



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 MOVILES Y ROBOT DE
TELEPRESENCIA**

Alonso Rodrigo Serrano Alvarez

Dirigido por: Ángel Pérez de Madrid y Pablo

Codirigido por: Carolina Mañoso Hierro

Curso: 2017/2018



SISTEMA DE RECONOCIMIENTO FACIAL PARA DISPOSITIVOS MOVILES Y ROBOT DE TELEPRESENCIA

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

Realizado por: Alonso Rodrigo Serrano Alvarez

Dirigido por: Ángel Pérez de Madrid y Pablo

Codirigido por: Carolina Mañoso Hierro

Fecha de lectura y defensa:

Resumen

La tecnología de reconocimiento facial ha dejado de ser exclusiva para grandes empresas. 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.

En primer lugar, en este proyecto se han analizado las distintas tecnologías software y hardware existentes que puedan ser empleadas en la creación de un sistema de reconocimiento facial. A continuación, se ha desarrollado un sistema de reconocimiento facial basado en 3 componentes: sistema de reconocimiento facial en continuo, sistema de entrenamiento mediante el uso de un teléfono móvil y servicio de computación. Empleando herramientas software y hardware de uso común y bajo coste como los lenguajes Python y Java o la plataforma Raspberry Pi, se han implementado dos aplicaciones que adquieren imágenes, reconocen rostros y, mediante el servicio externo Microsoft Azure, realizan su identificación usando clasificadores personalizados. Finalmente se han desarrollado pruebas para la verificación del correcto funcionamiento del sistema.

Uno de los principales objetivos de este trabajo ha sido crear una aplicación de apoyo, fácil de emplear, que no requiere conocimientos avanzados de software por parte del usuario. Entre sus posibles usos está el de apoyo a un sistema de vigilancia, como asistente en reuniones para ayudar a reconocer a los participantes, el acoplamiento a un robot de telepresencia, etc.

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

Facial recognition technology is no longer a big corporation's exclusive. 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.

First, in this project several different software and hardware existing technologies that can be used in the creation of a facial recognition system have been analysed. Then, a facial recognition system has been created using 3 base components: a continuous facial recognition system, a mobile phone-based training system and a computing service. Using common and low-cost software and hardware tools such as Python, Java or the Raspberry Pi platform, two applications have been coded that can acquire images of faces and proceed to their identification using custom classifiers via Microsoft Azure as an external service. Finally, a set of tests have been developed in order to verify the system correctness.

One of the main objectives of this project has been the creation of an auxiliary application, easy to use, that does not require advanced software knowledge from the user. Some of the many uses for this system include security system's aid, an assistant to identify persons in meetings, as part of a telepresence robot, etc.

Keywords

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

Índice

1. Introducción	11
1.1. Alcance del proyecto	11
1.2. Metas.....	11
1.3. Estructura del proyecto.....	12
2. Hardware	15
2.1. Elección del sistema	15
2.2. Raspberry Pi 3.....	20
2.3. Android Phone.....	22
3. Reconocimiento facial.....	23
3.1. Elección del sistema	23
3.2. Servicio principal: Azure	28
3.3. Servicio auxiliar: OpenCV	34
4. Software.....	39
4.1. Introducción	39
4.2. Elección del lenguaje.....	39
4.3. FaceRecon	40
4.4. FacePal	49
5. Entrenamiento.....	61
5.1. Comprobación con imágenes existentes	61
5.2. Entrenamiento con imágenes existentes y reconocimiento en continuo	70
5.3. Entrenamiento y reconocimiento en FacePal	72
5.4. Entrenamiento y reconocimiento FacePal+FaceRecon.....	75
6. Conclusiones	77
6.1. Objetivos alcanzados.....	77
6.2. Futuras ampliaciones	82
Bibliografía.....	85
Listado de siglas, abreviaturas y acrónimos	87
Anexo 1 – Materiales y métodos	89
Anexo 2 - Preparación de Raspbian	91

Lista de figuras y tablas

Figura 1 Esquema general del proyecto.	12
Figura 2 Sistema de telepresencia con iPad.....	16
Figura 3 Raspberry 3 Modelo B.	17
Figura 4 Arduino UNO.....	19
Figura 5 Cámara de Raspberry Pi.....	21
Figura 6 Panel de Azure al darse de alta en el servicio QeQ.....	29
Figura 7 Ejemplo de organización de entidades.	31
Figura 8 Ejemplo de ventana deslizante	36
Figura 9 Ejemplo de filtros de Haar.....	36
Figura 10 Cascada de clasificadores.	37
Figura 11 Descripción de FaceRecon.....	41
Figura 12 Ejecución de FaceRecon.	42
Figura 13 Diagrama de clases de FaceRecon.	43
Figura 14 Ejemplo de esquinas del rectángulo.	45
Figura 15 Esquema general de FacePal.....	49
Figura 16 Instalación de FacePal.....	52
Figura 17 Pantalla principal de FacePal.....	56
Figura 18 Menu configuración.....	57
Figura 19 Pantalla Configuración.	57
Figura 20 Retratos Bill Gates	62
Figura 21 Imagen de Bill Gates empleada como caraTest.	62
Figura 22 Ejemplo de ejecución durante prueba Bill vs. Bill.....	63
Figura 23 Grafica Bill vs. Bill.	63
Figura 24 Caras elegidas de Ben Linus	64
Figura 25 Ben Linus (Michael Emerson)	65
Figura 26 Ejecución del caraTest de Ben Linus antes del entrenamiento....	65
Figura 27 Ejecución de los 10 rostros de Ben Linus.	66
Figura 28 Ejecución del caraTest de Ben Linus tras el entrenamiento.	66
Figura 29 Gráfica Ben vs. Ben.	67
Figura 31 Imitador de Bill Gates	69
Figura 32 Ejecución de la prueba.....	69
Figura 33 Fotos de entrenamiento de "Admin".....	70

Figura 34 Primer reconocimiento de Admin.	71
Figura 35 Imagen de Admin con iluminación alternativa.	71
Figura 36 Akbar Espaillat.	72
Figura 37 Creación de persona Bastian Schiffthaler.	73
Figura 38 Cara agregada a Bastian Schiffthaler.	73
Figura 39 Javier Alvarez.	74
Figura 40 Reconocimiento de Akbar y Admin.	75
Figura 41 Reconocimiento de Javi y Admin.	75
Figura 42 Reconocimiento de Bastian.	76
Figura 43 Objetos identificados como rostros.	78
Figura 44 Conexión de cámara	91
 Tabla 1 Comparación entre plataformas de hardware.	 20
Tabla 2 Comparativa entre servicios de reconocimiento facial.	28
Tabla 3 Resultado de la prueba Bill Gates vs. Ben Linus	68
Tabla 4 Resultados de la prueba Bill Gates vs. Bill Gates Impersonator.	69

1. Introducción

1.1. Alcance del proyecto

Este proyecto consiste en el estudio y desarrollo de un sistema de reconocimiento facial en plataformas portátiles de hardware para que pueda ser incorporado en otros proyectos o ser empleados de manera autónoma con otros fines.

La base de este proyecto es la filosofía del IoT (Internet de las cosas), en el que las partes del proyecto emplean Internet como infraestructura de operación.

Este proyecto se divide en 3 fases:

- **Análisis:** inicialmente se realiza una evaluación de las tecnologías existentes, tanto de software como de hardware, al tiempo que se declaran las características que se buscan en ellas. El fin es evaluar cuales de ellas pueden ser incorporada al proyecto. Los resultados de esta fase se encuentran documentados en los capítulos 2 y 3.
- **Desarrollo:** comienza con el planteamiento de la estructura del proyecto y el comienzo de desarrollo de cada una de sus partes. El desarrollo se documenta en el capítulo 4.
- **Evaluación:** una vez que las partes han sido implementadas, se realizan pruebas de cada componente individual y posteriormente pruebas del sistema en conjunto. Durante esta fase se realizan los ajustes necesarios para alcanzar el objetivo final del proyecto. Los resultados y conclusiones se estudian con mayor detalle en los capítulos 5 y 6.

1.2. Metas

El sistema desarrollado debe buscar cumplir las siguientes características:

- **Rapidez:** se deben evitar largas esperas en el reconocimiento facial, debe ser tan rápido como sea posible.

- Eficacia: se debe gestionar la información imprescindible, evitando el envío, recepción o procesamiento de datos irrelevantes.
- Fácil uso: los interfaces deben ser simples e intuitivos, se deben evitar situaciones complicadas para el usuario.
- Fácil incorporación: el sistema debe ser adaptable físicamente a una plataforma de hardware, sin requerir conexiones complejas ni conocimientos avanzados de electrónica, siguiendo la filosofía del Plug and Play (conectar y listo).
- Fácil mantenimiento: todo código que componga el programa debe ser de sencilla interpretación, bien documentado y sin el uso de clases o bibliotecas innecesarias.
- Bajo coste: deben buscarse componentes de bajo coste, pero sin olvidar la calidad y fiabilidad de los mismos.

1.3. Estructura del proyecto

Una visión general del proyecto consiste en un sistema compuesto de 3 partes interconectadas mediante Internet, como puede observarse en la Figura 1.

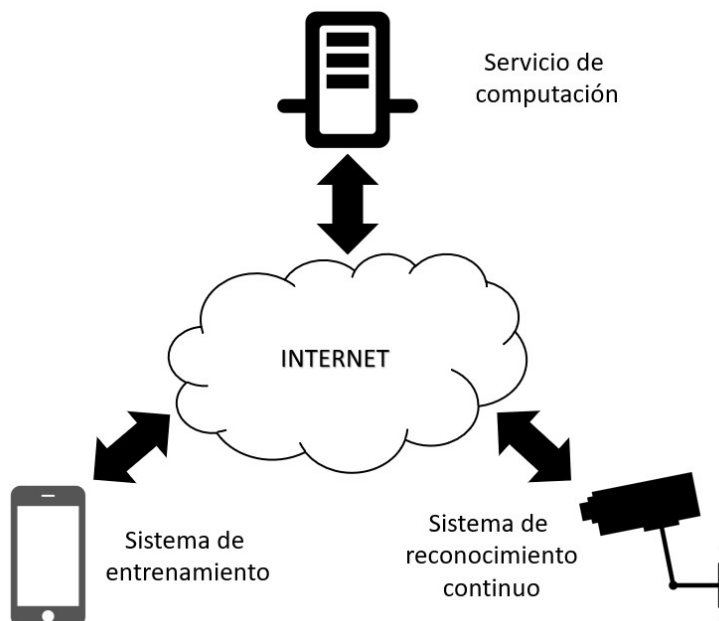


Figura 1 Esquema general del proyecto.

Servicio de computación

El reconocimiento facial implica un alto número de operaciones y cálculos, y cuanto mayor es el número de individuos que el sistema es capaz de reconocer, mayor será el número de cálculos necesarios. Por tanto, los recursos que necesita un sistema de reconocimiento facial pueden ser considerables. El servicio de computación propio o ajeno se dedica al almacenamiento de los datos de los individuos y el reconocimiento de caras aportadas por las otras partes del sistema. Toda la computación realizada debe ser transparente al usuario final.

Sistema de entrenamiento

Este componente permite a los usuarios finales entrenar el sistema mediante el envío de fotografías y nombres, y recepción de mensajes simples. Será de fácil uso y no debe requerir conocimientos de programación. La interfaz debe ser intuitiva y lo más simple posible.

Sistema de reconocimiento en continuo

El fin de este componente es la toma de imágenes a intervalos fijos, la detección de caras e identificación de estas. La salida de este componente puede ser de varios tipos:

- Imagen con las caras identificadas.
- Recortes de las caras en formato de imagen.
- Nombres de los sujetos identificados en texto plano.

Con estos datos el sistema puede adicionalmente:

- almacenarlos en disco duro o en una base de datos local o remota.
- enviarlos por Bluetooth, correo electrónico o a una web.
- mostrarlos por pantalla.

2. Hardware

2.1. Elección del sistema

Una vez planteada la meta del proyecto, se comenzó la búsqueda de la plataforma de hardware adecuada para el uso previsto.

Las características deseables para el proyecto 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

Las plataformas analizadas se describen a continuación:

Equipo de escritorio

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

Altamente modulares, estos equipos permiten configuración a la carta para las características del sistema. Una de las ideas iniciales fue la creación de un servidor capaz de hacer el servicio de reconocimiento facial, por lo que un equipo de escritorio es un buen candidato para este servicio.

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.

Portátiles

Comparado con los equipos de sobremesa, el portátil es más fácilmente acoplable a un sistema de telepresencia. Por el contrario, se sacrifica modularidad, los portátiles tienen menos capacidad de cambiar componentes. Si bien un portátil de gama media tiene recursos suficientes para los requisitos

del sistema, su precio y su dependencia de consumo de energía mediante el uso de baterías y/o cable de conexión a red eléctrica no lo convierten tampoco en un candidato viable.

Tablets

En algunos sistemas 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 robotshop.com).

Existen otros tipos de *tablet* como los basados en Android, con mayor 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 aun 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 adquiriera un móvil de gama alta.

Por estos motivos se descartó el teléfono móvil como núcleo del proyecto, pero no como una parte auxiliar del mismo.

Existen plataformas basadas en 3 sistemas: iOS, Android y Windows Phone.

No se puede valorar iOS para este proyecto puesto que no se dispone 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 sistema más extendidos.

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 (<https://www.raspberrypi.org/>) son un conjunto de diferentes modelos de ordenadores de pequeño tamaño con gran cantidad de componentes integrados, Bluetooth, Wifi, HDMI, USB... (Figura 3).



Figura 3 Raspberry 3 Modelo B (Fuente raspberrypi.org).

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, y su bajo consumo energético, que permite acoplarlo a la batería del robot sin que suponga un consumo excesivo.

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 y dispone de ciertos intérpretes de lenguaje y compiladores ya integrados para su uso (Python, C).

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

Arduino

Es una familia de plataformas de hardware *open source* que permite crear proyectos de hardware y software fácilmente empleando el lenguaje C y conexión sencilla de sensores y efectores (<https://www.arduino.cc/>) (Figura 4). El precio y consumo de los Arduino es muy bajo.



Figura 4 Arduino UNO (Fuente arduino.cc).

Sin embargo, este sistema es muy limitado en cuanto a recursos de memoria, haciendo imposible que sirva para este proyecto. Si bien es cierto que se puede emplear como parte de un robot de telepresencia, esto queda fuera del alcance de este proyecto.

Otras plataformas

Existen en el mercado otra serie de plataformas parecidas a Raspberry Pi:

- ASUS Tinker Board
- Banana Pi M3
- C.H.I.P. Pro Dev Kit
- Intel Joule
- NanoPC-T3
- NanoPi Neo Plus 2
- ODroid Xu4
- Orange Pi Plus 2E
- Pine A64

Varían en cuanto a precio, memoria, velocidad de procesador y componentes integrados. Algunos como el Intel Joule ofrecen características muy interesantes para este proyecto, pero sus precios son muy elevados.

No obstante, ninguno de los citados anteriormente tiene tanta documentación y soporte como la plataforma Raspberry Pi.

En la

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

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

Tabla 1 Comparación entre plataformas de hardware.

2.2. Raspberry Pi 3

Finalmente, se ha escogido esta plataforma por estar a mitad de camino entre el tamaño y consumo de un móvil y los recursos y versatilidad de un ordenador.

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

A continuación, se analizarán diversos aspectos de la Raspberry Pi.

Configuración del sistema

La configuración específica y paquetes instalados en la Raspberry Pi para este proyecto se detallan en el Anexo 2.

Cámara integrada vs. webcam

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 programación

La cámara de la propia Raspberry tiene como ventajas:

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

Su desventaja principal es que al ser poco más que una placa de circuito (Figura 5) no incorpora soporte de serie y en ocasiones es difícil de colocar en una posición correcta y estable, lo cual le da más fragilidad.



Figura 5 Cámara de Raspberry Pi

Por tanto, se decide adquirir una cámara propia de Raspberry modelo Module 2 cuyas características más relevantes son sus 8 Megapixels y video en 1080p.

Una vez conectada, el método para configurarla es muy sencillo.

En una ventana de terminal se ejecuta el comando

```
sudo raspi-config
```

Se selecciona “Enable camera”, posteriormente “Finish” y se procede a realizar un reinicio del sistema. Una vez concluido el reinicio, se puede comprobar si el sistema funciona empleando este simple comando en el terminal

```
raspistill -v -o test.jpg
```

que tomará la foto y la guardará en el archivo test.jpg.

Otros componentes

Para poder usar la Raspberry Pi correctamente se emplean otros componentes de los cuales no es necesario entrar en detalle ni requieren una configuración complicada. Estos componentes son: pantalla, teclado, ratón, conexión Wifi y conexión Bluetooth.

2.3. Android Phone

Como se ha explicado anteriormente, un teléfono móvil no es suficiente para ser la parte acoplada al sistema de telepresencia. Sin embargo, puede ser altamente útil para entrenar el sistema o incluso para poder tener el sistema de reconocimiento de manera portable.

La elección de Android es debida a la gran extensión de este sistema en multitud de diversos modelos de teléfonos y *tablets*. Por tanto, una vez generada la aplicación se puede instalar en diversos dispositivos que podrán realizar 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.

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

3. Reconocimiento facial

3.1. Elección del sistema

Durante la fase de análisis del proyecto, se encontraron números servicios de reconocimiento facial que podían cubrir las necesidades de este proyecto.

Cada servicio dispone de características diferentes lo cual hizo necesario un análisis profundo antes de determinar cuál era el servicio más adecuado.

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 brevemente los servicios candidatos y en mayor detalle los finalistas.

Amazon Rekognition

<https://aws.amazon.com/rekognition/>

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 numero de caras a reconocer al día.

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

Face++

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

Además del servicio basado en la nube, dispone de un servicio *offline* para Android. En principio esta opción era 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 ha quedado atascada en todo intento.

Kairos

<https://www.kairos.com>

Este servicio ofrece un valor añadido como 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 continuo de caras y menos adecuado para análisis en tiempo real de video.

Google Vision

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

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

Durante la fase de implementación, se descubrió que emplear OpenCV para la detección de caras en el teléfono móvil conlleva un coste de recursos y de implementación alto.

Si bien el servicio de nube no se empleará en el sistema, un subconjunto llamado Mobile Vision dispone de una biblioteca con métodos que permiten una detección de caras más rápida y ha sido finalmente empleado en conjunto con Android.

IBM Watson

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

En principio se planteó emplear este servicio para el proyecto debido a la robustez y fama de Watson.

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 podría ser mejorable y necesita aportar más ejemplos, pero con el material publicado se puede empezar a crear un proyecto viable.

Aunque explícitamente Watson no tiene un sistema de reconocimiento facial, este sistema quedó finalista puesto que existía una manera de crear dicho sistema.

Watson 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; de hecho, en el entrenamiento “Bill Gates vs. Ben Linus” (capítulo 5), los porcentajes de confianza obtenidos eran muy altos (hasta 82%).

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 10. No siempre es posible disponer 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 10 fotos. Emplear la misma foto 10 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 6 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 remediarse, 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 motivó a un cambio en el planteamiento del proyecto y tener que comenzar a codificar de nuevo el sistema.

Microsoft Azure

<https://azure.microsoft.com>

Azure 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.

Las primeras pruebas devolvían un porcentaje de confianza algo inferior a la obtenida con Watson en las pruebas “Bill Gates vs. Ben Linus” (hasta un 73%).

El sistema se dejó de lado hasta que surgieron los problemas con Watson detallados anteriormente, por lo que a mediados de febrero se retomaron las pruebas con el sistema Azure y se ha seguido manteniendo para finalizar el sistema.

OpenCV

<https://opencv.org>

No es un servicio como los anteriores, sino una biblioteca abierta con algoritmos que se pueden emplear para proyectos de visión artificial y aprendizaje automático.

En principio era una opción muy deseable, dado que dispone de gran cantidad de algoritmos muy robustos, revisados y comprobados. Sin embargo, la limitación del hardware, como se expuso en el capítulo anterior, hace que este sistema no sea factible a largo plazo. Cuantas más caras formen parte del sistema, más se tardará en el entrenamiento. El hardware elegido es muy limitado y no funcionará bien pasado cierto número de personas y/o caras.

Esta biblioteca es gratuita, no siendo necesario ningún tipo de suscripción, pero no ofrece un servicio explícito de reconocimiento facial, este debe ser construido.

En principio queda descartado como núcleo de este proyecto, pero tiene un valor añadido que le hace quedar como finalista como servicio de apoyo.

En la

Tabla 2 se comparan los diversos servicios analizados.






							
Servicio online	SI	SI	SI	SI	SI	SI	NO
Servicio offline	NO	SI	NO	NO	NO	NO	SI
Reconocimiento facial exclusivo	SI	SI	SI	NO	NO	SI	NO
SDK	SI	NO	NO	SI	SI	SI	NO
API	SI	SI	SI	SI	SI	SI	NO
Plan estudiante	NO	NO	NO	NO	SI	SI	SI
Plan gratuito	Limitado	Limitado	Limitado	Limitado	Limitado	Limitado	SI
Servicios adicionales (almacén, bases de datos)	SI	SI	SI	SI	SI	SI	NO
Documentación	MEDIA	ESCASA	ESCASA	BUENA	MEDIA	BUENA	BUENA
Valor añadido (detección de emociones, rasgos, sexo)	SI	SI	SI	NO	NO	SI	SI

Tabla 2 Comparativa entre servicios de reconocimiento facial.

3.2. Servicio principal: 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 elige el servicio QeQ Cognitive Services (Figura 6) que permitirá trabajar con la API Face. 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.

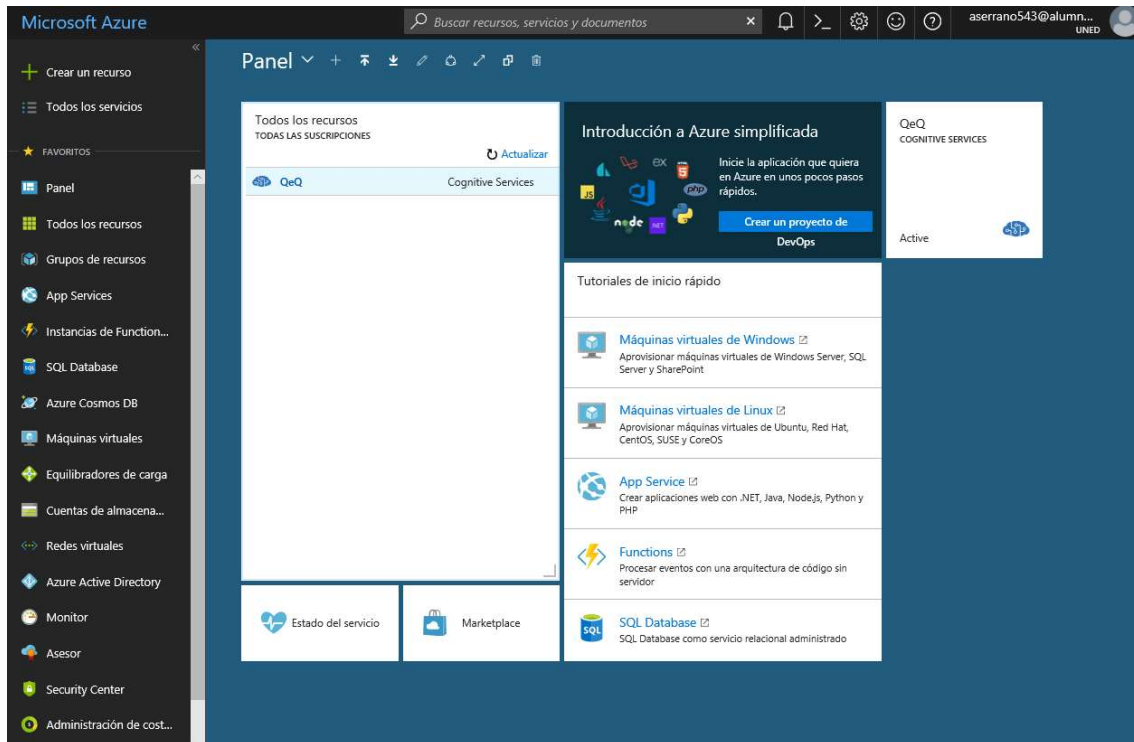


Figura 6 Panel de Azure al darse de alta en el servicio QeQ.

Para el reconocimiento facial Azure dispone del servicio Cognitives 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\[?use rData\]\[&targetFace\]](/persongroups/{personGroupId}/persons/{personId}/persistedFaces[?use rData][&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 serán empleadas, siendo estas Face, PersonGroup y Person, de las que solo se describirán los métodos generales que se emplearán.

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 7).

Cada entidad tiene su identificador único que permite recuperar rápidamente la información pertinente. Los 3 que se emplean son:

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



Figura 7 Ejemplo de organización de entidades (Fuente Microsoft).

Face

Dispone de un método **Detect**, que detecta caras en una imagen, pero por eficiencia no se emplea en este proyecto. Esta tarea se deriva al OpenCV y se describirá más adelante.

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

- un `faceId` (pueden ser varios) con la cara que queremos identificar (esta cara tiene que estar en el sistema). En la sección de implementación se explicará cómo realizar esta operación.
- un `groupPersonId` para identificar el grupo en el que se comparará la cara. Azure permite comparar una cara con un grupo entrenado por llamada, aunque sería fácil implementar un código para verificar todos los grupos a la vez en varias llamadas.

- 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** serán datos en formato JSON indicando los personID de las personas a las cuales podría pertenecer la cara objeto de identificación y la confianza de estas.

Person

Dispone de métodos para agregar, recuperar y borrar personas. Se emplean principalmente 3 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. Opcionalmente se pueden añadir las coordenadas del rectángulo que delimita donde está la cara situada dentro de la imagen. 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 funciona de forma parecida a Person, 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 prudencial usar este método para comprobar si se ha terminado el entrenamiento y es seguro continuar usando el sistema.

El parámetro de entrada será el `groupPersonId` del grupo objetivo de verificación y se obtendrán 4 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 podrán ver en el capítulo 4.

Consideraciones

La clave será omitida del código fuente y de las imágenes, puesto que es una clave privada que se emplea en otros proyectos.

El autor de este proyecto actualmente reside en Suecia, por este motivo el servicio empleado de Azure puede que requiera una configuración distinta en España. El servicio de Azure emplea el servidor:

<https://northeurope.api.cognitive.microsoft.com>

mientras que en España debería emplearse el servidor:

<https://westeurope.api.cognitive.microsoft.com>

En las aplicaciones se contempla la posibilidad de cambiar este parámetro para que sea fácil configurarlo para otras regiones.

3.3. Servicio auxiliar: OpenCV

Como se vio en la parte de análisis y en el capítulo dedicado a hardware, OpenCV requiere mayor uso de recursos cuanto más complejo sea el sistema y, debido al hardware elegido, los recursos son muy limitados.

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

Este proyecto pretende ser móvil, para ello las comunicaciones a Internet serán 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. Las consecuencias de un gran tamaño son:

- El envío de datos mediante medios inalámbricos generalmente tiene menor velocidad que si se empleasen 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 será 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 ello se empleará un conjunto de algoritmos de OpenCV que, tras la toma de la foto, recortan las caras relevantes para su envío individual al servicio y posterior identificación de cada una de ellas.

Si bien se ha comentado la limitación del hardware, la implementación del recorte de caras es ligera, rápida y no demanda demasiados recursos.

OpenCV dispone de un módulo llamado **objdetect** (Object Detection) que contiene la clase CascadeClassifier.

CascadeClassifier está basado en la cascada de clasificadores. Este método se emplea ampliamente para la detección de caras debido al bajo consumo de recursos.

Una breve introducción general teórica es necesaria para entender por qué se elige este método.

Para la detección de caras se emplea en primer lugar una generación de candidatos basado en el uso de una ventana deslizante de diversos tamaños que se aplica a diferentes partes de la imagen; cada ventana aplica un filtro que determina si es buen candidato o no (Figura 8).

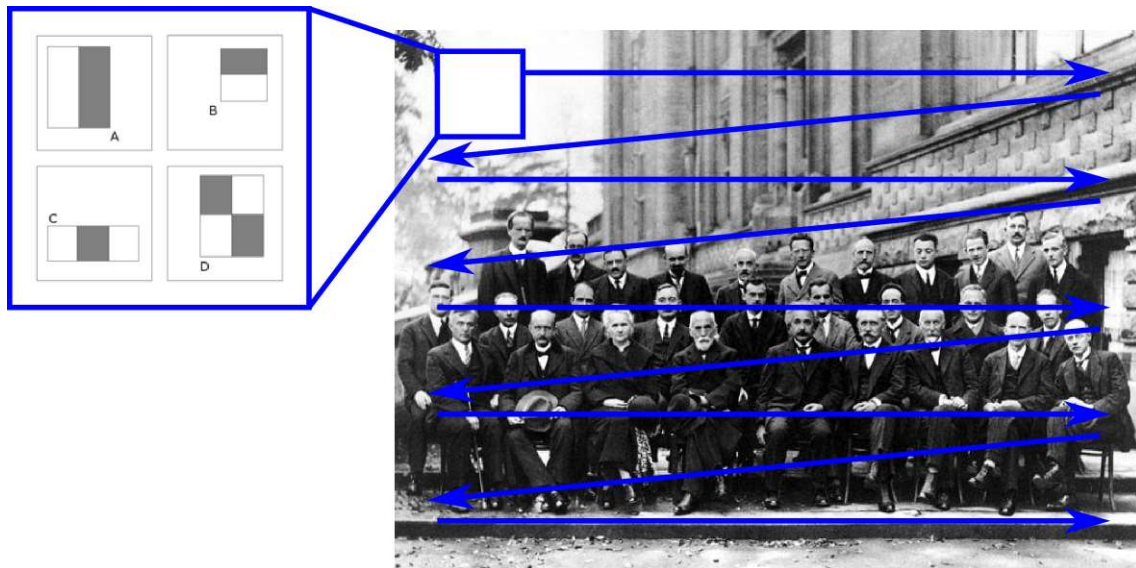


Figura 8 Ejemplo de ventana deslizante (Fuente Wikipedia)

Tras la generación de candidatos se realiza la clasificación, cada ventana se somete al clasificador y este devuelve correcto o incorrecto.

El CascadeClassifier que va a emplearse funciona en líneas generales de esta manera:

Para el filtrado se emplean filtros de Haar en diferentes escalas y en múltiples posiciones (Figura 9):



Figura 9 Ejemplo de filtros de Haar (Fuente Coursera).

Tras la aplicación de los filtros de Haar en las ventanas deslizantes se generan los candidatos y se almacenan.

Posteriormente cada candidato se somete a una cascada de clasificadores. El objetivo del sistema es que el candidato pase cada clasificador individual y si uno de ellos no lo clasifica, entonces el candidato se descarta.

automáticamente. De esta forma se incrementa la eficacia, puesto que, al estar en serie, si un clasificador rechaza al candidato no se requieren más comprobaciones (Figura 10).

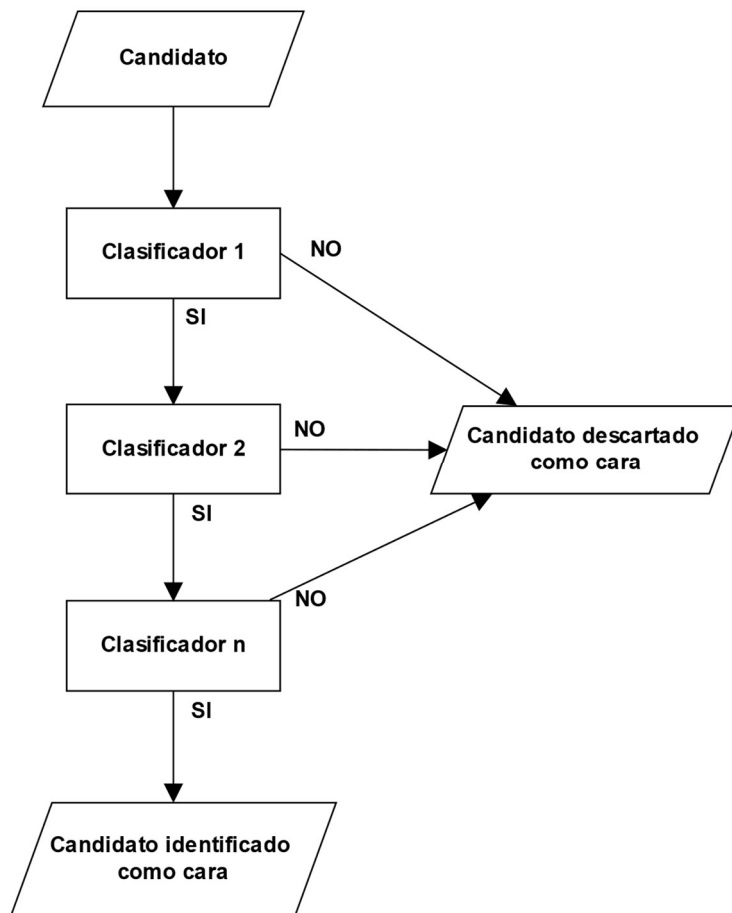


Figura 10 Cascada de clasificadores.

Junto a CascadeClassifier se pueden emplear también los LBP (Local Binary Patterns) como alternativa a los filtros de Haar. El motivo principal de usar filtros de Haar frente a los LBP es la precisión: LBP es más rápido, pero menos preciso.

Para este proyecto se requiere rapidez, sin embargo, la detección de caras para esta parte del sistema no genera ningún tipo de almacenamiento ni creación de clasificadores ni entrenamiento, por lo que la ganancia de velocidad es menos relevante que la precisión.

Se emplean los filtros de Haar en combinación con la clasificación en cascada.

4. Software

4.1. Introducción

El sistema se divide en dos partes:

FaceRecon: Reconocimiento continuo de caras. Instalado en la Raspberry pi con un programa que continuamente reconoce caras y devuelve una imagen con las identificaciones de las caras.

FacePal: Entrenador de bolsillo. Se instala en un teléfono móvil y permite el entrenamiento de una manera más cómoda. Incluye el reconocimiento de caras.

4.2. Elección del lenguaje

Aunque existe una gran variedad de lenguajes y sería interesante explorar varios candidatos para elegir el que mejor se adapta al proyecto, las limitaciones establecidas por el hardware y servicios elegidos acotan bastante la lista de lenguajes posibles a emplear.

FaceRecon

Al estar instalado dentro de la Raspberry Pi, existen muchas opciones de elección del lenguaje. La primera pregunta para poder realizar la criba es ¿lenguaje compilado o interpretado?

El lenguaje compilado, como C++, tiene la ventaja de mayor rapidez. En Raspian pueden instalarse fácilmente varios compiladores de C++ (g++ o clang), pero a medida que crece el proyecto, el enlazado con otras bibliotecas de software se puede volver complicado y confuso. Además, es responsabilidad del programador evitar las violaciones de acceso (comúnmente denominadas segfault) y en ocasiones se tarda mucho en averiguar dónde está el problema en el código. Dado el margen de tiempo del que se dispone, se descarta C++ como lenguaje elegido.

Existen otros lenguajes como Rust que son compilados y quitan al programador la responsabilidad de muchas comprobaciones. Sin embargo, en el momento

de la redacción de este proyecto Rust es un lenguaje relativamente nuevo y aun no tan extendido como otras opciones.

Los lenguajes interpretados requieren el uso de un intérprete cada vez que se ejecuta el código, esto hace que el rendimiento global sea inferior al de un código compilado. Sin embargo, permite la creación de aplicaciones más ágiles puesto que al ser ejecutadas el intérprete se encarga de todo, no es necesario compilar y luego ejecutar.

Los dos lenguajes que serían adecuados para programar esta aplicación serían Java y Python. Si bien ambos son muy extendidos y están bien documentados, se decide emplear Python (versión 3) por su facilidad de uso y facilidad de lectura del código. Además, Python permite que una función devuelva más de un valor, incluso de distintos tipos, o permite colecciones de objetos de distinto tipo sin tener que crear clases exclusivas para este fin.

FacePal

Al estar destinado a que sea instalado en un sistema Android, la opción más lógica es el lenguaje Java. 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.

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.

Consideraciones previas

Se ha optado por usar inglés como idioma para determinar el nombre de variables, funciones y otras piezas de software. Sin embargo, en esquemas y comentarios del código, se empleará español.

4.3. FaceRecon

La descripción genérica de la aplicación es la expuesta en la Figura 11.

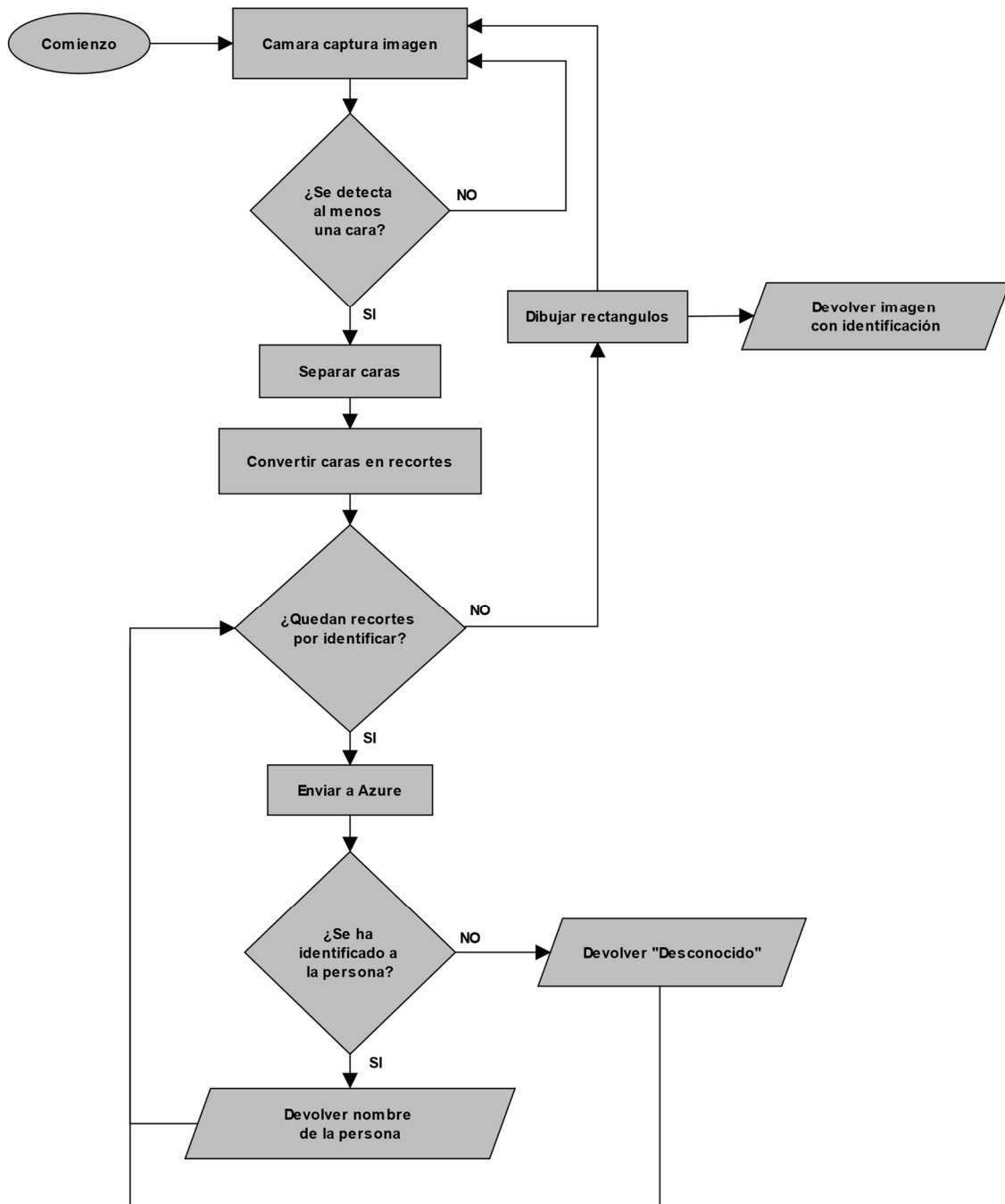


Figura 11 Descripción de FaceRecon.

Instalación

La instalación es sencilla, se realiza copiando la carpeta faceRecon dentro de la carpeta src de este proyecto en el directorio destino que se desee dentro de la Raspberry Pi.

En la Figura 12, FaceRecon está localizado en

/home/pi/Desktop/PFG/src/faceRecon

de esta forma abriendo un terminal y escribiendo

```
cd /home/pi/Desktop/PFG/src/faceRecon
```

se accede al directorio. Posteriormente, se ejecuta con el comando

```
python3 facerecon.py
```

El programa se ejecuta en un bucle infinito hasta que el usuario pulse la combinación Ctrl+C, apague el sistema u ocurra un error.

Las imágenes creadas por FaceRecon son almacenadas en el directorio img, cada una con el nombre `image_{marca_temporal}.jpg`.

Figura 12 Ejecución de FaceRecon.

Para facilitar la comprensibilidad del sistema y el mantenimiento, se divide el código en módulos/clases agrupando funciones similares. En la Figura 13 se puede observar la dependencia de clases/módulos.

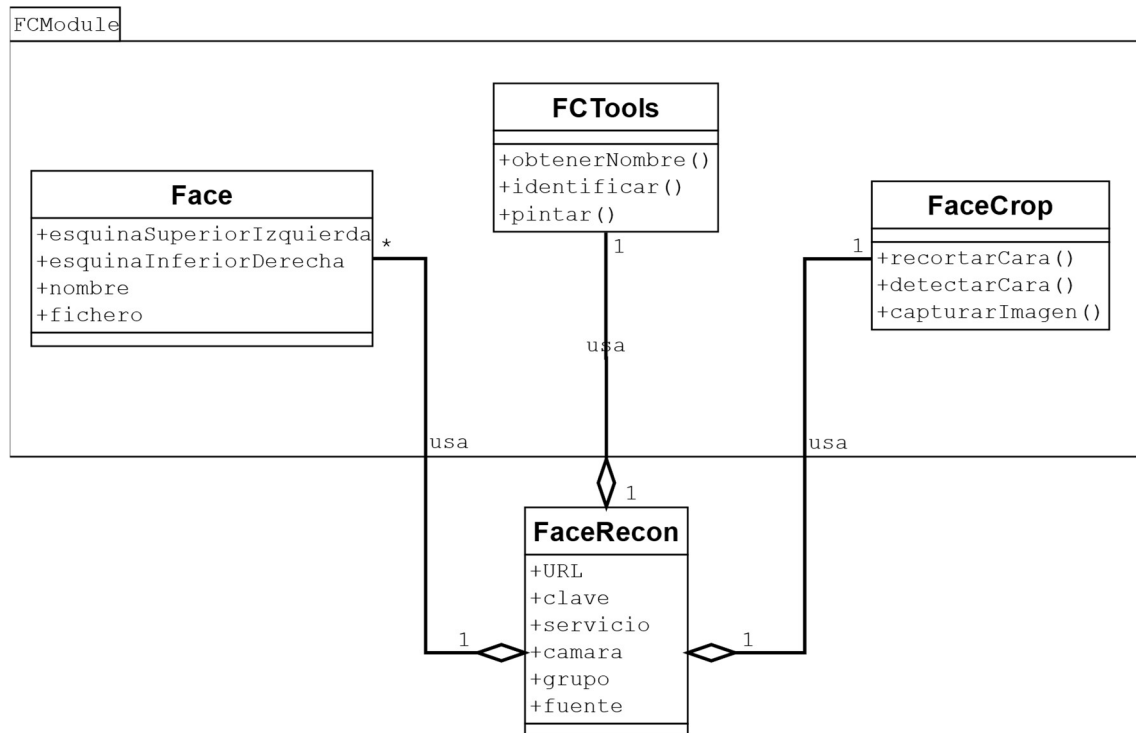


Figura 13 Diagrama de clases de FaceRecon.

FaceRecon

Es el programa principal, sirve como ejemplo de uso del módulo FCmodule, se encarga de cargar la configuración, ejecutar el bucle principal y contener la función de identificación.

Atributos:

- sKey: contiene la clave de suscripción de Azure.
- server: indica el valor que se añade a la URL base para la conexión con Azure. El valor por defecto es northeurope.
- faceDetector: el archivo XML que contiene la información de filtros a emplear. Se encuentra en la carpeta 'utils'. Estos filtros se han obtenido de OpenCV y se almacenan en este directorio por comodidad pero en futuras versiones este directorio almacenaré los filtros personalizados.
- groupName: el nombre del grupo en Azure que contiene las personas cuyos datos ya han sido subidos. Por defecto será 'conocidos'.
- waitTime: tiempo de espera entre identificaciones.

Función principal:

Consiste en 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)

    Imagen = capturar imagen

    Caras.insertar (Imagen) //detecta, recortar
    por cara en Caras

        identificar cara

        pintar cara en Imagen

Mostrar imagen
```

Face

Clase que 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 14.Figura 14

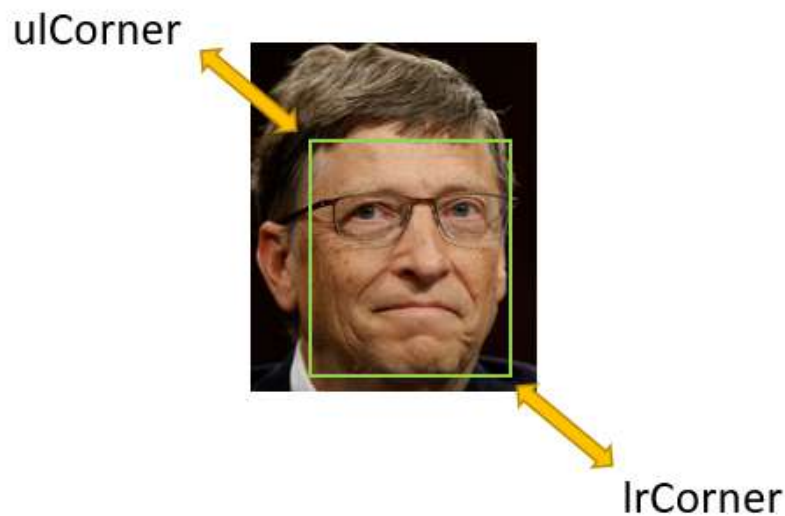


Figura 14 Ejemplo de esquinas del rectángulo.

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 de la persona tras su identificación.

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

faceCrop

Este módulo contiene las funciones de gestión de las imágenes.

Función `captureImage(camera)`:

Captura una imagen desde la cámara de 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.

Función faceDetect(faceDetector, image, cv2):

Detecta caras en una imagen.

Parámetros:

- faceDetector: XML 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.

Función 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.

FCTools

Este módulo contiene varias funciones para el entrenamiento inicial del sistema. Solo se describirán en detalle las funciones más relevantes, del resto solo se hará una descripción breve. Las otras funciones no se emplean durante

el funcionamiento normal de FaceRecon pero son útiles para futuros usos y necesarias para los entrenamientos.

Función `identifyFace(service, groupName, face)`:

Parámetros:

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

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

Parámetros:

- `cv`: objeto de OpenCV que contiene 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`.

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.

Función `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 ID del candidato, pero no su nombre. Esta función devuelve el nombre real del 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 funciones que se han usado para los entrenamientos iniciales y que pueden formar parte de otros proyectos.

Función testAzure (service):

Test de funcionamiento.

Función getPersonID(service, personGroup, personName):

Obtiene el ID de una persona mediante el nombre real.

Función createPersonGroup(service, personGroup):

Crea un grupo de personas.

Función createPerson(service, personGroup, personName):

Crea una persona.

Función addFace(service, personGroup, personName, image):

Añade una cara a una persona ya dada de alta.

Función deletePerson(service, personGroup, name)

Elimina una persona y todas las caras asociadas

Función deletePersonGroup(service, personGroup):

Elimina un grupo de personas y toda la información asociada.

Función listPersons(service, personGroup):

Devuelve información sobre las personas de un grupo.

Función detectPerson(service, personGroup, file):

Identifica una persona desde una imagen de archivo.

4.4. FacePal

Una descripción genérica de la aplicación sería la siguiente (Figura 15):

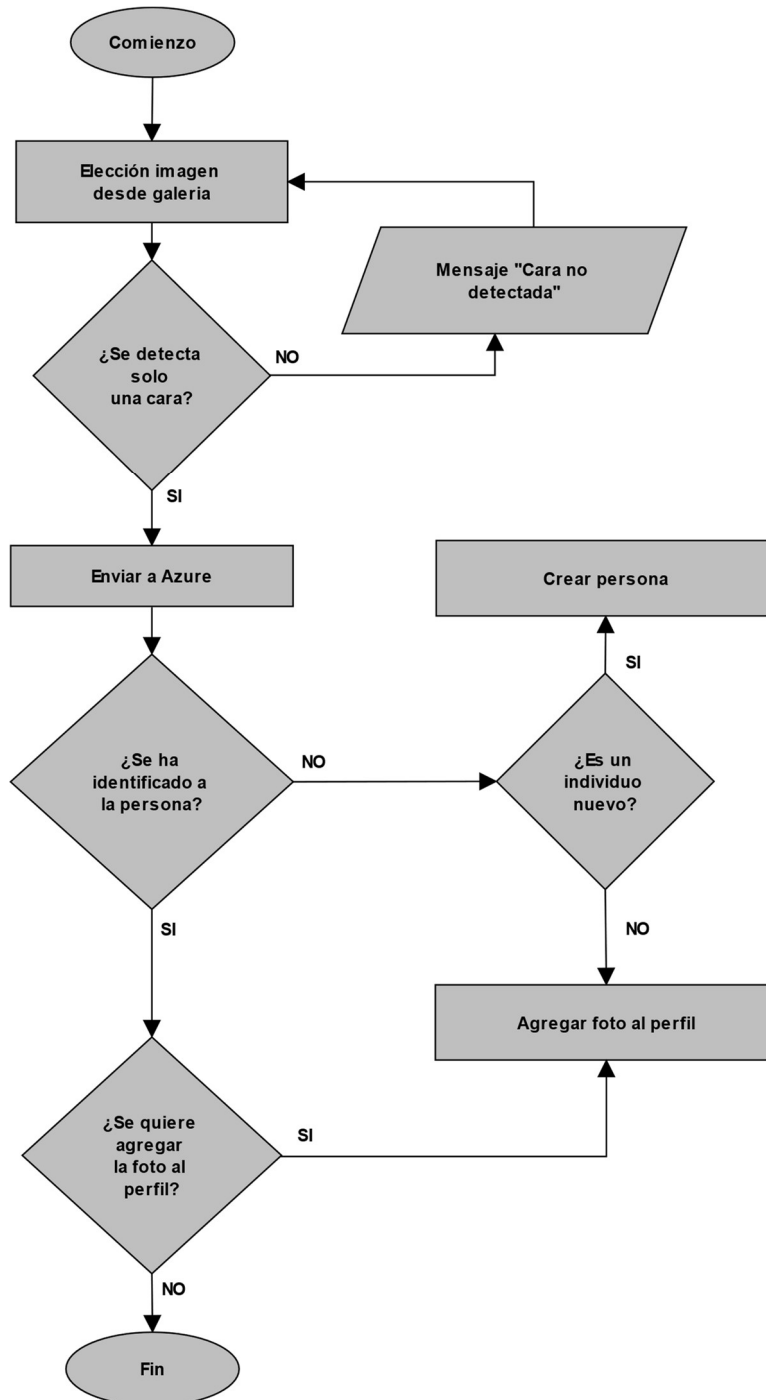


Figura 15 Esquema general de FacePal.

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 teléfonos. 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 teléfonos Android de la misma forma. Por ejemplo, en el teléfono de pruebas principal (Samsung Galaxy S3) la cámara siempre toma las fotos con 90° 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 teléfono. 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 teléfono móvil esperando a ser identificado.
- pueden emplearse imágenes descargadas de internet o enviadas al teléfono del usuario mediante aplicaciones de mensajería, redes sociales, etc.

Sobre el SDK

Microsoft tiene disponible un SDK para aplicaciones Android y facilitar el uso de Face API en aplicaciones propias. Al contrario que su equivalente en Python, el SDK está pobremente 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 ha tomado 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` para realizar las solicitudes directamente a Azure. Para interpretar resultados se ha empleado la biblioteca JSON que incorpora Android Studio.

Instalación

En el momento de la redacción de este proyecto, FacePal no se encuentra en el Play Store para su instalación rápida; por tanto, existen dos formas de instalar: desde archivo APK o desde el propio Android Studio.

Instalación desde APK

El archivo **facepal.apk** contiene la aplicación completa, se encuentra dentro del directorio build de este proyecto.

Para su instalación, se debe copiar **facepal.apk** al almacenamiento del teléfono (puede ser en memoria interna o la tarjeta SD). Una vez copiado, se localiza el archivo usando el gestor de ficheros correspondiente y se pulsa una vez para instalar. Es importante que el sistema esté configurado para permitir la instalación desde fuentes desconocidas.

Una vez que acaba la instalación, el icono de FacePal aparece en pantalla de selección de aplicaciones.

Instalación desde Android Studio

Tras abrir el proyecto y conectar el teléfono a un puerto USB, se selecciona el menú **Run** y posteriormente se selecciona la opción **Run 'app'**, como se ve en la Figura 16, y se selecciona el dispositivo en el que se desea instalar el programa. Tras unos instantes aparece en la pantalla del dispositivo la pantalla principal de FacePal.

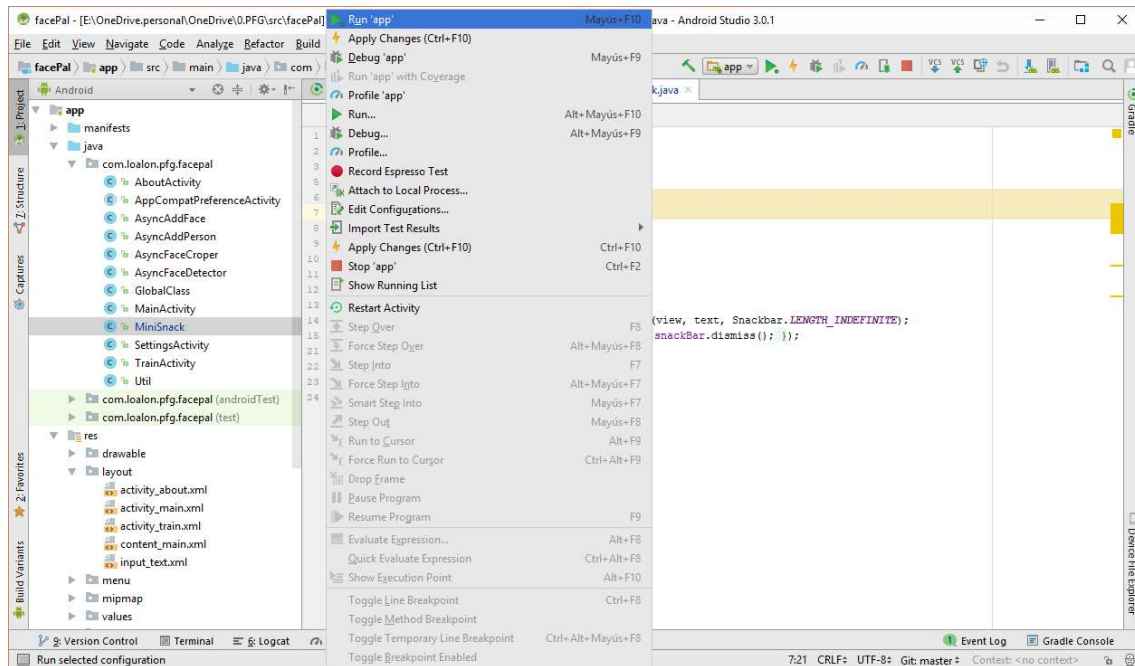


Figura 16 Instalación de FacePal.

Es importante recordar que tras la primera instalación se debe acceder al menú de configuración de FacePal y añadir los siguientes parámetros:

- grupo de persona de entrenamiento
- clave de suscripción
- servidor

Funcionamiento

La parte de código relevante se compone de varias clases:

- Actividades: declaran las funciones que se emplean en cada actividad.

FacePal tiene varias actividades:

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

- Clase con funciones Util:

La clase 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 ruido irrelevante. En caso de no encontrar un solo rostro, se retorna un objeto nulo.

- `String getName(String personID)`

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 una 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.

- 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 (override) tres métodos útiles:

- 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 `onPostExecute` 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 determinadas características empleado en la devolución de mensajes por parte de la aplicación.

Para explicar el desarrollo de esta aplicación se estudiarán sus distintas pantallas y se entrará en los detalles de cada elemento.

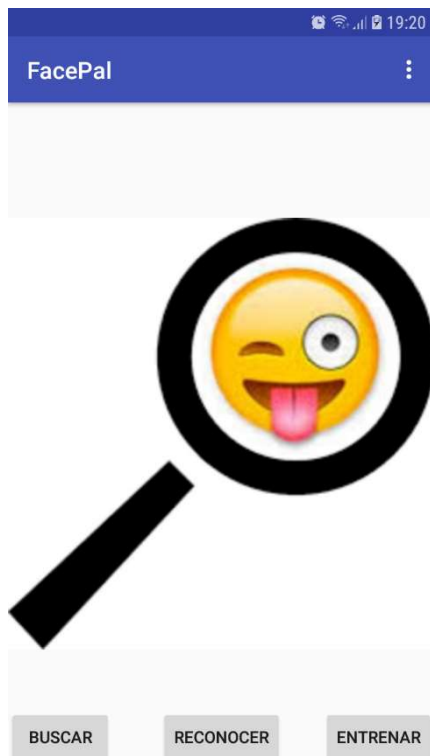


Figura 17 Pantalla principal de FacePal.

La pantalla principal (Figura 17) dispone de 3 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 4 funciones, 2 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.

El menú configuración (Figura 18) dispone de dos opciones: Configuración y Acerca de...

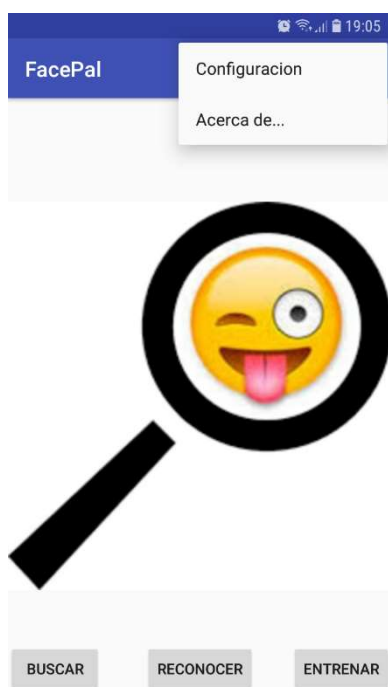


Figura 18 Menu configuración.

La pantalla de configuración solo dispone del submenú General, permitiendo futuras ampliaciones. Dentro de este submenú se dispone de 3 opciones de configuración (Figura 19).

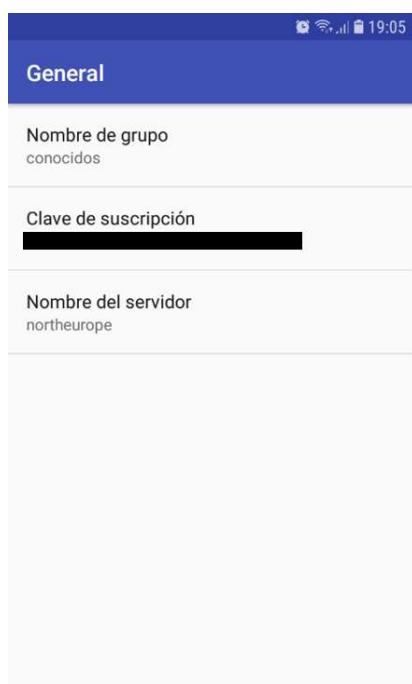


Figura 19 Pantalla Configuración.

- 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.

Volviendo a la pantalla principal se analizan los botones. Se muestran varias pantallas de ejemplo de funcionamiento a lo largo del capítulo 5.

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.

Si la imagen contiene más de un rostro, ninguno o la rotación es incorrecta, devolverá un mensaje de error y sacará 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`. Envía mensajes al usuario de la tarea que se ejecuta y se encarga de la tarea de detectar 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 su diseño esta descrito en activiy_train.xml.

La pantalla de entrenamiento consiste en 4 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 3 tareas:

- Comprueba que el nombre introducido en el área de texto no exista 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.

5. Entrenamiento

5.1. Comprobación con imágenes existentes

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

- al aumentar el número de imágenes, aumenta la confianza obtenida
- el sistema es capaz de distinguir personas con rasgos muy similares

Para ello se va a emplear el módulo de entrenamiento FCTools que forma parte de FaceRecon creada para el entrenamiento en la Raspberry Pi, aunque por comodidad se ha utilizado en Windows 10. Además, se diseñan cuatro pruebas 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

Todas estas pruebas se realizan con imágenes existentes.

Prueba Bill Gates vs. Bill Gates

En esta primera prueba se emplean 10 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 20).



Figura 20 Retratos Bill Gates (Fuente Google).

Para confirmar la primera hipótesis se han realizado una serie de pruebas que se explican con el siguiente pseudocódigo:

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



Figura 21 Imagen de Bill Gates empleada como caraTest (Fuente Microsoft).

```

Windows PowerShell
PS E:\PFG\pyazure> python test1.py
{'persistedFaceId': '6bac7072-ac1d-4d45-b8cb-7515f019910b'}
Datos caraTest
[{'faceId': 'dcbde737-5994-488b-916a-ae5ecb78d9ef', '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 22 Ejemplo de ejecución durante prueba Bill vs. Bill.

Como se puede comprobar en el gráfico de confianza de la Figura 23, cuantas más imágenes se adjunen a la misma persona, mayor será la confianza obtenida, por tanto, la primera hipótesis queda confirmada.

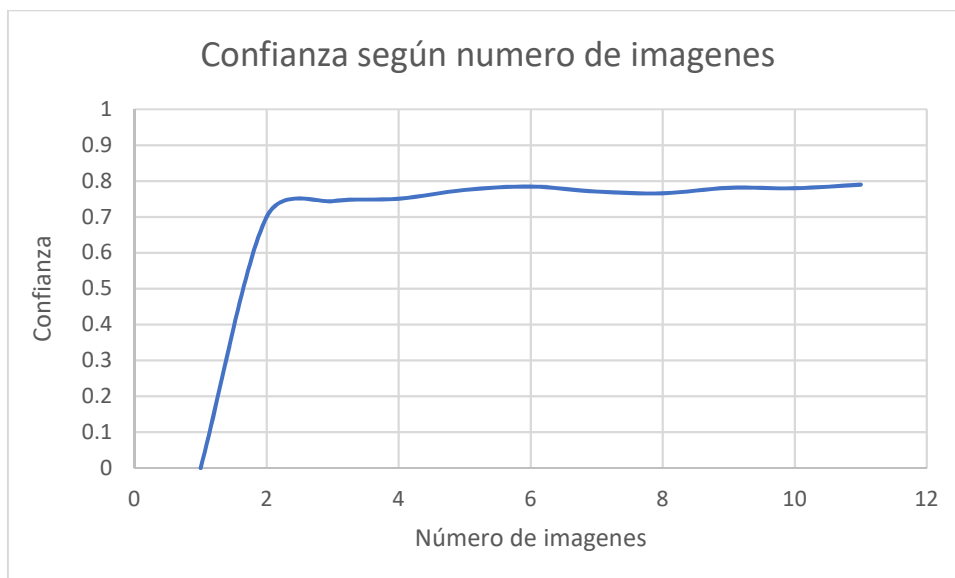


Figura 23 Grafica Bill vs. Bill.

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, bill6 y bill8, y pueden deberse a los siguientes motivos:

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

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

Prueba Ben Linus vs. Ben Linus

Esta prueba está diseñada para poder verificar la primera hipótesis, es decir, es complementaria a la prueba anterior.

Para esta prueba, creamos a una nueva persona en el sistema 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 creará la persona de Ben Linus en el sistema y se le añadirán las 10 caras de la Figura 24.



Figura 24 Caras elegidas de Ben Linus (Fuente Google).

La diferencia en este caso es que las imágenes no son tan homogéneas en tamaño ni forma como en la prueba anterior, de esta manera se puede observar si el sistema es capaz de discernir el ruido que supone toda información fuera del área de la cara.

Adicionalmente se comprueba el caraTest empleando el rostro de la Figura 25 y cada cara de Ben Linus individualmente antes incluso de crear la persona en el sistema, para comprobar si alguna imagen tiene un parecido razonable con Bill Gates.



Figura 25 Ben Linus (Michael Emerson) (Fuente Wikipedia).

Tras la ejecución se obtienen los resultados observados en la Figura 26 y Figura 27.

```
PS E:\PFG\pyazure> python test2.py
Datos caraTest
[{'faceId': '8b81a686-4adb-4d28-90d5-2e9f60bd8d72', 'faceRectangle': {'top': 119, 'left': 69, 'width': 197, 'height': 197}}]
Datos mejor candidato
[{'faceId': '8b81a686-4adb-4d28-90d5-2e9f60bd8d72', 'candidates': []}]
No existe un candidato que concuerde
PS E:\PFG\pyazure>
```

Figura 26 Ejecución del caraTest de Ben Linus antes del entrenamiento.

```

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-2f77-4c1a-a85f-633e53c4dd0c', 'faceRectangle': {'top': 457, 'left': 343, 'width': 634, 'height': 634}}]
Datos mejor candidato
[{'faceId': 'ff6617bf-2f77-4c1a-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>

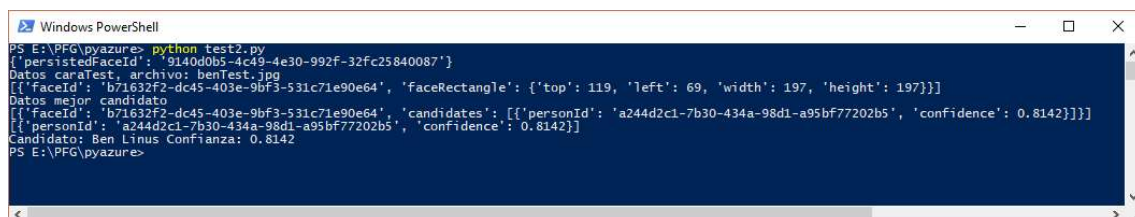
```

Figura 27 Ejecución de los 10 rostros de Ben Linus.

En ningún caso el sistema el rostro como perteneciente a Bill Gates. Se debe tener en cuenta que Azure no devuelve a ningún candidato cuyo umbral de confianza esté por debajo de 0.5.

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.

De forma similar a la prueba anterior, se realiza la subida de las fotos de entrenamiento de Ben Linus y se ejecuta la prueba del caraTest (Figura 28).



```

Windows PowerShell
PS E:\PFG\pyazure> python test2.py
{'persistedFaceId': '9140d0b5-4c49-4e30-992f-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-98d1-a95bf77202b5', 'confidence': 0.8142}]}]
[{'personId': 'a244d2c1-7b30-434a-98d1-a95bf77202b5', 'confidence': 0.8142}]
Candidato: Ben Linus Confianza: 0.8142
PS E:\PFG\pyazure>

```

Figura 28 Ejecución del caraTest de Ben Linus tras el entrenamiento.

Como puede apreciarse en la Figura 29, el resultado es el esperado: a medida que aumenta el número de imágenes, aumenta la confianza que devuelve el sistema al usar el caraTest.

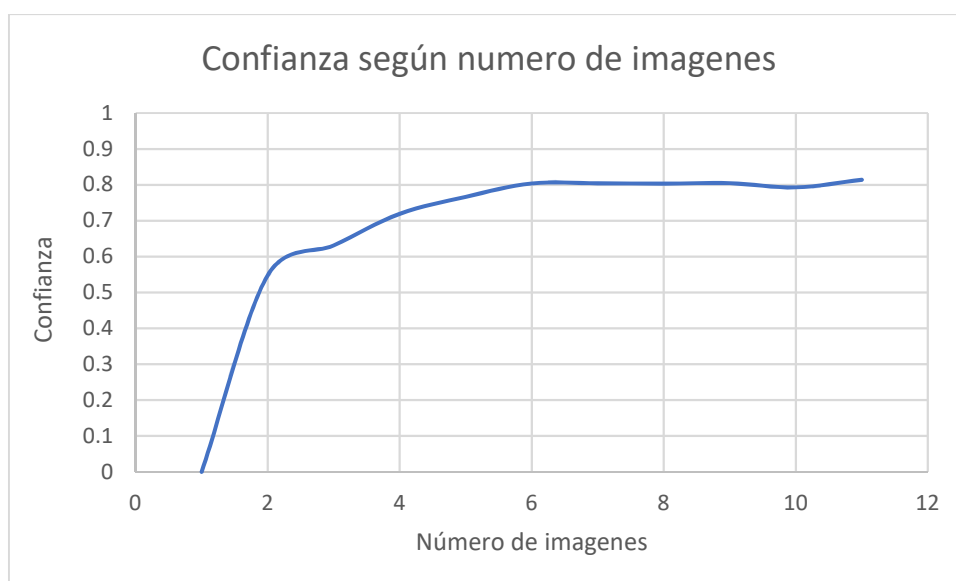


Figura 29 Gráfica Ben vs. Ben.

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 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 (la primera que se sube al sistema) es oscura y Ben no tiene gafas, mientras que el caraTest es luminoso 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 el caraTest.

Prueba Bill Gates vs. Ben Linus

Para esta prueba se somete al grupo que contiene a ambos personajes a todas las caras empleadas para ambos entrenamientos. De esta manera se puede saber si alguna de las caras de Bill es reconocida como Ben y viceversa. Los datos de confianza se reflejan en la

Tabla 3.

Archivos	Bill Gates	Ben Linus
bill0	0.83524	0
bill1	0.80866	0
bill2	0.85215	0
bill3	0.82128	0
bill4	0.87035	0
bill5	0.88662	0
bill6	0.82094	0
bill7	0.81845	0
bill8	0.80564	0
bill9	0.87751	0
ben0	0	0.80195
ben1	0	0.72930
ben2	0	0.76416
ben3	0	0.81979
ben4	0	0.83860
ben5	0	0.85980
ben6	0	0.60980
ben7	0	0.86963
ben8	0	0.79360
ben9	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 es clasificada como Ben Linus, ni como segunda opción con peor confianza.

Del mismo modo ocurre que ninguna imagen de Ben Linus es identificada como Bill Gates.

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

Prueba Bill Gates vs. Bill Gates Impersonator

Por último, se decide 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) y 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 30. Como en los casos anteriores, se somete la foto del imitador al método de verificación.



Figura 30 Imitador de Bill Gates (Fuente lookalike.com).

```
Windows PowerShell
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 31 Ejecución de la prueba.

Archivos	Bill Gates	Ben Linus
billImpersonator	0	0

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

El clasificador no encuentra a ningún candidato que tenga similitudes con el imitador, por tanto, la segunda hipótesis queda reforzada.

5.2. Entrenamiento con imágenes existentes y reconocimiento en continuo

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

Se han empleado 9 fotos del autor con distintas edades, luminosidad, barba y corte de pelo (Figura 32).

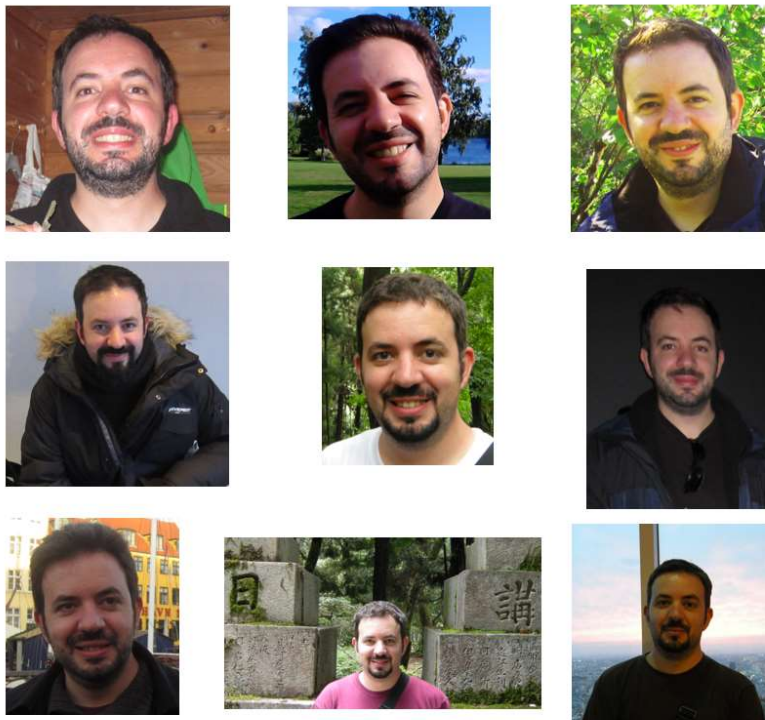


Figura 32 Fotos de entrenamiento de "Admin".

Al contrario que en los casos anteriores, no se empleó un caraTest 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 FaceRecon funcionaba correctamente.

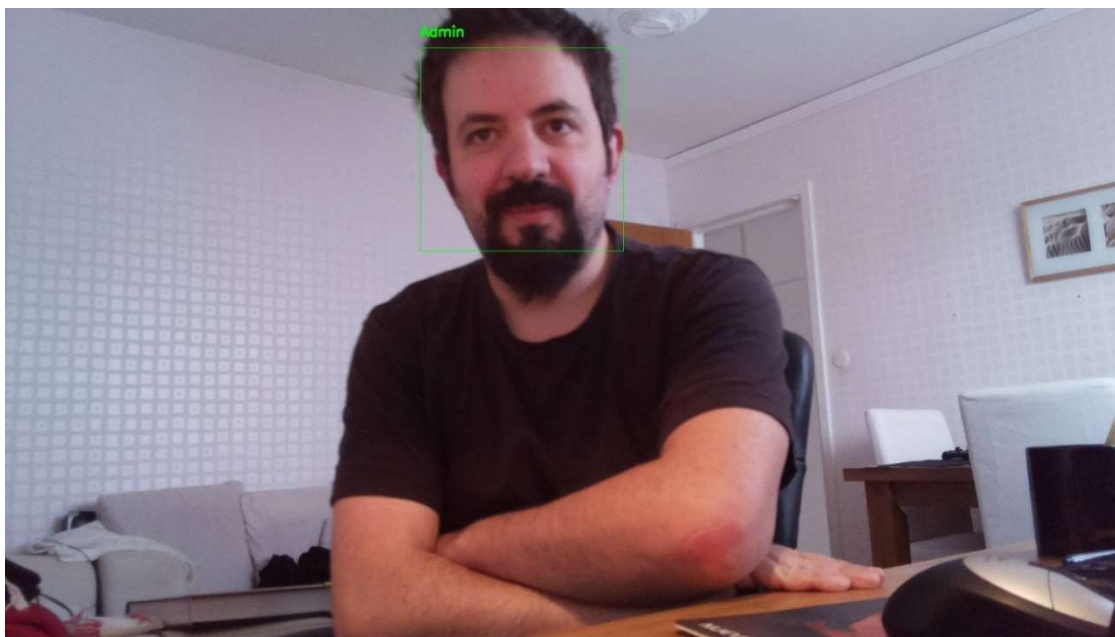


Figura 33 Primer reconocimiento de Admin.

La Figura 33 es la primera evidencia de que el sistema, que solo se había entrenado con imágenes ya existentes, es capaz de reconocer a un individuo en una imagen tomada en continuo.

La Figura 34 se realiza desde un ángulo distinto y con iluminación artificial frontal y natural trasera. El sistema reconoce a Admin sin problemas.

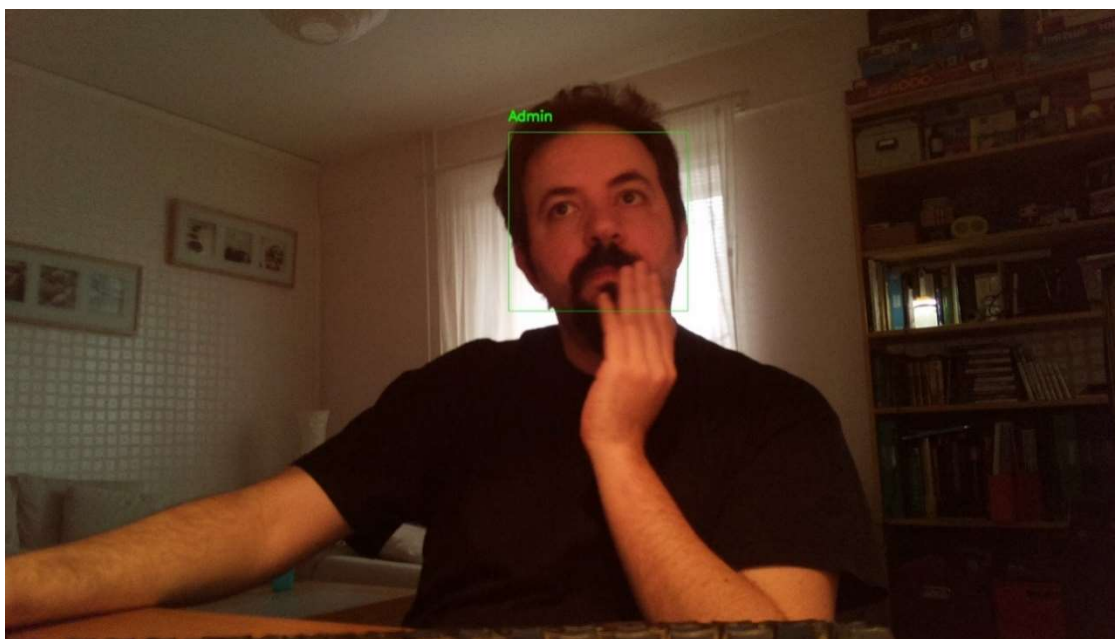


Figura 34 Imagen de Admin con iluminación alternativa.

5.3. Entrenamiento y reconocimiento en FacePal

Para esta prueba se han tomado retratos de 3 individuos con la cámara del teléfono móvil y se han sometido a la función de reconocimiento de FacePal. Tras confirmar que no los reconoce, se han agregado a la lista de personas conocidas.

Sujeto 1: Akbar Espaillat

En la Figura 35 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 ha sido agregada al sistema.

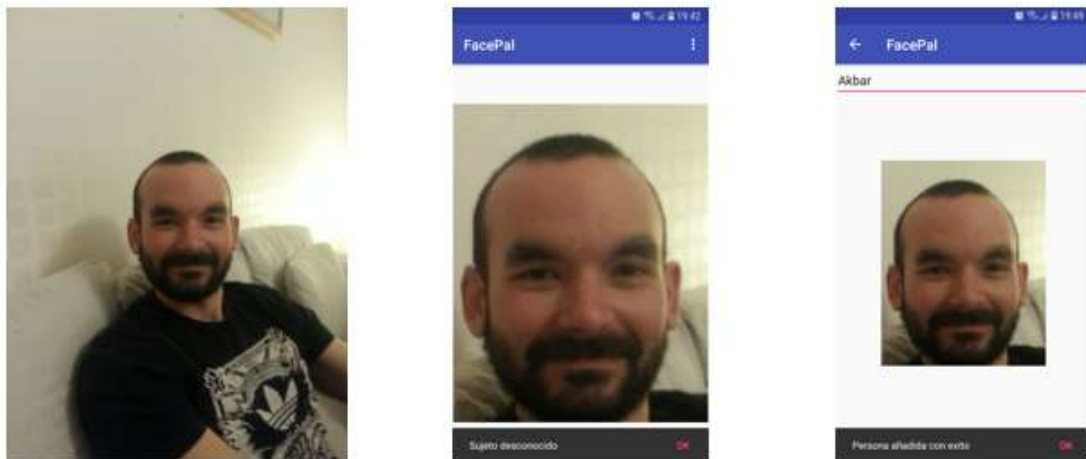


Figura 35 Akbar Espaillat.

Sujeto 2: Bastian Schiffthaler

La Figura 36 muestra cómo se agrega a Bastian Schiffthaler al sistema, en esta foto el sujeto tiene gafas puestas. En la Figura 37 se comprueba que el sistema reconoce a Bastian sin gafas y permite añadir una nueva cara a esa persona.

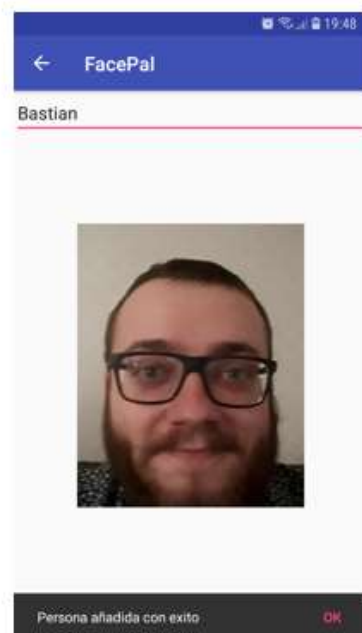


Figura 36 Creación de persona Bastian Schiffthaler.

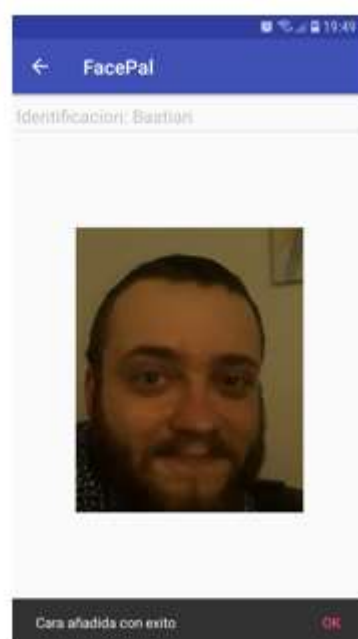


Figura 37 Cara agregada a Bastian Schiffthaler.

Sujeto 3: Javier Alvarez

En el caso de Javier Alvarez se han tomado 2 retratos distintos: uno con su aspecto habitual y otro con gafas y gorro. El propósito es saber si mediante FacePal se identifica al mismo individuo a pesar de tener un aspecto distinto.

La Figura 38 muestra cómo se agrega la primera imagen al sistema, cómo el sistema reconoce 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 a que se agrega la segunda cara a los datos de la persona.

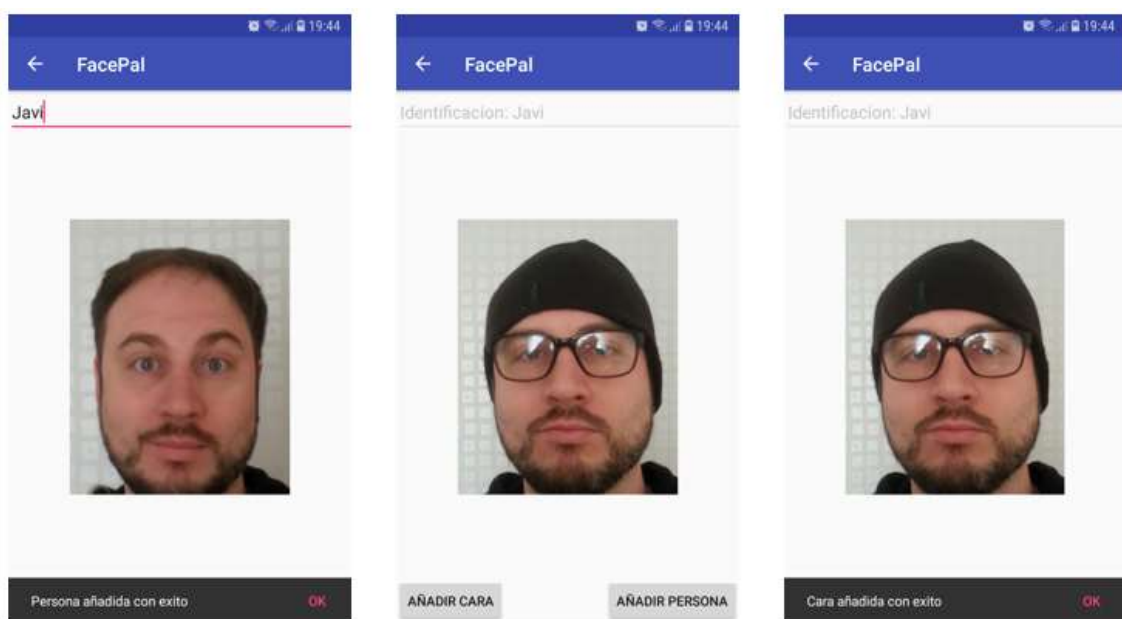


Figura 38 Javier Alvarez.

Tras estas pruebas se puede comprobar el buen funcionamiento del sistema, 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.

5.4. Entrenamiento y reconocimiento FacePal+FaceRecon

En este punto, el sistema contiene varias personas almacenadas y en algunos casos varias fotos por persona. La última prueba consiste en dejar funcionar a FaceRecon durante 1 día y comprobar si reconoce caras en una situación real.

Se tomaron aproximadamente 300 fotos durante el funcionamiento continuo. La mayoría eran del autor frente al ordenador, siendo poco relevantes. El resto contienen a más de una persona. Se muestran 3 de ellas para demostrar el correcto funcionamiento.

La Figura 39 muestra a Akbar y Admin reconocidos durante una demostración de la aplicación.

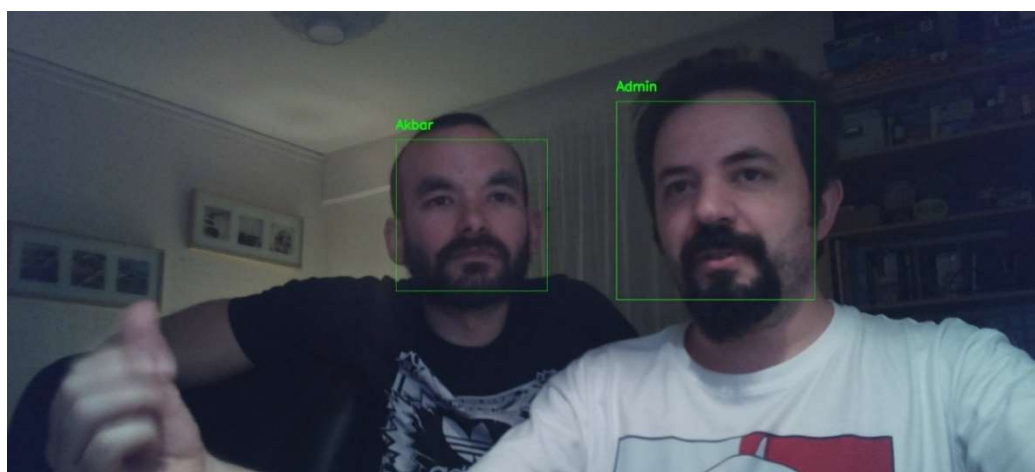


Figura 39 Reconocimiento de Akbar y Admin.

La Figura 40 muestra a Javi y Admin durante el día.



Figura 40 Reconocimiento de Javi y Admin.

La Figura 41 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 para comprobar si la detección facial funcionaba.

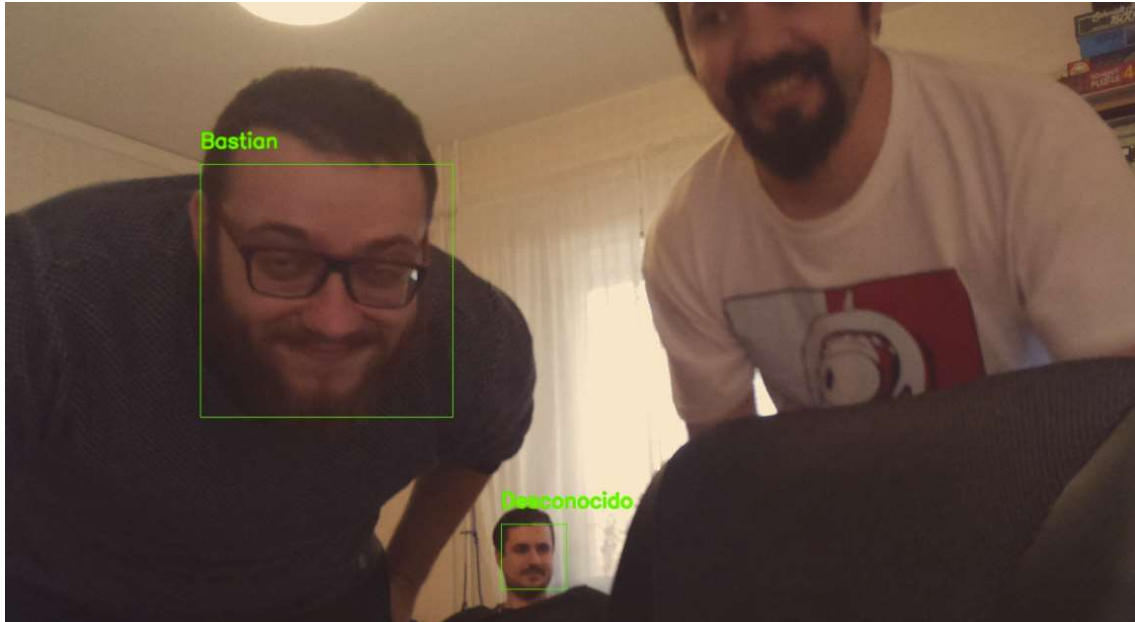


Figura 41 Reconocimiento de Bastian.

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 FaceRecon es un buen recurso base para crear un proyecto mayor.

6. Conclusiones

6.1. Objetivos alcanzados

Tras el éxito de las pruebas se puede concluir que el sistema cumple en alto grado con las metas planteadas al inicio de este proyecto. Para ello se volverán a citar las metas y analizar su grado de cumplimiento.

Rapidez

La velocidad de FaceRecon es adecuada para el uso previsto. Con dos o más rostros en pantalla, el tiempo total desde que captura la imagen, realiza los recortes, los envía, recibe respuestas, las procesa, señala 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, en 3 segundos de media desde que se pulsa el botón Reconocer se obtiene respuesta con la identificación del sujeto. Al añadir cara, el tiempo son 3 segundos para reconocer y otros 2 segundos para 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 aceptables y no suponen que el usuario tenga que estar pendiente demasiado tiempo del teléfono móvil.

FacePal presenta un cuello de botella en la detección de caras en una imagen. El empleo de una ventana deslizante como se explicó al comienzo de este proyecto implica deslizar una ventana de tamaño decreciente por toda la imagen. En cada deslizamiento se emplean los filtros para buscar el rostro. 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, la ventana deslizante tarda más en adquirir el rostro. Durante las pruebas 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 están en 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 puede ser hasta 4 veces más. No se entiende muy bien por qué ocurre este evento. Hasta que una alternativa sea aceptable simplemente se deben evitar las imágenes lejanas.

Mobile Vision no es perfecto, pero es adecuado. Lo ideal sería emplear OpenCV, pero su implementación en Android es más compleja que en Python y en un teléfono no se disponen de los mismos recursos hardware.

Eficacia

Tanto FacePal como FaceRecon, realizan un recorte de las caras en una imagen, reduciendo en gran medida el ruido adicional 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.

FaceRecon en ocasiones reconoce algunos objetos como rostros, como se ve en la Figura 42. Esto se ha tenido en cuenta para que no los muestre como sujeto desconocido en el resultado final.



Figura 42 Objetos identificados como rostros.

Estos artefactos son enviados a Azure, si bien se obtiene una respuesta negativa como rostro no detectado y no se realizan más operaciones.

Fácil uso

FacePal dispone de una interfaz simple e intuitiva. Sus características más destacables son:

- 3 opciones de configuración, solo texto.
- 3 botones en la pantalla principal y 2 en la de entrenamiento.

Se ha mantenido la aplicación lo más simple posible, sin controles u opciones adicionales ni ningún elemento que haga al usuario final dudar o no entender. En las pruebas se dio la aplicación a dos personas no relacionadas con el desarrollo y no tuvieron ningún problema en manejarla.

FaceRecon no dispone de un interfaz como tal, solo mensajes saliendo por pantalla y un directorio con las imágenes guardadas, 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, puede realizarse con una interfaz sencilla o muy avanzada, dependiendo del uso final.

Fácil incorporación

FacePal se puede instalar fácilmente en casi cualquier dispositivo con sistema Android moderno, aunque subir la aplicación a Play Store hará la instalación mucho más sencilla.

La versión de FaceRecon diseñada para este proyecto es fácilmente incorporable, solo necesita conectar los cables de alimentación y video a la Raspberry Pi y acoplarlo donde sea necesario. Si forma parte de un proyecto distinto habrá que acoplarlo a la aplicación o plataforma que se desee, esto puede requerir conocimientos adicionales.

Fácil mantenimiento

El código de ambas aplicaciones está debidamente comentado y se ha usado la compartimentalización siempre que ha sido posible, creando funciones fácilmente sustituibles o actualizables sin tener que alterar el resto del programa.

FaceRecon se ha diseñado de forma que la aplicación principal no contenga ninguna función, todas son accesibles usando el módulo FCmodule de forma que no se altera para realizar ampliaciones.

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.

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 + PiCamera tiene un precio inferior a 70€ lo cual implica que el uso de este equipo en diversos proyectos no implica una perturbación del presupuesto. Con respecto al teléfono móvil, no hace falta un móvil concreto, la mayor parte de los teléfonos con Android podrán tener FacePal instalado. Para un usuario común esto no es un gasto adicional si dispone de dicho teléfono.

El plan de Azure empleado para la elaboración de este proyecto es gratuito; sin embargo, para un uso cotidiano implica unos gastos por consumo.

Existen varias combinaciones, pero se puede emplear como ejemplo la siguiente situación:

Una empresa de 200 empleados quiere emplear el sistema de reconocimiento 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 FaceRecon, es decir 2 Raspberry Pi y 2 PiCamera.

Por cada empleado se toman 3 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 2 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
- búsqueda del nombre real

En total 1600 transacciones al día, 49600 transacciones al mes.

Se puede desglosar el gasto total en gasto inicial y gasto mensual:

Concepto	Unidades	Precio unidad	Total
Raspberry pi	2	40 €	80 €
PiCamera	2	30 €	60 €
Entrenamiento	2400	0.844 € / 1000 transacciones	2.02 €
TOTAL			142.02 €

Concepto	Unidades	Precio unidad	Total
Transacciones	49600	0.844 € / 1000 unidades	41.86 €
Almacenamiento de imágenes	600 (3 imágenes por empleado)	0.211 € / 1000 imágenes	0.211 €
TOTAL			42.07 €

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

Conclusión

El sistema diseñado cumple en alto grado las metas marcadas inicialmente, las pruebas del sistema demuestran que funciona como se esperaba, incluso mejor en algunos aspectos.

El cambio de Watson a Azure como servicio principal fue muy positivo, Azure funciona mucho mejor con menos fotografías por persona y con mayor rapidez.

Si bien el sistema puede ser ampliado, tal y como se ha implementado satisface los requisitos del proyecto.

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 bug en algunos teléfonos de Android que impiden 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 90° con respecto a cómo se tomó.

La alternativa de futuro es cargar la imagen, al pulsar con el dedo rotar la imagen 90° y realizar la búsqueda de rostros. Tras 4 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 eviten transferencias a Azure.

Como se ha visto anteriormente, Azure no devuelve el nombre de las personas al identificar un rostro, devuelve su ID. En este proyecto es necesario buscar la lista de individuos del grupo y contrastar cada ID con el ID deseado, esto 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 Azure cuando se quiera obtener el nombre real. 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 puede ampliarse para realizar gestiones mayores como eliminar usuarios, crear otros grupos de personas, etc. Si bien esta ampliación debería solo hacerse en una versión de administrador para evitar gestiones inoportunas del sistema. Esta versión de administrador podría realizarse en Windows fácilmente empleando las funciones dentro de FCTools que eliminan, añaden, entrenan, etc. Solo necesitaría una interfaz gráfica adecuada o incluso podría crearse un servicio web combinando HTML y PHP ejecutando scripts de Python en el lado del servidor.

FaceRecon ofrece incluso mayor potencial. Usando las funciones de FCmodule se pueden crear aplicaciones que realicen una o varias de las siguientes operaciones:

- devolución solo de nombres,
- devolución solo de caras recortadas,
- que envíe la información por correo electrónico,
- envío de los datos por Bluetooth con otro dispositivo a la escucha,
- 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 a un personal con mayor conocimiento de software y de programación.

Una idea que fue descartada durante la fase de análisis fue construir un servidor capaz de realizar el servicio de reconocimiento. Sin embargo, la complejidad de construir y montar todo el sistema quedó fuera del alcance de este proyecto debido a la limitación de tiempo para el desarrollo de este proyecto.

No obstante, el uso de un servicio profesional como es Azure sigue la línea de evitar el anti-patrón conocido como reinventar la rueda.

El tándem del sistema lo hace muy flexible para muchas aplicaciones. Un usuario experto o diseñador prepara la tarea de su FaceRecon y usuarios con poco conocimiento entrenan el sistema con FacePal. El ejemplo del sistema de fichaje moderno propuesto en el apartado anterior es una buena demostración de este tándem.

Bibliografía

Bradsky G., Kaehler A. Learning OpenCV. O'Reilly. 2008

Szeliski R. Computer Vision: Algorithms and Applications. Springer. 2010

Viola P., Jones M. J. "Robust Real-Time Face Detection", International Journal of Computer Vision 57(2), 137–154, 2004

Android developer (Manual de clases Android)

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

Camera Module - Raspberry Pi Documentation

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

Detección de objetos (MOOC)

<https://www.coursera.org/learn/deteccion-objetos>

Face API Documentation (Documentación Azure)

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

httpClient-android

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

Mobile Vision

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

OpenCV

<https://opencv.org/>

Stack Overflow

<https://stackoverflow.com/>

Wrong picture orientation on Samsung devices (Bug Samsung)

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

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.

LBP (Local Binary Patterns): descriptor visual empleado para la clasificación en visión artificial.

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.

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.

Anexo 1 – Materiales y métodos

Plataformas hardware

Raspberry Pi 3 modelo B: implementación de FaceRecon.

Samsung Galaxy S3: implementación de FacePal.

Samsung Galaxy A3 2017: pruebas secundarias de FacePal.

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

Servicios online:

Microsoft Azure: reconocimiento facial.

Github: control de versiones de programas.

Software:

Android Studio: desarrollo de aplicaciones.

Dia: desarrollo de esquemas y diagramas.

IDLE (Python 3.6): desarrollo en Python.

Notepad++: edición de código.

Powerpoint: elaboración de figuras.

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

Windows 10: sistema operativo.

Word: elaboración de la memoria.

Material suplementario:

El proyecto se ha dividido en varios directorios para compartimentalizar 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.

src/facePal

Código fuente de FacePal, contiene todos los archivos para cargar este proyecto desde Android Studio.

src/FaceRecon

Código fuente de FaceRecon, contiene los scripts listos para usar de esta aplicación.

test

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

Anexo 2 - Preparación de Raspbian

Para que FaceRecon funcione en la Raspberry Pi, es necesario instalar y actualizar el sistema Raspbian y tener las dependencias necesarias instaladas.

Conexión de la cámara



Figura 43 Conexión de cámara (Fuente adafruit.com).

Actualización del sistema

Se deben ejecutar los siguientes comandos y en el orden indicado

```
sudo apt-get update
```

```
sudo apt-get dist-upgrade
```

Dependencias de FaceRecon (Python 3.6)

```
sudo pip3 install numpy
```

```
sudo pip3 install opencv-python
```

```
sudo pip3 install cognitive_face
```