen varios tramos de facturación en función del número de transacciones que se realicen en un mes. Los precios se pueden consultar en:

<https://azure.microsoft.com/es-es/pricing/details/cognitive-services/face-api/?cdn=disable>

A continuación, se propone un ejemplo para ilustrar el coste de adquisición y uso. Una empresa de 200 empleados quiere emplear el sistema de reconocimiento facial para registrar la hora de entrada y salida de sus empleados, es decir, un moderno sistema de fichaje. La empresa tiene una entrada y una salida separadas físicamente, un departamento de recursos humanos con un teléfono Android y un ordenador que funciona como base de datos SQL.

Para instalar el sistema necesita dos dispositivos que puedan albergar una aplicación basada en FCModule, es decir dos Raspberry Pi y dos PiCamera.

Por cada empleado se toman tres fotos para realizar el entrenamiento. Agregar a una persona realiza las siguientes transacciones:

- reconocer la foto para comprobar que no existe la persona.
- crear la persona.
- agregar una foto a la persona.
- entrenar el grupo.

Es decir, un total de 12 transacciones por empleado, 2400 transacciones para poner en marcha el sistema.

Durante la jornada, los 200 empleados entrarán al menos una vez y saldrán otra vez en condiciones normales. Si se añade la pausa para comer como evento de fichaje, se deben añadir otros dos reconocimientos por empleado, es decir, el reconocimiento se ejecutará al menos 800 veces al día.

Por cada reconocimiento se realizan dos transacciones, envío de la imagen para identificación y búsqueda del nombre real.

En total 1600 transacciones al día, 32000 transacciones al mes.

Se puede desglosar el gasto total en gasto inicial (Tabla 6) y gasto mensual (Tabla 7):

Concepto	Unidades	Precio unidad	Total
Raspberry Pi	2	40 €	80 €
PiCamera	2	30 €	60 €
Entrenamiento	2400	0.844 € / 1000 transacciones	2.53 €
<b>TOTAL</b>			<b>142.53 €</b>

Tabla 6. Coste inicial.

Concepto	Unidades	Precio unidad	Total
Transacciones	32000	0.844 € / 1000 unidades	27.01 €
Almacenamiento de imágenes	600 (3 imágenes por empleado)	0.211 € / 1000 imágenes	0.211 €
TOTAL			27.22 €

*Tabla 7. Coste mensual.*

En estas condiciones el coste mensual es más que asumible por una empresa con las condiciones descritas, lo cual hace que este sistema de reconocimiento sea una opción viable de bajo coste.

## 6. Conclusiones y trabajos futuros

Este capítulo sirve como cierre de la memoria de este proyecto. Tras el desarrollo de las aplicaciones y la realización de pruebas, se exponen las conclusiones obtenidas recordando los objetivos iniciales expuestos en el Capítulo 1 y el grado de cumplimiento de estos.

Este capítulo también recoge ideas para futuras ampliaciones y oportunidades de mejora.

### 6.1. Conclusiones

Tras el éxito de las pruebas se puede concluir que el sistema cumple con alto grado de satisfacción con su uso previsto y en ocasiones se han superado las expectativas iniciales, como en el caso de reconocimiento de varios sujetos tan solo empleando una imagen de entrenamiento.

Se ha obtenido un conjunto de herramientas útiles que permiten el reconocimiento facial de manera sencillo y portátil. El empleo de la Raspberry Pi para instalar la biblioteca y las aplicaciones ha sido una buena elección. Permite el uso tanto como equipo de escritorio con la aplicación FacePi, como equipo portátil en el que no es necesario intervenir directamente gracias a FaceBT, permitiendo acoplar la Raspberry Pi a un robot de telepresencia o emplear la Raspberry Pi de manera autónoma.

La aplicación para Android FacePal funciona satisfactoriamente como complemento a las demás aplicaciones y permite añadir nuevas personas y caras cómoda y eficazmente.

A su vez, la biblioteca FCModule y la documentación generada a partir de sus clases, permitirá a otros desarrolladores usar sus funciones fácilmente para desarrollar sus propias aplicaciones adaptadas a sus proyectos.

Se ha conseguido un sistema **rápido**. La velocidad de las aplicaciones derivadas de FCModule es adecuada para el uso previsto. Con dos rostros en pantalla, el tiempo total desde que la aplicación captura la imagen, realiza los recortes, los envía, recibe

respuestas, las procesa, etiqueta las caras y guarda el archivo es de media 5 segundos, lo cual es aceptable.

FacePal realiza una comunicación rápida una vez obtiene la imagen con la cara recortada. De media, tarda 3 segundos desde que se pulsa el botón RECONOCER y se obtiene la respuesta con la identificación del sujeto. Desde que se pulsa el botón AÑADIR CARA, la aplicación tarda 3 segundos en ejecutar el reconocimiento y otros 2 segundos en añadir la cara. Al añadir una persona, el tiempo es de 3 segundos para reconocer y 3 segundos para añadir los datos. Estos son tiempos asumibles y no suponen que el usuario tenga que estar pendiente demasiado tiempo del teléfono móvil.

Sin embargo, FacePal presenta un cuello de botella en la detección de caras en una imagen cuando el sujeto se encuentra alejado. Cuando la imagen es tomada muy cerca de la cara del sujeto los tiempos de carga son de 4 segundos de media, con una resolución de 8 megapíxeles. El problema es que, si la imagen con la misma resolución contiene un rostro pequeño, es decir, que el sujeto está lejos de la cámara, el algoritmo de detección facial tarda más en adquirir el rostro. Durante las pruebas realizadas, algunas caras tardaron hasta 14 segundos en ser cargadas. Se debe entender por tiempo de carga desde que el usuario selecciona la imagen de la galería hasta que aparece por pantalla la imagen recortada.

Los motivos de esta tardanza se atribuyen a la biblioteca de Mobile Vision. Si el reconocimiento está ligado a la creación de una actividad Android en condiciones desfavorables (rostro lejano, alta resolución) el tiempo de carga es de 3 segundos, pero en el resto de los casos (cargar imágenes dentro de una actividad Android existente) puede ser hasta 4 veces más. No se entiende muy bien por qué ocurre este evento. De hecho, existen referencias de otros desarrolladores que se han encontrado el mismo problema (32) (33) (37). Hasta que una alternativa sea aceptable, se deben evitar las imágenes lejanas.

Se ha conseguido un sistema **eficaz**. Tanto FacePal como las aplicaciones basadas en FCModule realizan un recorte de las caras en una imagen, reduciendo en gran medida la información irrelevante de la imagen. A su vez, a Azure se manda solo el rostro, lo que supone un ahorro de computación y menor consumo de datos en el envío. FCModule en

ocasiones reconoce algunos objetos como rostros, como se ve en la Figura 48. Durante el desarrollo de las aplicaciones, esto se ha tenido en cuenta para que estos artefactos no sean mostrados como sujeto “Desconocido” en el resultado final.



Figura 48. Objetos identificados como rostros.

Estos falsos positivos son enviados a Azure, obteniéndose en todos los casos una respuesta negativa como rostro no detectado. A partir de este punto, las aplicaciones desarrolladas no realizan más operaciones con estos elementos.

Se han desarrollado aplicaciones de **fácil uso**. FacePal y FacePi disponen de interfaces sencillas, funcionales y sin complejidad innecesaria. De esta forma, no hay opciones adicionales ni ningún elemento que haga al usuario final dudar o no entender el funcionamiento de las aplicaciones. En las pruebas se dio la aplicación a dos personas no relacionadas con el desarrollo y no tuvieron ningún problema en manejarla.

FaceBT permite una conexión mediante Bluetooth sencilla y tan solo requiere el envío de comandos mediante un terminal, tarea que puede ser automatizada dentro del dispositivo al que vaya emparejado.

FaceRecon no dispone de un interfaz como tal, solo emite mensajes por pantalla y almacena imágenes, por lo que el usuario no debe preocuparse por nada en absoluto. No se debe olvidar que FaceRecon es un ejemplo de uso de la biblioteca FCModule y que como ejemplo es configurable de muchas maneras dependiendo del uso final.

Se han conseguido aplicaciones de **fácil incorporación**. FacePal se puede instalar fácilmente en cualquier dispositivo con sistema Android moderno.

Las aplicaciones basadas en FCModule están preinstaladas en la Raspberry Pi de este proyecto. La instalación de todo el conjunto tan solo necesita el volcado de una imagen en una tarjeta microSD y la configuración de la red Wifi deseada. A partir de ese momento puede empezar a usarse FacePi como aplicación de escritorio o directamente ser acoplado a otro proyecto y emplear las funciones de FaceBT mediante conexión Bluetooth.

Se ha logrado que las aplicaciones sean de **fácil mantenimiento**. El código de las aplicaciones está debidamente comentado y se ha usado la compartmentalización siempre que ha sido posible, creando funciones fácilmente sustituibles o actualizables sin tener que alterar el resto del programa.

En FacePal se ha separado claramente el interfaz gráfico de las funciones de recorte, comunicación y reconocimiento. Si es necesario cambiar el diseño de la aplicación, esto no afectará a su funcionalidad.

Finalmente, se ha conseguido obtener un sistema de reconocimiento facial de **bajo coste**. En cuanto al *software*, todos los programas y lenguajes empleados son gratuitos. El *hardware* empleado es de bajo coste, la combinación Raspberry Pi y PiCamera tiene un precio inferior a 70 €, que es un coste asumible en la práctica totalidad de los proyectos.

## 6.2. Futuras ampliaciones

El sistema tiene mucho potencial para ser ampliado y mejorado. El principal foco de atención es la mejora de la rapidez en el recorte de rostros de FacePal. Es posible que existan bibliotecas más rápidas que Mobile Vision, o incluso podría ser interesante fabricar una biblioteca propia o funciones auxiliares a FacePal.

Otra opción interesante sería implementar la autorrotación de imágenes desde la galería o rotación en el área donde se carga la imagen en FacePal. Se ha intentado la autorrotación consiguiendo los metadatos de las imágenes que almacena Android y creando una función de rotación automática en función de la orientación original. Sin

embargo, existe un error (35) en algunos teléfonos de Android que impide que las fotografías se tomen de acuerdo a la rotación de la cámara y, por tanto, algunos metadatos guardan la orientación correcta si bien la imagen queda rotada 90º con respecto a cómo se tomó.

Una posibilidad es cargar la imagen, y agregar una función para que al pulsar con el dedo la imagen rote 90º y realice la búsqueda de rostros. Tras cuatro pulsaciones, si no se ha detectado ningún rostro el sistema debería desaconsejar al usuario seguir usando esa imagen.

Otras mejoras relacionadas con la eficiencia y el ahorro de coste pueden ser la revisión del código para localizar puntos donde se reduzca el número de transferencias a Azure.

Como se ha visto anteriormente, Azure no devuelve el nombre de las personas al identificar un rostro, solo devuelve su ID. En este proyecto es necesario buscar la lista de individuos del grupo y contrastar cada ID con el ID deseado, lo que requiere una transferencia por cada reconocimiento y el procesamiento de la lista de personas añadidas. Si la lista es grande, esto puede ser muy ineficaz. La mejor manera de gestionarlo sería crear una base de datos que relacionase nombres e IDs, de esta manera se consultaría la base de datos y no a Azure cuando se quiera obtener el nombre asignado a una persona. Esta base de datos podría ser de cualquier tipo, incluso un fichero JSON en el almacén interno del dispositivo, para evitar complejidad innecesaria. Se podrían agregar funciones de verificación de consistencia periódicas u otras opciones.

FacePal y FacePi puede ampliarse para realizar gestiones mayores como eliminar usuarios, crear otros grupos de personas, etc. Se necesitaría modificar la interfaz gráfica y añadir las funciones pertinentes. Otro tipo de aplicación que podría derivarse de FCMModule sería la de crear un servicio web combinando HTML y PHP ejecutando scripts de Python en el lado del servidor. De esta forma, el usuario podría subir una imagen al servidor, el servidor realizaría el procesamiento y el usuario obtendría la misma imagen con las caras etiquetadas.

Otra futura ampliación sería añadir comandos adicionales para indicar a FaceBT cuál es el nombre de la red inalámbrica a la que se desea conectar y la contraseña de la misma. De esta manera, se evitaría totalmente tener que emplear la Raspberry Pi como equipo de escritorio solo para configurar la red Wifi.

FCModule ofrece mucho potencial. Usando sus funciones se pueden crear aplicaciones que realicen una o varias de las siguientes operaciones:

- devolución de nombres, confianza, coordenadas.
- devolución solo de caras recortadas.
- envío de información por correo electrónico.
- rostros mostrados por pantallas pequeñas en robots de patrulla.
- lectura de los nombres en voz digital (*text to speech*).
- disparar una alarma ante ciertas detecciones.
- sistema de apertura de puertas.
- fichaje de personal.

Las opciones son muy amplias pero esta parte sí que requiere el desarrollo por parte de personal con mayor conocimiento de *software* y de programación.

La flexibilidad que proporciona la combinación de emplear FacePal y las aplicaciones que usan FCModule, dan lugar a multitud de usos. Un desarrollador crearía aplicaciones usando FCModule y usuarios con poco conocimiento de desarrollo entrenarían el sistema con FacePal. El ejemplo del sistema de fichaje moderno propuesto en el Capítulo 5 es una buena demostración de este uso conjunto.

## Bibliografía

1. Bledsoe, Woodrow y Chan, Helen. *A Man-Machine Facial Recognition System-Some Preliminary Results.* Palo Alto, California. : Technical Report PRI 19A, Panoramic Research, Inc., 1965.
2. Bledsoe, Woodrow. *Semiautomatic Facial Recognition.* Menlo Park, California : Technical Report SRI Project 6693, Stanford Research Institute, 1968.
3. Recognition, Woodrow Bledsoe Originates of Automated Facial.  
<http://www.historyofinformation.com/expanded.php?id=2495>.
4. Cámaras de reconocimiento facial llegarán el 31 de enero a Panamá.  
[https://www.prensa.com/yalena\\_ortiz/Camaras-reconocimiento-facial-llegaran-Panama\\_2\\_3302689702.html](https://www.prensa.com/yalena_ortiz/Camaras-reconocimiento-facial-llegaran-Panama_2_3302689702.html).
5. Park, Madison y Price, Sabrina. How did U.S. confirm the body was bin Laden's?  
<http://edition.cnn.com/2011/HEALTH/05/02/bin.laden.body.id/index.html>.
6. Black Book.  
<http://gaming.nv.gov/index.aspx?page=72>.
7. *DeepFace: Closing the Gap to Human-Level Performance in Face Verification.* Taigman, Yaniv, y otros. 2014, facebook research.
8. Windows Phone ha muerto.  
<https://www.xataka.com/moviles/windows-phone-ha-muerto-joe-belfiore-de-microsoft-confirma-el-secreto-a-voces>.
9. Web oficial Raspberry Pi.  
<https://www.raspberrypi.org/>.
10. Revista Make:.  
<https://makezine.com/>.
11. How-to Geek.  
<https://howtogeek.com/>.
12. Web Amazon Rekognition.  
<https://aws.amazon.com/rekognition/>.
13. Face++.  
<https://www.faceplusplus.com/>.

14. Kairos.

<https://www.kairos.com/>.

15. Google Vision.

<https://cloud.google.com/vision/>.

16. IBM Watson.

<https://www.ibm.com/watson/>.

17. Microsoft Azure.

<https://azure.microsoft.com>.

18. OpenCV.

<https://opencv.org/>.

19. Mobile Vision.

<https://developers.google.com/vision/introduction>.

20. ML Kit.

<https://developers.google.com/ml-kit/>.

21. Camera Module - Raspberry Pi Documentation.

<https://www.raspberrypi.org/documentation/hardware/camera/README.md>.

22. Android developer.

<https://developer.android.com/index.html>.

23. Face API documentation.

<https://docs.microsoft.com/en-us/azure/cognitive-services/face/>.

24. Cascade classifier.

[https://docs.opencv.org/3.0-beta/doc/tutorials/objdetect/cascade\\_classifier/cascade\\_classifier.html](https://docs.opencv.org/3.0-beta/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html).

25. Face Detection using Haar Cascades .

[https://docs.opencv.org/3.3.0/d7/d8b/tutorial\\_py\\_face\\_detection.html](https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html).

26. Bradski, Gary y Kaehler, Adrian. *Learning OpenCV*. s.l. : O'Reilly Media, 2008.

27. Qt.

<https://www.qt.io/>.

28. PyQt5.

<http://pyqt.sourceforge.net/Docs/PyQt5/>.

29. Android authority.

<https://www.androidauthority.com/develop-android-apps-languages-learn-391008/>.

30. RFCOMM protocol.

[https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth\\_info/rfcomm.html#Multiple%20Emulated%20Serial%20Ports](https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/rfcomm.html#Multiple%20Emulated%20Serial%20Ports).

31. httpclient-android.

<https://github.com/smarek/httpclient-android>.

32. Bluetooth Serial Terminal.

<https://www.microsoft.com/en-us/p/bluetooth-serial-terminal/9wzdncrdfst8>.

33. Pressman, Roger. *Ingeniería del Software. Un enfoque práctico*. s.l. : McGraw-Hill, 2010.

34. Hays.

<http://www.hays.es>.

35. Mobile vision face detector slow in other function than onCreate.

<https://stackoverflow.com/questions/47199329/mobile-vision-face-detector-slow-in-other-function-than-oncreate>.

36. mobile vision API takes too long to detect face.

<https://stackoverflow.com/questions/43727348/mobile-vision-api-takes-too-long-to-detect-face>.

37. Google Mobile Vision: Poor FaceDetector performance without CameraSource.

<https://stackoverflow.com/questions/34132444/google-mobile-vision-poor-facedetector-performance-without-camerасource>.

38. Wrong picture orientation on Samsung devices .

<https://github.com/google/cameraview/issues/22>.



## Anexo A – Contenido del PFG

El proyecto se ha dividido en varios directorios para compartmentalizar la información de manera más eficaz. A continuación, se listan los directorios y su contenido:

### **build**

Contiene el archivo facepal.apk necesario para la instalación de FacePal.

### **docs**

Contiene la memoria de este proyecto y manuales.

### **setup**

Contiene la imagen con todas las aplicaciones para volcar en una microSD e insertar en la Raspberry Pi.

### **src**

Contiene el código fuente de la biblioteca y de cada aplicación desarrollada.

### **test**

Contiene el código fuente y las imágenes empleadas para el entrenamiento y realización de las pruebas Bill Gates vs. Ben Linus.



## Anexo B – Manual de instalación

### Raspberry Pi

#### Requisitos mínimos:

- Raspberry Pi 3 modelo B.
- Raspberry Pi Camera Module v2 con cable.
- Teclado USB.
- Ratón USB.
- Pantalla HDMI.
- Fuente de alimentación externa con conexión micro USB.
- Conexión a internet mediante Wifi o Ethernet.
- Tarjeta microSD de 16 GB.

#### Preparación de la tarjeta SD

En Windows, instalar el programa Win32 Disk Imager, disponible en:

<https://sourceforge.net/projects/win32diskimager/>

Abrir el programa y seleccionar la imagen que se encuentra en:

[PFG/setup/rpi.img](#)

Seleccionar en *Device* la unidad de destino donde se encuentre la tarjeta microSD.

Pulsar *Write*.

Una vez terminado el proceso, la tarjeta estará lista para su uso con la Raspberry Pi.

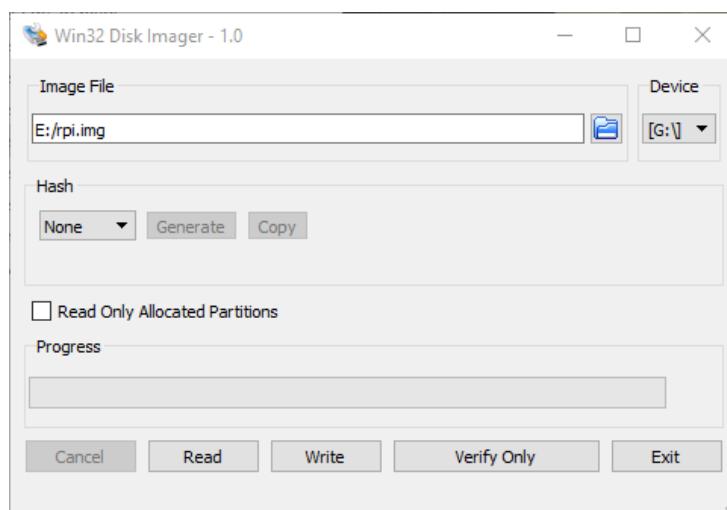


Figura 49. Captura de pantalla de Win32 Disk Imager.

## Instalación

Conectar el extremo del cable de la cámara en el bus dedicado en la Raspberry Pi, como se muestra en la Figura 50.



Figura 50. Conexión de la cámara con la Raspberry Pi.

Conectar ratón, teclado y pantalla en sus puertos correspondientes.

Insertar la tarjeta SD con la imagen del PFG instalada en la ranura correspondiente.

Conectar la fuente de alimentación al conector micro USB en el lateral.

## Configuración

Configurar la red Wifi, haciendo *click* en el icono Wifi, como se ve en la Figura 51.

Seleccionar la red correspondiente e introducir la clave de seguridad.

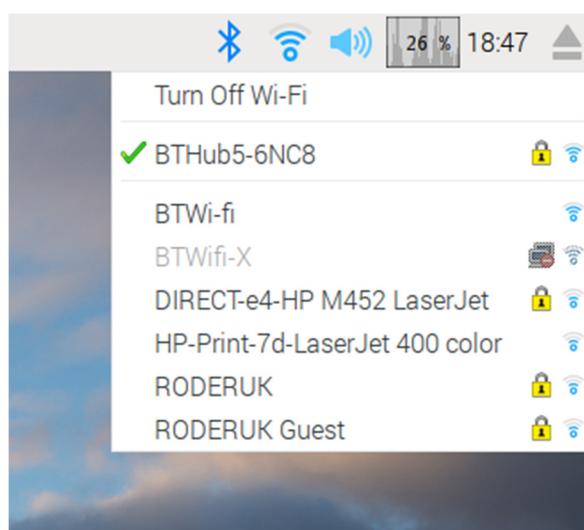


Figura 51. Conexión a red Wifi.

## **FCModule**

FCModule está preinstalado en esta configuración de Raspberry Pi.

Para instalarlo en otro sistema, copiar el contenido de la carpeta [PFG/src/FCModule](#) en el destino deseado.

En el directorio donde se ha copiado FCModule, ejecutar el siguiente comando:

```
python setup.py install
```

Este comando requiere permiso de administrador y la versión de Python 3.4 o superior.

Todas las dependencias de FCModule son instaladas junto a la ejecución de este comando en caso de que no esté previamente instaladas.

## **FacePal**

### **Requisitos mínimos:**

- Dispositivo Android con Android 6 (Marshmallow).

### **Instalación**

Copiar el archivo FacePal.apk situado en la carpeta [PFG/build/](#) en la tarjeta SD del dispositivo deseado.

En el dispositivo, ejecutar el archivo FacePal.apk.

Una vez instalado, aparecerá el ícono de FacePal en la lista de aplicaciones.

### **Configuración**

Seleccionar el botón de menú en la parte superior derecha de la aplicación.

Seleccionar la opción “Configuración” y después “General”.

En esta pantalla (Figura 52) se puede seleccionar y cambiar los valores del grupo de personas, clave de suscripción y servidor.

## Sistema de reconocimiento facial para dispositivos móviles y robot de telepresencia

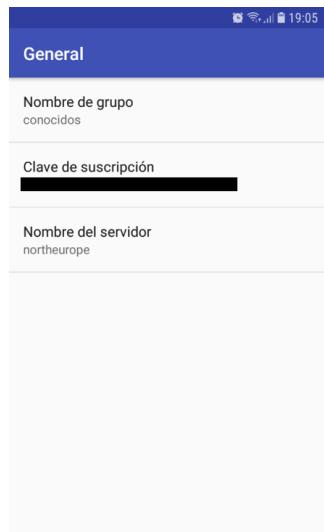


Figura 52. Menú de configuración de FacePal.

## Anexo C – Manual de usuario

### FCModule

En el script de Python, incluir las siguientes líneas:

```
import FCModule.faceCrop as fc
import FCModule.FCTools as FCT
from FCModule.face import Face
```

Consultar el Manual de Clases ([PFG/docs/manualClases/html/index.html](http://PFG/docs/manualClases/html/index.html)) para obtener información sobre las funciones de cada módulo.

### FacePi

El programa se encuentra preinstalado en la Raspberry Pi. Para iniciar la aplicación, ejecutar el comando:

```
python /home/pi/Desktop/PFG/src/FacePi/facepi.py
```

La interfaz gráfica se compone de tres partes (Figura 53):

1. Área de visualización de imagen
2. Botones de comando
3. Consola de texto

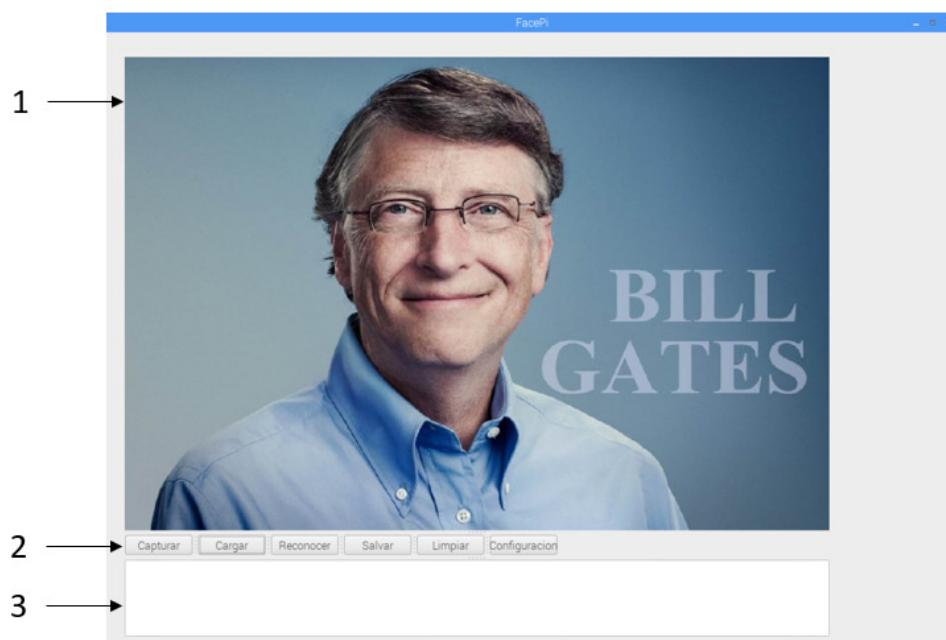


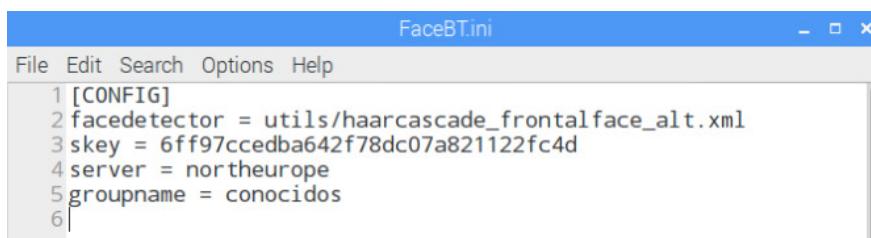
Figura 53. Pantalla principal de FacePi.

Botones:

- Botón Capturar: captura una imagen empleando la cámara de la Raspberry Pi y la muestra por pantalla.
- Botón Cargar: carga una imagen previamente salvada.
- Botón Reconocer: ejecuta el reconocimiento y la identificación de caras.
- Botón Salvar: guarda la imagen mostrada en pantalla en destino deseado.
- Botón Limpiar: limpia la consola.
- Botón Configuración: permite cambiar los valores de:
  - o clave de suscripción.
  - o servidor.
  - o grupo de personas.

### FaceBT

El programa se encuentra preinstalado en la Raspberry Pi y se autoejecuta al encenderla. Para cambiar la configuración de la aplicación, editar el archivo /home/pi/Desktop/PFG/src/FaceBT/FaceBT.ini (ubicación de filtro, clave de suscripción, servidor y grupo de personas).



```
FaceBT.ini
File Edit Search Options Help
1 [CONFIG]
2 facedetector = utils/haarcascade_frontalface_alt.xml
3 skey = 6ff97cceda642f78dc07a821122fc4d
4 server = northeurope
5 groupname = conocidos
6
```

Figura 54. Archivo de configuración FaceBT.

Para usar FaceBT es necesaria una aplicación de terminal capaz de comunicarse mediante el puerto de serie por Bluetooth y una conexión de Bluetooth.

Si solo se desea usar FaceBT, no es necesario teclado, ratón o pantalla.

Emparejar el dispositivo con la Raspberry Pi. En la búsqueda de dispositivos aparecerá como “FaceBT”.

Abrir la aplicación de terminal y conectar con el puerto de serie del dispositivo emparejado.

Los comandos que FaceBT acepta son los siguientes:

- `getTexto`: devuelve información en forma de texto de la imagen capturada y de los rostros identificados.
- `getJSON`: devuelve un archivo en formato JSON con la información de la imagen capturada y de los rostros identificados.
- `getImage`: devuelve la imagen capturada con los rostros identificados.
- `apagar`: cierra la aplicación y apaga la Raspberry Pi.

Nota: la aplicación de terminal debe enviar el carácter de nueva línea al final de cada envío, para que el comando sea reconocido.

Para el uso portátil de la Raspberry Pi y acceso a las funciones de esta aplicación, solo es necesario la Raspberry Pi con la cámara correspondiente y una batería externa conectada. Se recomienda una batería externa con una salida de 5V y 2 Amperios para el correcto funcionamiento del sistema.

### **FaceRecon**

El programa se encuentra preinstalado en la Raspberry Pi. Para iniciar la aplicación, ejecutar el comando:

```
python /home/pi/Desktop/PFG/src/FaceRecon/FaceRecon.py
```

Para cambiar la configuración de la aplicación, editar el archivo `/home/pi/Desktop/PFG/src/FaceRecon/FaceRecon.ini` (ubicación de filtro, clave de suscripción, servidor y grupo de personas).

El programa se ejecuta continuamente, sin necesidad de intervención del usuario.

A intervalos fijos toma una imagen, identifica rostros si los hubiese y almacena cada imagen en el directorio `/home/pi/Desktop/PFG/src/FaceRecon/img`.

La combinación de teclas Control+C termina la aplicación.

### **FacePal**

La pantalla de inicio dispone de un área de visualización de imagen y de tres botones (Figura 55).



Figura 55. Pantalla principal de FacePal

- Botón BUSCAR: abre la galería de imágenes y permite seleccionar una imagen. Ejecuta la detección de rostro y muestra por pantalla el recorte.
- Botón RECONOCER: realiza la identificación de la cara cargada en la aplicación.
- Botón ENTRENAR: abre la pantalla de entrenamiento.

La pantalla de entrenamiento (Figura 56) consiste en cuatro partes:

- Área de texto.
- Área de imagen.
- Botón AÑADIR CARA: añade la cara cargada a una persona existente.
- Botón AÑADIR PERSONA: añade una nueva persona al sistema, con el nombre indicado en el área de texto y la imagen cargada en pantalla.

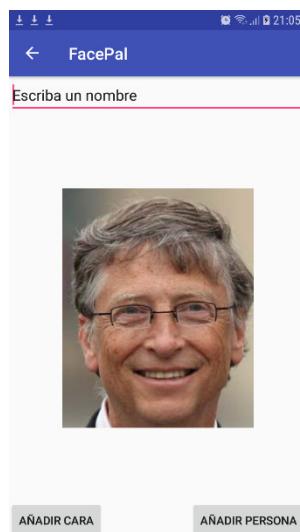


Figura 56. Pantalla de entrenamiento de FacePal.

## Anexo D – Materiales empleados

### **Plataformas *hardware*:**

Raspberry Pi 3 modelo B: implementación de FCModule, FaceBT, FacePi y FaceRecon.

Samsung Galaxy S3: implementación de FacePal.

Samsung Galaxy A3 2017: pruebas secundarias de FacePal.

AMD Phenom 9650 Quad-Core 8 GB: equipo de escritorio para Android Studio y elaboración de diagramas y memoria.

Surface Pro: pruebas de comunicación por Bluetooth.

### **Servicios online:**

Microsoft Azure: reconocimiento facial.

Github: control de versiones.

### **Software:**

Android Studio: desarrollo de aplicaciones Android.

Bluetooth Serial Terminal: terminal de serie para Windows.

Dia: desarrollo de esquemas y diagramas.

Doxygen: Generación automática de documentación.

IDLE (Python 3.4): desarrollo en Python.

Notepad++: edición de código.

Powerpoint: elaboración de figuras.

Raspbian GNU/Linux 8: distribución empleada en Raspberry Pi.

Serial Bluetooth Terminal: terminal de serie para Android.

Windows 10: sistema operativo.

Word: elaboración de la memoria.

### **Bibliotecas *software*:**

PyQt: *binding* de la biblioteca gráfica Qt para la realización de interfaces gráficos.