

Universidad de Sevilla. Escuela Politécnica Superior de Sevilla



Escuela Politécnica Superior

Trabajo Fin de Grado en Ingeniería Electrónica Industrial

Diseño e implementación de un sistema de videoportero con reconocimiento facial para computadores empotrados

Autora: Loreto Oliva Gutiérrez
Tutor: Daniel Cagigas Muñiz

Fecha: 21/07/2023

Diseño e implementación de un sistema de videoportero con reconocimiento facial para computadores empotrados

Autora: Loreto Oliva Gutiérrez
Tutor: Daniel Cagigas Muñiz

Resumen del trabajo

En este proyecto se ha desarrollado un prototipo de videoportero automático con la posibilidad de abrir la puerta desde una página web si el usuario no es reconocido por el programa. Para ello, el hardware empleado ha sido una Raspberry Pi 2, una webcam y un teléfono móvil. Para simular la apertura de la puerta se ha usado un LED, una resistencia, cables y una placa de pruebas. En lo referente al software, se ha empleado Python y OpenCV para el reconocimiento facial, Django para desarrollar la aplicación web y MariaDB como base de datos para la trazabilidad de los intentos y la eficiencia del modelo.

Palabras clave

Videoportero automático, reconocimiento facial, Raspberry Pi, OpenCV, Haar Cascade, Python, MariaDB, Django, JavaScript, Ajax.

Abstract

This project consists in an automatic video door station with the chance of opening the door from a website if the person could not be recognized by the script. The hardware used has been a Raspberry Pi 2, a webcam, and a mobile phone. The software used has been Python and OpenCV for facial recognition, Django to develop the web application and MariaDB as database to be able to have a traceability of the performance and efficiency of the model.

Keywords

Automatic video door station, facial recognition, Raspberry Pi, OpenCV, Haar Cascade, Python, MariaDB, Django, JavaScript, Ajax.

ÍNDICE

ÍNDICE	4
1. RESUMEN	6
2. INTRODUCCIÓN	7
3. ESTADO DEL ARTE.....	8
4. MATERIALES, TÉCNICAS Y MÉTODOS	10
4.1. Requerimientos hardware	10
4.2. Requerimientos de software.....	15
5. DISEÑO DEL SISTEMA.....	18
6. IMPLEMENTACIÓN DE LA SOLUCIÓN	21
6.1. Preparación del sistema operativo	21
6.2. Reconocimiento facial	28
6.2.2. Comprobación cámara	30
6.2.3. Detección facial	30
6.2.4. Recolección de datos	32
6.2.5. Entrenamiento del modelo.....	32
6.2.6. Reconocimiento facial.....	33
6.2.7. Integración botón y encendido LED.....	34
6.3. Desarrollo web y base de datos	37
6.3.1. Configuración IP fija.....	37
6.3.2. Instalación MariaDB.....	38
6.3.3. Instalación y preparación del entorno de desarrollo	38
6.3.4. Configuración del entorno.....	41
6.3.5. Desarrollo del entorno web.....	44
6.3.6. Registro de entradas	50
6.3.7. Envío de notificación a Telegram	51
7. PRUEBAS REALIZADAS.....	56
8. ANÁLISIS ECONÓMICO DE LA SOLUCIÓN.....	64
9. ANÁLISIS TEMPORAL	68
9. MANUAL DE DESPLIGUE DE LA APLICACIÓN Y USUARIO	71
9.1. Despliegue e instalación de la aplicación.....	71
9.2. Manual de uso de la aplicación	75
10. CONCLUSIONES.....	76
11. TRABAJO FUTURO	77
12. BIBLIOGRAFÍA.....	79

ANEXO.....	83
Anexo 1. Toma de muestras	83
Anexo 2. Entrenamiento del modelo.....	85
Anexo 3. Reconocimiento facial.....	86
Anexo 4. Encendido de LED.....	88
Anexo 5. Comprobación de pulsación pulsador.....	89
Anexo 6. Fichero que contiene las URL de Django	90
Anexo 7. Fichero que contiene las vistas de Django	91
Anexo 8. Estructura página web	92
Anexo 9. Página web.....	93
Anexo 10. Registro de entrada a base de datos.....	95
Anexo 11. Envío mensaje a Telegram.....	96

1. RESUMEN

El proyecto está enfocado en el desarrollo de un prototipo de videoportero automático que sea capaz de reconocer caras previamente registradas, y abra la puerta automáticamente si reconoce a la persona. Si la cara no es reconocida, está la posibilidad de abrir la puerta mediante una página web donde se puede ver el vídeo en tiempo real para comprobar quién hay en la puerta, y además un registro de las últimas entradas e intentos de reconocimiento.

Para llevarlo a cabo se ha elegido una Raspberry Pi 2 (en adelante Raspberry Pi) como microordenador por su precio y versatilidad. Como cámara se ha escogido una webcam, pero también se podría implementar con la PiCam¹.

Para simular la apertura de la puerta se ha usado una placa de pruebas con un LED y una resistencia, y cables conectados a puertos de la Raspberry Pi.

Para lanzar el reconocimiento facial se ha usado un pulsador conectado a un puerto de la Raspberry Pi.

En cuanto al software, se pueden diferenciar varias partes:

Por un lado, está el reconocimiento facial. Para implementarlo se ha usado Python y OpenCV, por ser software libre muy utilizado con gran cantidad de librerías.

Por otra parte, para la página web se ha usado Django, un entorno de desarrollo de aplicaciones web. Se ha elegido por usar Python como lenguaje y por su capacidad ejecutar programas de Python enviando y recibiendo parámetros de estos. Al estar todo lo relacionado con el reconocimiento facial y la apertura de la puerta programado en Python, hace mucho más fácil la comunicación entre la web, el programa de reconocimiento facial y el programa de apertura de la puerta.

Finalmente, se encuentra la base de datos. Se ha escogido MariaDB como sistema de base de datos de Django por ser software libre, muy eficiente y de uso muy extendido. MariaDB se comunica con Python y con Django de forma efectiva, por lo que tanto el programa de reconocimiento facial como la web pueden lanzar consultas a la base de datos. Se ha usado esta funcionalidad para llevar una trazabilidad de los reconocimientos, monitoreando la fecha y hora del intento de reconocimiento, si reconoció a algún usuario y si el reconocimiento fue exitoso. De esta forma también se puede tener una idea de tasa de fallo del reconocimiento y si hay algún usuario que necesite actualizar su cara para un mejor reconocimiento.

¹ <https://www.electronicwings.com/raspberry-pi/pi-camera-module-interface-with-raspberry-pi-using-python>

2. INTRODUCCIÓN

Con el avance de la tecnología cada vez hay más soluciones de videoportero para poder ver de una forma rápida y cómoda quién hay en la puerta de casa e incluso abrir o comunicarse con esa persona mediante el teléfono móvil o una tableta. Y es que los dispositivos móviles ya son una parte fundamental en la vida de las personas y hay muchas aplicaciones que se pueden desarrollar para que continúen haciendo la vida más fácil.

Por otro lado, la inteligencia artificial está avanzando a grandes pasos, “amenazando” con cambiar tanto la industria como la vida cotidiana de las personas de la forma en que la conocemos. Poco a poco, la domótica nos rodea cada vez más, y ya es muy común, por ejemplo, tener asistentes de voz en casa tales como Alexa². Hacer la lista de la compra con Alexa, crear rutinas de luces, cambiar de canal de televisión o poner la lavadora a una hora determinada aún sin estar en casa nunca había sido tan fácil.

Desde hace poco, y con la llegada de aplicaciones tipo ChatGPT³, se abre paso también en el entorno laboral de la mano del concepto de Industria 4.0. Gigantes como Google, Amazon y Windows están en la carrera por controlar este nuevo mercado.

Por tanto, se puede afirmar que la inteligencia artificial ya es una realidad, hay muchísimas aplicaciones que están aún por descubrirse y tiene un gran potencial aún por desarrollar. Esta es la razón por la cual se ha intentado aplicar el uso de esta tecnología disruptiva en mejorar una aplicación ya existente.

El objetivo de este proyecto es ir un paso más allá en el campo de los videoporteros y aplicar también la inteligencia artificial, y más en concreto las redes neuronales, con el fin de lograr un videoportero que sea capaz de reconocer y abrir la puerta automáticamente a los usuarios que ha sido entrenado para reconocer.

Además, es una solución de bajo coste de implementación ya que los requerimientos de hardware son económicos, el software utilizado es libre y en todos los hogares hoy en día se dispone de wifi y dispositivos móviles, ordenadores o incluso Smart TV donde se puede ver la página web en el navegador.

² <https://developer.amazon.com/es-ES/alexa>

³ <https://es.wikipedia.org/wiki/ChatGPT>

3. ESTADO DEL ARTE

El videoportero es una solución muy extendida actualmente y ya ha reemplazado por completo a portero tradicional en las nuevas construcciones. La ventaja frente al portero tradicional es que permite ver quién está en la puerta antes de decidir si abrir o no, y esto aporta más seguridad a la hora de saber quién pretende entrar en casa.

Los más comunes y económicos son porteros convencionales con una cámara integrada en la placa de la puerta, para que desde dentro se pueda ver en el monitor quién hay fuera.

A partir de ahí han ido evolucionando para mostrar imagen y sonido en las tabletas y teléfonos móviles, y hoy en día no es necesario estar dentro de la casa para poder abrir la puerta a alguien o comunicarse con esa persona que pretende entrar, si no que desde una aplicación móvil se puede abrir la puerta de la calle cuando se desee.



Figura 1. Ejemplo de uso de teléfono móvil como videoportero⁴

Los más sofisticados pueden grabar vídeo y actúan como cámara de videovigilancia que se activan con ciertos estímulos externos. Estos últimos son bastante más caros, ya que requieren de más tecnología, sensores y espacio para ir almacenando los vídeos tomados.

En resumen, el videoportero ha ido avanzando para aportar más comodidad a la hora de abrir la puerta, pero sobre todo más seguridad a la hora de evitar fraudes e incluso disuadir delincuencia por la grabación de video. Por supuesto, es algo únicamente complementario a los sistemas de vigilancia y alarma de las casas, y no deberían sustituir a los mismos.

⁴ Fuente: <https://romotelecom.com/videoporteros-y-control-de-accesos/>

En cuanto al marco legal, ha de tenerse en cuenta la ley de protección de datos. No es de aplicación para el videoportero convencional, pero sí cuando se esté grabando constantemente. En este proyecto, como se reproduce en tiempo real las imágenes de lo que está pasando en la vía pública habría que acogerse a esta ley.

“La normativa de protección de datos no es de aplicación cuando se trate de tratamientos mantenidos por personas físicas en el ejercicio de actividades exclusivamente personales o domésticas, como ocurre, por ejemplo, cuando el tratamiento sea efectuado a través de videoporteros.

Sin embargo, si será de aplicación cuando el servicio se articule mediante procedimientos que reproducen y/o graban imágenes de modo constante, y resultan accesibles -ya sea a través de Internet o mediante emisiones por la televisión de los vecinos-, y en particular cuando el objeto de las mismas alcance al conjunto del patio y/o a la vía pública colindante.

En consecuencia, cuando una cámara permite reproducir en tiempo real las imágenes que concurren en la portería de un edificio, su actuación excede con mucho del ámbito personal y doméstico, por lo que implica un tratamiento de datos de carácter personal, sujeto a la normativa de protección de datos.”⁵

En la “ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales”[12] puede encontrarse más información al respecto.

Este proyecto está pensado para casas particulares, pero en caso de querer implementarlo en una comunidad de vecinos habría que tener en cuenta, además, la ley de propiedad horizontal⁶.

⁵Fuente: <https://www.aepd.es/es/preguntas-frecuentes/9-comunidades-de-propietarios/FAQ-0912-la-normativa-de-proteccion-de-datos-se-aplica-a-los-videoporteros>

⁶ <https://www.boe.es/buscar/act.php?id=BOE-A-1960-10906>

4. MATERIALES, TÉCNICAS Y MÉTODOS

Se procederá a explicar con detalle los requerimientos de hardware y las tecnologías y software empleados en la implementación del proyecto.

4.1. Requerimientos hardware

Los componentes de hardware necesarios para la implementación del sistema son los siguientes:

Raspberry Pi 2 Model B: es un miniordenador de bajo coste que consta de puertos USB, Ethernet, HDMI y 40 pines GPIO (entradas/salidas de propósito general) entre otras características. Monta un procesador ARM Cortex-A7 Quad-Core a 900 MHz con 1 GB de RAM que es más que suficiente para el propósito del proyecto.



Figura 2. Raspberry Pi 2 Model B⁷

Para más información sobre las características de esta placa visitar <https://amzn.eu/d/0R6bQwp>

Fuente alimentación micro USB: se necesita para conectar la Raspberry Pi a la corriente. Es recomendable que sea de al menos 1mA.

⁷ Fuente: <https://amzn.eu/d/0R6bQwp>



Figura 3. Cargador micro USB⁸

Tarjeta de memoria Micro SD: es donde irá instalado el sistema operativo de la Raspberry Pi, se recomienda que sea de clase 10 y de 32 GB.



Figura 4. Tarjeta de memoria Micro USB⁹

Adaptador WiFi USB inalámbrico: generalmente el router no estará cerca de la puerta por lo que será necesaria la conexión por WiFi.



Figura 5. Adaptador WiFi USB inalámbrico¹⁰

Teclado: vale cualquiera que funcione por USB.

⁸ Fuente: <https://amzn.eu/d/hinyVeR>

⁹ Fuente: <https://amzn.eu/d/eJpptFO>

¹⁰ Fuente: <https://amzn.eu/d/1Yj62eU>



Figura 6. Teclado¹¹

Ratón: al igual que el teclado, es válido cualquier ratón que funcione por USB.



Figura 7. Ratón¹²

Monitor: es válido cualquier monitor con conexión HDMI.



Figura 8. Monitor¹³

Cable HDMI: para conectar la Raspberry al monitor.

¹¹ Fuente: <https://amzn.eu/d/dRzcfmY>

¹² Fuente: <https://amzn.eu/d/hm2im6A>

¹³ Fuente: <https://amzn.eu/d/hxv0wKp>



Figura 9. Cable HDMI¹⁴

Cámara web: para el desarrollo del proyecto se ha usado la webcam EyeToy de la PlayStation 2, pero sirve cualquiera cámara web que funcione por USB.



Figura 10. WebCam EyeToy¹⁵

Teléfono móvil: en este caso se ha usado un OnePlus 9 Pro para las pruebas, pero vale cualquier teléfono móvil o tableta (sin importar el sistema operativo) conectado a la red WiFi y con acceso al navegador. Para recibir la notificación es necesario que también tengan Telegram instalado (se explicará como instalarlo posteriormente paso por paso).



Figura 11. Teléfono móvil OnePlus 9 Pro¹⁶

¹⁴ Fuente: <https://amzn.eu/d/4ohBRIK>

¹⁵ Fuente: <https://amzn.eu/d/cRHv9GJ>

¹⁶ Fuente: <https://amzn.eu/d/4L5wWX0>

LED: se usará para simulación de la apertura de la puerta.



Figura 12. LED¹⁷

Resistencia: para limitar la corriente que pasa a través del LED y así evitar dañarlo. Una resistencia de 220 ohm funcionará perfectamente.



Figura 13. Tira de resistencias¹⁸

Pulsador: para activar el reconocimiento facial.



Figura 14. Pulsador momentáneo¹⁹

Router wifi: necesario para poder ver la página web desde los dispositivos.

¹⁷ Fuente: <https://kumotica.es/componentes-robotica/135-lote-100-diodos-led-5-mm-azul.html>

¹⁸ Fuente: <https://amzn.eu/d/2KUGFyk>

¹⁹ Fuente: <https://es.aliexpress.com/item/1005004159746274.html?channel=twiner>



Figura 15. Router wifi²⁰

Placa de pruebas y cables: para conectar los componentes con la Raspberry Pi.

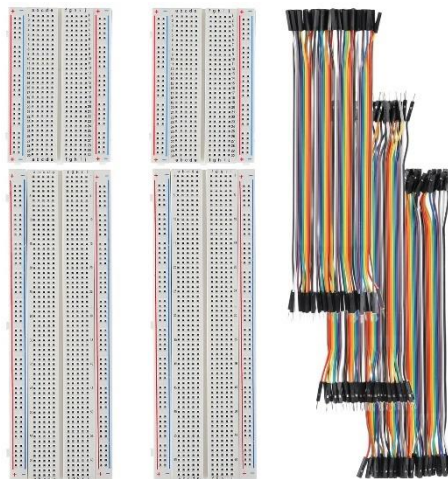


Figura 16. Placa de pruebas y cables²¹

Teclado, ratón y monitor han sido necesarios para la implementación y programación del sistema, pero una vez está todo instalado y funcionando no es necesario tenerlos conectado a la Raspberry.

4.2. Requerimientos de software

El software elegido para la Raspberry Pi ha sido Raspberry Pi OS por ser el oficial y más adaptado para funcionar de forma óptima en la Raspberry. Raspberry Pi OS.

“Al ser una distribución de GNU/Linux las posibilidades son infinitas. Todo software de código abierto puede ser recompilado en la propia Raspberry Pi para arquitectura armhf que pueda utilizarse en el propio dispositivo en caso

²⁰ Fuente: <https://amzn.eu/d/j1Wlx0b>

²¹ Fuente: <https://amzn.eu/d/4CRQpe2>

de que el desarrollador no proporcione una versión ya compilada para esta arquitectura. Además esta distribución, como la mayoría, contiene repositorios donde el usuario puede descargar multitud de programas como si se tratase de una distribución de GNU/Linux para equipos de escritorio. Todo esto hace de Raspberry Pi un dispositivo que además de servir como placa con microcontrolador clásica, tenga mucha de la funcionalidad de un ordenador personal.”²²

Como lenguaje de programación se va a usar Python. Python es un lenguaje con un uso muy extendido y unas características que lo hacen muy potente, además de ser código abierto:

“Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo, ejemplos: Instagram, Netflix, Spotify, Panda3D, entre otros.² Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma”²³

También se usará OpenCV²⁴, que es una librería para Python que permite el procesamiento de imágenes y encaja bastante bien con las necesidades del proyecto para hacerlo eficiente y óptimo.

Como base de datos se descarga y configura MariaDB²⁵, una base de datos muy usada y código abierto. MariaDB es relativamente nueva, pero se ha elegido frente a MySQL porque MariaDB es completamente código abierto y MySQL tiene módulos de código cerrado. Esta nueva base de datos es desarrollada por los mismos ingenieros que desarrollan MySQL y en algunos aspectos tiene mejor rendimiento.

Por último, para desarrollar la página web se ha usado Django²⁶. Django es un entorno de Python muy potente para el desarrollo de páginas web. Los lenguajes usados para el desarrollo web han sido Python, JavaScript²⁷, HTML²⁸ y CSS²⁹.

A grandes rasgos, Django se basa en un sistema de peticiones al servidor y respuestas de este. Al escribir la URL en el navegador se está enviando una petición al servidor, que realiza una serie de acciones y devuelve como respuesta una URL. Se puede ver un poco más claro en el siguiente esquema:

²² Fuente: https://es.wikipedia.org/wiki/Raspberry_Pi_OS

²³ Fuente: <https://es.wikipedia.org/wiki/Python>

²⁴ <https://es.wikipedia.org/wiki/OpenCV>

²⁵ <https://mariadb.org/>

²⁶ [https://es.wikipedia.org/wiki/Django_\(framework\)](https://es.wikipedia.org/wiki/Django_(framework))

²⁷ <https://developer.mozilla.org/es/docs/Web/JavaScript>

²⁸ <https://developer.mozilla.org/es/docs/Web/HTML>

²⁹ <https://developer.mozilla.org/es/docs/Web/CSS>

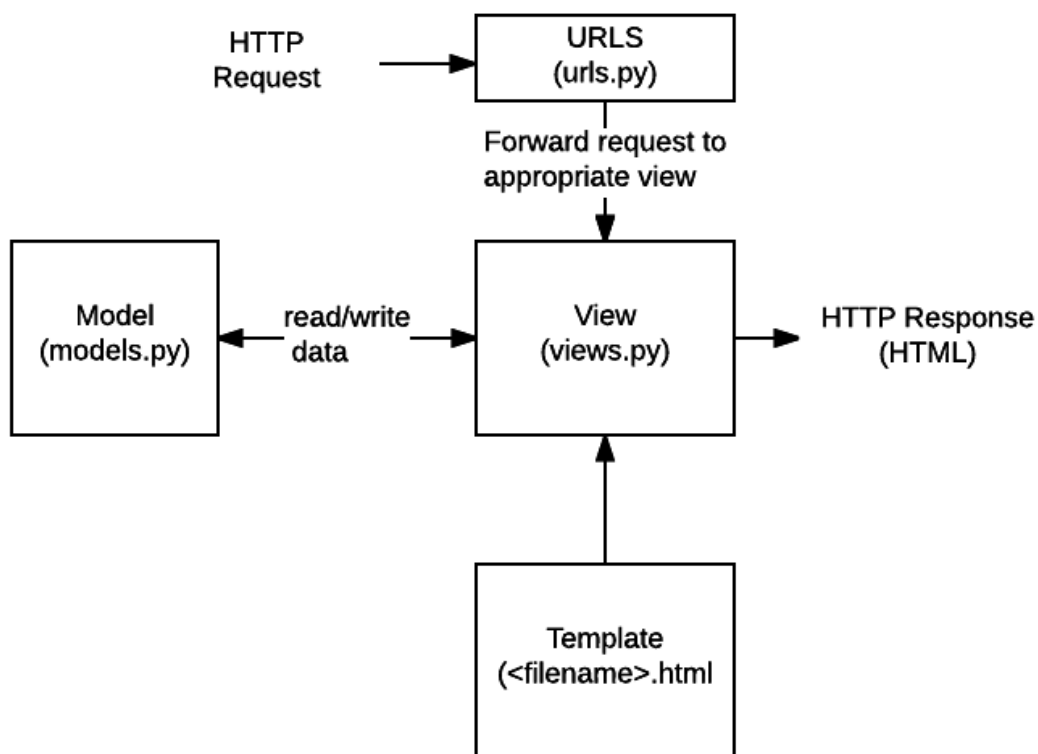


Figura 17. Esquema estructura Django³⁰

Llega una petición desde el navegador y esta es gestionada por el fichero urls.py, que se encarga de identificar la petición y enviarla a la vista correspondiente.

En el fichero views.py se encuentran las vistas, que es donde se realizan las acciones que se hayan programado para cada petición (aquí es donde se pueden ejecutar otros Scripts, leer y escribir en la base de datos, mostrar el vídeo en tiempo real...).

Si hay peticiones de lectura o escritura de la base de datos estas consultas se hacen en las vistas con los modelos creados en el fichero models.py, donde se modelan las tablas y sus componentes.

Una vez ejecutado el código correspondiente en la vista, esta devuelve una respuesta en forma de URL y la muestra en el navegador.

Durante el desarrollo del proyecto se irá explicando más en profundidad cómo se configura Django y cada uno de sus ficheros.

³⁰ Fuente: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Home_page

5. DISEÑO DEL SISTEMA

En este apartado se hace una descripción del flujo de información/datos del sistema y como están relacionados, ya que se pueden diferenciar varias partes independientes. El reconocimiento facial, la gestión de la apertura de la puerta mediante el navegador web y la trazabilidad en la base de datos son aspectos independientes, pero estrechamente relacionados para cumplir la función.

En el siguiente diagrama de flujo se observa el funcionamiento general:

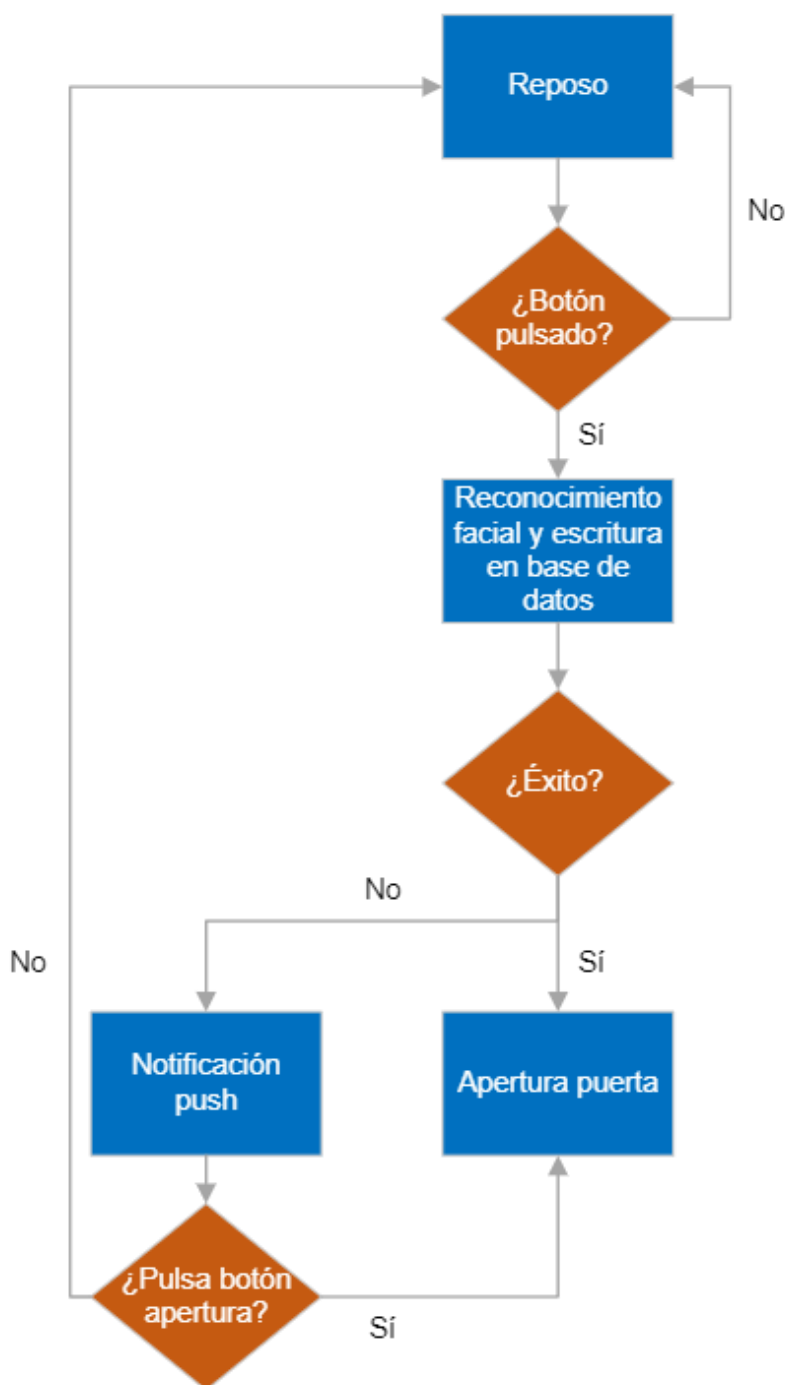


Figura 18. Diagrama de flujo funcionamiento

Cuando se pulsa el pulsador de apertura de la puerta, se activa automáticamente el programa de reconocimiento facial y durante un tiempo el programa está intentando reconocer al usuario. Si lo reconoce con una confianza mayor al 70%, ejecutará el programa de apertura de la puerta y se encenderá el LED. Si el tiempo pasa y el usuario no es reconocido, o no lo es con una confianza superior al 70%, se requerirá la apertura manual desde el navegador web.

Independientemente de si el reconocimiento ha sido exitoso o no, se registrará en la tabla de la base de datos una entrada con la fecha y hora, el nombre del usuario si lo ha reconocido, el porcentaje de confianza y si ha sido exitoso (si se ha encendido el LED).

Si el reconocimiento no ha sido exitoso se enviará una notificación vía servicio de mensajería móvil Telegram a los dispositivos registrados para alertar de que hay alguien en la puerta esperando para entrar. Desde el navegador de cualquiera de los dispositivos se puede ver quién hay en la puerta y decidir si pulsar el botón de abrir o no.

En cuanto a la estructura del programa, la parte del reconocimiento facial y la página web son totalmente independientes. Para el reconocimiento facial se ha creado un fichero de Python que está constantemente comprobando si se pulsa el botón, y si se pulsa comienza el reconocimiento facial. El programa del reconocimiento facial es el encargado de lanzar los distintos programas: el programa para mandar el registro a la base de datos y el programa de abrir la puerta o el de enviar la notificación a Telegram según el resultado del reconocimiento.

En la página web hay solo una URL por el momento donde se gestiona todo (la página de inicio), pero se ha preparado la estructura para que se puedan añadir más páginas en el futuro. Al cargar la página de inicio se ejecutan las vistas para mostrar el vídeo de lo que sucede en la puerta y para mostrar la tabla de últimos accesos. Si se pulsa el botón de abrir la puerta se ejecuta otra vista más, que es la ejecuta el programa de encender el LED.

Se va a modificar el esquema de la estructura de Django mostrado anteriormente para adaptarlo a la estructura del proyecto:

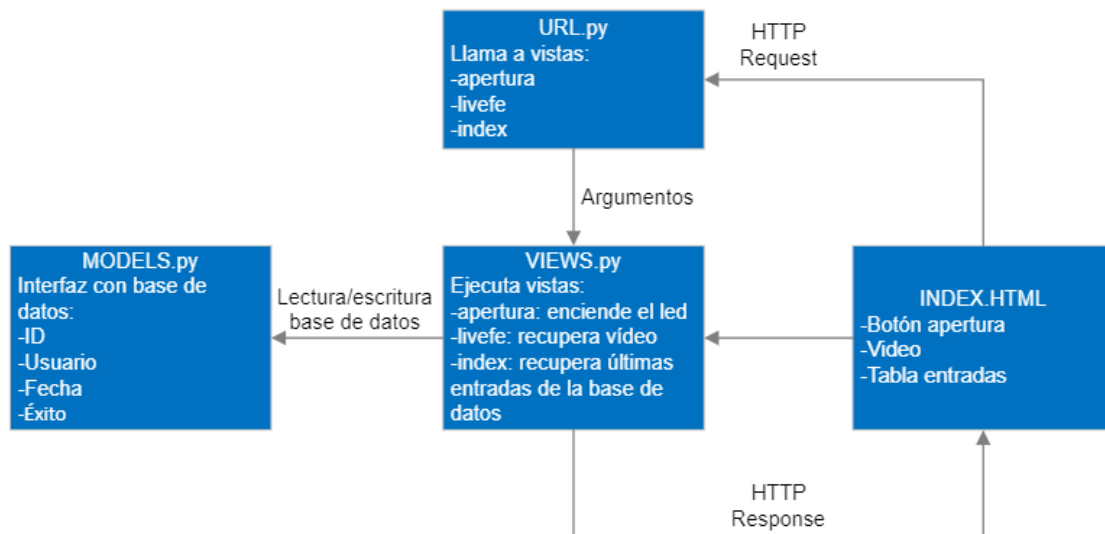


Figura 19. Esquema estructuración Django

Al cargar la URL en el navegador, este envía una petición al servidor que es gestionada por el fichero urls.py. Dependiendo de la URL recibida derivará la consulta a la vista adecuada.

Cuando carga la página de inicio se llaman a dos vistas llamadas “index” y “livefe”. La primera se encarga de mostrar la tabla de registros, cuyos parámetros están modelados en models.py y es la que devuelve la URL de la página index.html. La vista llamada “livefe” se encarga de mostrar el vídeo.

Si se pulsa el botón de apertura en la página web se lanza una petición que llama a la vista “apertura”, la cual se encarga de ejecutar el programa que enciende el LED.

6. IMPLEMENTACIÓN DE LA SOLUCIÓN

6.1. Preparación del sistema operativo

El primer paso para la implementación del sistema es instalar el sistema operativo en la Raspberry Pi y configurarlo. Esto se ha llevado a cabo en un portátil con Windows 10, pero se podría hacer de manera análoga desde cualquier otro sistema operativo.

Previamente es necesario tener la tarjeta microSD formateada en formato FAT32. En un pc que tenga ranura para tarjeta de memoria o con un adaptador a USB, se introduce la tarjeta en el PC. En “Este equipo” (se puede acceder mediante el explorador de archivos o buscando directamente en el buscador) aparecen los discos externos conectados.

Una vez identificada la tarjeta de memoria, haciendo click con el botón derecho sobre ella se selecciona la opción “Formatear...”.

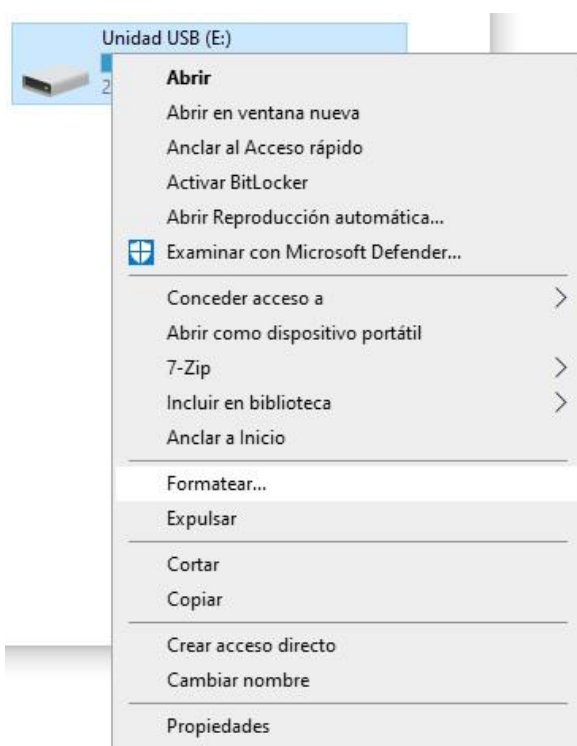


Figura 20. Captura pantalla opciones dispositivo microSD

En la nueva pantalla que aparece hay que seleccionar la siguiente configuración: en el apartado “Sistema de archivos” se selecciona “FAT32”, en “Tamaño de unidad de asignación” se seleccionan “4096 bytes” y se marca la casilla “Formato rápido”. Una vez hecha la selección, hacer click en “Iniciar” y se formateará.

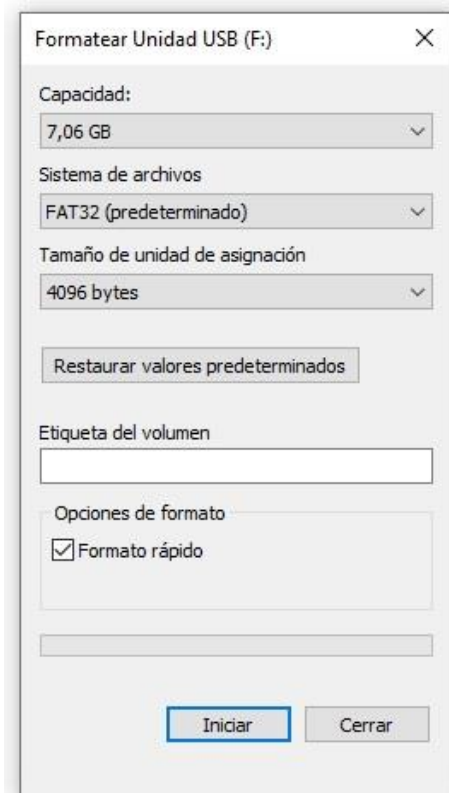


Figura 21. Captura configuración formato microSD

Por otra parte, es necesario descargar el sistema operativo que correrá la Raspberry Pi. Como se describió previamente, se ha elegido Raspberry Pi OS como sistema operativo, el cual se puede descargar directamente desde el sitio web oficial de Raspberry.

En la web <https://www.raspberrypi.com> se puede encontrar más información acerca del sistema operativo y en la pestaña "Software", se puede descargar de forma gratuita. Hay que elegir una opción de descarga dependiendo del sistema operativo del pc, en este caso Windows 10.

Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. [Watch our 45-second video](#) to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Windows](#)

[Download for macOS](#)

[Download for Ubuntu for x86](#)

To install on **Raspberry Pi OS**, type `sudo apt install rpi-imager` in a Terminal window.

Figura 22. Opciones de descarga SO

Una vez descargado, y aún con la tarjeta microSD insertada en el pc, se procede a la instalación de Raspberry Pi OS en la tarjeta de memoria. Para ello, se ejecuta el instalador que anteriormente hemos descargado (estará en la carpeta de descargas) y se siguen los pasos de instalación que se indican a continuación:

Al ejecutar el instalador aparece una nueva pantalla donde habrá que seleccionar el sistema operativo a instalar y la unidad en la que se desea hacerlo.



Figura 23. Instalación SO

Para elegir el sistema operativo se hace click en “Choose OS” y se selecciona “Raspberry Pi OS”.

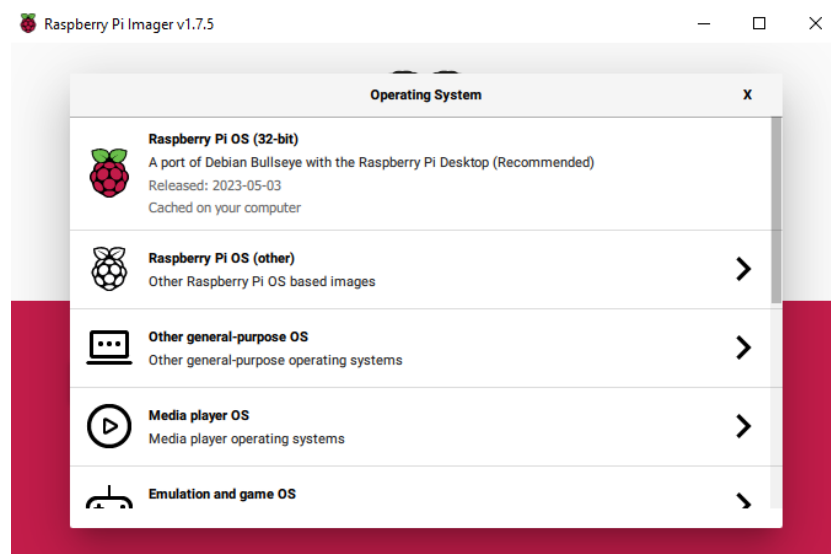


Figura 24. Selección sistema operativo

Para elegir la unidad en la que instalarlo se hace click en “Choose Storage” y se selecciona la tarjeta de memoria previamente formateada en la ventana emergente.

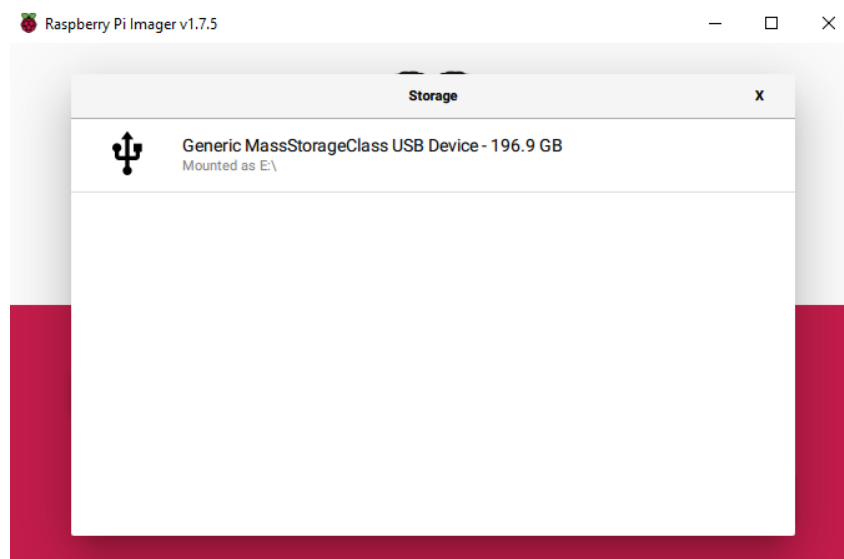


Figura 25. Selección unidad de instalación

Una vez seleccionado sistema operativo y la unidad, se hace click en “Write” y comenzará la instalación. Una vez completada, aparecerá una ventana emergente con la que se muestra a continuación y la tarjeta de memoria ya estará lista para introducirla en la Raspberry Pi.

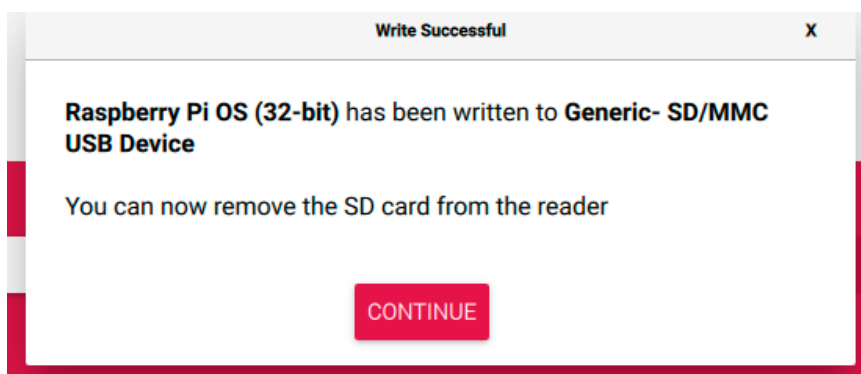


Figura 26. Instalación sistema operativo completa

Completada la instalación Raspberry Pi OS es momento de expulsar la tarjeta de memoria del pc e insertarla en la Raspberry PI. Para arrancar y configurar el sistema operativo habrá que conectar la Raspberry Pi a la corriente, a un monitor con un cable HDMI y teclado, ratón y adaptador WiFi conectados a los puertos USB. A continuación, se muestra un esquema de las conexiones:

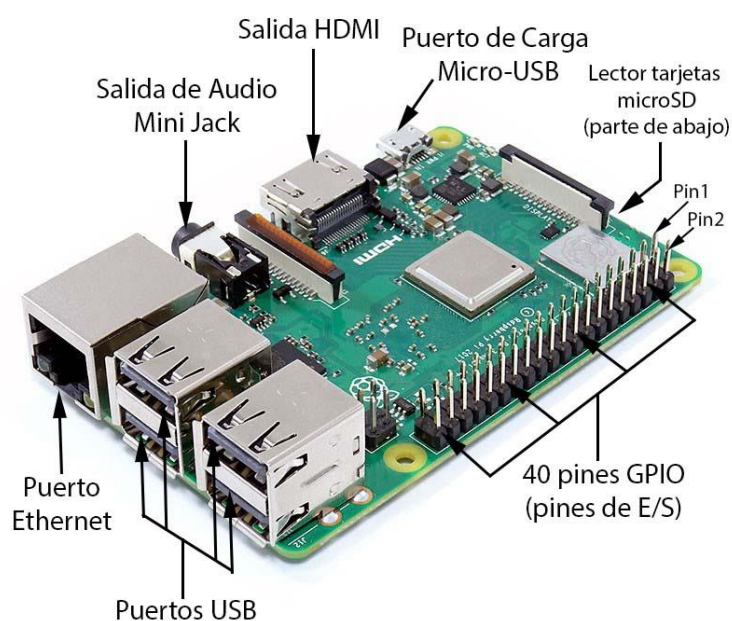


Figura 27. Esquema conexiones Raspberry Pi³¹

La primera vez que se arranca hay que configurar el sistema. Aparece una pantalla de bienvenida y un asistente de configuración inicial. Se hace click en “Next” y aparece una pantalla donde hay que seleccionar país, lenguaje y zona horaria. Se ha seleccionado como país España, como lenguaje español y como zona horaria Madrid.

³¹ Fuente: <https://www.arcadexpress.com/blog/como-conectar-un-mando-arcade-a-la-raspberry-pi/>



Figura 28. Configuración país Raspberry Pi

En la siguiente ventana de configuración pide cambiar la contraseña por seguridad, lo recomendable es cambiarla por una segura y hacer clic en “Next”.



Figura 29. Configuración contraseña Raspberry Pi

El siguiente paso es la configuración de la resolución de pantalla, si está bien no hay que hacer nada en este paso y clicar en “Next”. Si la pantalla tiene un borde negro, hay que marcar la casilla “*This screen shows a black border around the desktop*” para cambiar la resolución. Todos estos ajustes se pueden modificar más adelante también.



Figura 30. Configuración resolución pantalla Raspberry Pi

A continuación, se pedirá conectar a una red WiFi, ya que no está conectado directamente al router por cable Ethernet. Se puede configurar la conexión WiFi en ese momento o dejarlo para más adelante pulsando en "Skip". Se selecciona la red WiFi a la que conectarse, se introduce la contraseña y se hace clic en "Next".



Figura 31. Configuración WiFi Raspberry Pi

Por último, ofrece la posibilidad de buscar e instalar actualizaciones. De nuevo, se puede hacer clic en "Skip" para hacerlo más adelante o "Next" para hacerlo en el momento. Es recomendable actualizar para tener la última versión liberada y evitar problemas de compatibilidad.



Figura 32. Actualización software Raspberry Pi

Antes de pasar a implementar el reconocimiento facial se conecta la webcam al puerto USB de la Raspberry Pi y se hace una prueba para ver si funciona. Tras esto ya estaría todo listo para comenzar.

6.2. Reconocimiento facial

La implementación del reconocimiento facial se ha llevado a cabo en varias partes: lo primero ha sido instalar OpenCV y las librerías necesarias. Una vez completadas las instalaciones, el siguiente paso ha sido probar que la cámara funciona correctamente y probar la detección de caras. Después se ha creado un *dataset* con el usuario a registrar, posteriormente se ha entrenado el modelo para que reconozca a ese usuario y finalmente se ha llegado al reconocimiento facial.

6.2.1. Instalación librerías

Python ya viene preinstalado en Raspberry Pi OS, para comprobar que versión está instalada basta con poner en el terminal la siguiente línea:

```
python3 --version
```

```
log@raspberrypi:~ $ python3 --version
Python 3.9.2
log@raspberrypi:~ $
```

Figura 33. Versión Python

OpenCV está disponible para instalar con *pip*, el administrador de paquetes de Python. Gracias a esto ya no es necesario instalarlo con el código fuente. Para instalarlo hay que ejecutar la siguiente línea de código en el terminal:

```
sudo apt install python3-opencv
```

Para verificar que la instalación se ha realizado correctamente se crea un pequeño programa para comprobar la versión de OpenCV instalada. Se hace en un entorno de desarrollo como es Thonny por ejemplo, que viene preinstalado.

```
import cv2
cv2.__version__
```

Al ejecutar el código debería devolver en la consola la versión instalada.



```
1 import cv2
2 print (cv2.__version__)
3
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
4.5.1
>>>
```

Figura 34. Versión OpenCV

Se van a necesitar otra serie de librerías para el correcto desarrollo del proyecto. Estas son *cmake*, *dlib*, *face recognition*, *numpy*, *pillow* y *imutils*.

Cmake es una herramienta necesaria para que se pueda instalar el módulo de *face recognition*.

Dlib contiene algoritmos de *machine learning* y servirá para la detección facial.

Face recognition es el módulo que ayudará a identificar y reconocer caras en el video.

Numpy se usa normalmente en procesamiento de operaciones matemáticas y lógicas, pero en este caso servirá para ayudar a procesar imágenes.

Pillow agrega la funcionalidad de abrir, modificar y guardar muchos formatos de imágenes directamente desde Python.

Imutils añade funciones para redimensionar más imágenes.

Se van instalando los módulos en el terminal:

```
pip3 install cmake
pip3 install dlib
pip3 install face_recognition
pip3 install numpy
pip3 install pillow
pip3 install imutils
```

6.2.2. Comprobación cámara

Antes de empezar con el reconocimiento facial hay que comprobar que la cámara funciona correctamente y se puede mostrar por pantalla. Se genera un código para mostrar las imágenes que recibe la cámara en la pantalla tanto a color como en blanco y negro llamado “CamTest.py”.

La parte más relevante de este código son las siguientes líneas, que se encargan de abrir la cámara para capturar lo que esté pasando y leerlo.

```
cap = cv2.VideoCapture(0)
(...)
ret, frame = cap.read()
```

6.2.3. Detección facial

Una vez comprobado el correcto funcionamiento de la cámara, se prueba la detección de caras antes de empezar a entrenar el modelo.

Para la detección facial se ha usado el clasificador preentrenado disponible en OpenCV llamado Haars Cascades. Haars Cascades detecta objetos usando el método propuesto por Paul Viola y Michael Jones en el artículo científico “*Rapid Object Detection using a Boosted Cascade of Simple Features*”[26].

Basado en técnicas de Machine Learning, una “función cascada” es entrenada con muchas imágenes positivas y otras negativas. En este caso particular, con muchas imágenes con caras y otras muchas sin ellas. En el directorio de “Haars Cascade” se encuentran muchas de estas imágenes para poder entrenar el modelo.

Se ha escogido este método el lugar de otros porque a pesar de ser algo más lento es más fiable que los demás, y para este proyecto la fiabilidad es muy importante.

Se desarrollarán a continuación las partes más importantes del código de programa "FaceDetection.py"

Con la siguiente línea se carga el clasificador en la carpeta "Cascades" que se ha creado previamente.

```
faceCascade =
cv2.CascadeClassifier('Cascades/haarcascade_frontalface_def
ault.xml')
```

Luego se llama a función detectMultiScale pasándole algunos parámetros:

```
faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.2,
    minNeighbors=5,
    minSize=(20, 20)
)
```

Gray para indicar que es en escala de grises.

ScaleFactor indica el porcentaje al que se va a ir reduciendo la imagen para ir reconociendo rostros de distintos tamaños.

MinNeighbors indica cuantos "vecinos" debe tener un rectángulo para considerarlo cara. Un número muy bajo en este parámetro implicaría falsos positivos y un número demasiado alto podría dar falsos negativos y no reconocer alguna cara real.

MinSize: tamaño mínimo del objeto, objetos más pequeños que el pasado en el parámetro se ignoran.

En el capítulo de pruebas realizadas se ira mostrando qué va sucediendo al variar estos parámetros y se desarrollará el razonamiento de la elección de estos.

Al reconocer una cara, se almacenan los puntos x, y, ancho y alto y se representa el rectángulo con el siguiente bucle:

```
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
```

Se pueden identificar más cosas a parte de la cara, también sonrisa y ojos, pero eso queda de fuera del alcance de este proyecto.

6.2.4. Recolección de datos

En esta parte se recopilan datos de las caras de las personas que se deseen identificar para que la función sea capaz de diferenciar las características que hacen única a esa persona respecto a cualquier otra.

El código se puede encontrar en el anexo 1, pero a continuación se desarrollarán las partes del código más relevantes.

Se añade al código un input para definir qué id se le va a asignar al usuario que va a registrar su cara en el sistema. Este número ha de ser único para cada persona que se vaya a registrar.

```
face_id = input('\n introduzca id de usuario y presione
<return> ==>  ')
```

Cada una de las fotos tomadas se guarda en la carpeta definida y se guardarán con el formato user.id.count.jpg

```
cv2.imwrite("dataset/User." + str(face_id) + '.' +
str(count) + ".jpg", gray[y:y+h,x:x+w])
```

Se ha decidido tomar 30 fotos porque se considera que es un número de muestras suficiente para ser capaz de reconocer a la persona en la inmensa mayoría de los casos, pero mientras más muestras haya mejor. Lo ideal es que la persona vaya cambiando de expresión durante la toma de las muestras para así tener más material y una mejor idea de cómo es la cara de la persona en distintas circunstancias.

6.2.5. Entrenamiento del modelo

Recolectadas las muestras del usuario que se desee registrar, hay que entrenar el modelo para que sea capaz de reconocerlo. Este entrenamiento se lleva a cabo mediante una función de OpenCV y genera un archivo de extensión yml que se guarda en la carpeta “trainer”. Este programa se llama “FaceDataset.py” y el código se puede encontrar en el anexo 2.

Para analizar la imagen se usa la función LBPHFaceRecognizer, que va estudiando la imagen por porciones

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

La función getImagesAndLabels tomará las fotos de la carpeta en la que se encuentran y “entrenará” al modelo para que reconozca esa cara.

Cada vez que se desee introducir un nuevo usuario habrá que ejecutar tanto el código que toma las muestras como este para entrenar nuestro modelo con ese usuario específico y así sea capaz de reconocer su cara.

6.2.6. Reconocimiento facial

Esta parte será la que se use para abrir la puerta o no dependiendo de si se reconoce al usuario. Capturará el vídeo, detectará la cara y comprobará si es conocida. Si lo es, indicará con qué porcentaje de confianza lo es.

El programa se llama “FacialRecognition.py” y todo el código se puede encontrar en el anexo 3.

Se añade una línea para crear un array de nombres que asigna el nombre deseado a cada id. De esta forma, al reconocer a la persona saldrá su nombre en la pantalla en vez del id.

```
names = ['None', 'Loreto'] # Names related to ids
```

La porción analizada de la imagen se pasa a la función recognizer.predict, que toma la porción y la compara con las caras que posee, devolviendo el usuario más probable (si es que lo hay) y la “confianza” en que la cara que se está analizando pertenezca a ese usuario con el que se la ha identificado.

```
id, confidence = recognizer.predict(gray[y:y+h,x:x+w])
```

La función devolverá un cero si se da una coincidencia perfecta, por tanto, mientras mayor sea este número menor será la confianza en que realmente sea ese usuario.

Si hay coincidencia, se coloca el nombre del usuario en la pantalla y se calcula la probabilidad de acierto, que no es más que cien menos el índice de confianza obtenido de la función anterior. Si no hay coincidencia, aparecerá la palabra “desconocido” en la pantalla.

```
if (confidence < 100):
    id = names[id]
    confidence = " {0}%".format(round(100 - confidence))
else:
    id = "unknown"
    confidence = " {0}%".format(round(100 - confidence))
```

De esta forma, se ha creado paso a paso un fichero de Python que es capaz de reconocer a personas previamente entrenado para ello.

6.2.7. Integración botón y encendido LED

Una vez está operativo el reconocimiento facial, se le añade al programa nuevo código para detectar cuando se pulsa el pulsador y también para encender el LED cuando se reconozca al usuario.

El esquema del circuito es el que se muestra a continuación. Consta de una resistencia y un LED para simular la apertura de la puerta y un pulsador para activar el reconocimiento facial. Se ha conectado el pin 15 al LED y el pin 24 al pulsador.

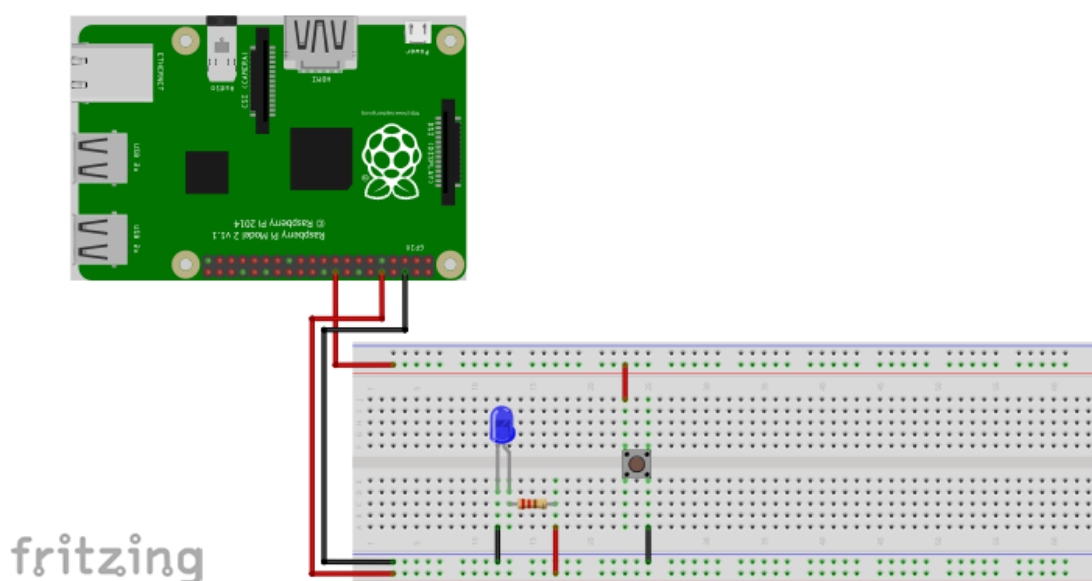


Figura 35. Esquema circuito³²

La apertura de la puerta se simula con un LED que se enciende unos segundos si la cara es reconocida, y si no se ha reconocido a nadie en unos segundos se sale del reconocimiento (posteriormente se incorporará la funcionalidad de la notificación al dispositivo).

El código del programa apertura.py se encuentra en el anexo 4 para su consulta. Este programa es llamado si se reconoce a la persona con una confianza mayor al 70%, Para ello, se introducen algunos cambios en el programa de reconocimiento facial para adaptarlo.

Se comienza inicializando la variable “confidence” con un valor de 200 (cero sería una coincidencia perfecta y por debajo de 100 se considera cierta coincidencia). También se registra la hora actual y se crea una nueva variable con el tiempo límite.

```
confidence =200
(...)
```

³² Diseñado con Fritzing: <https://fritzing.org/>

```
ini_time=time.time()
end_time=ini_time + 5
```

Se crea una nueva condición donde si la confianza es mayor al 70%, se lanza el programa de apertura.py. Sin embargo, si pasa el tiempo sin reconocer a nadie sale del programa.

```
if (confidence < 30):
    abrir(id) # Open the door (turn on LED)
    break
else:
    if (time.time() > end_time):
        break
```

Para el pulsador se ha creado un código bastante sencillo llamado timbre.py que está pendiente constantemente del pulsador y ejecuta el programa de reconocimiento facial cuando detecta que se ha pulsado. El código se puede encontrar en el anexo 5 para su consulta.

Es necesario que este programa empiece a ejecutarse automáticamente cuando arranca la Raspberry Pi se ha usado crontab. Cron es un servicio que se inicia cuando arranca la Raspberry Pi y permite ejecutar código programado por el usuario. Se puede hacer que ejecute un código cuando arranque el sistema, o bien se puede configurar para que se ejecute diariamente, semanalmente... Tiene muchas opciones.

Antes de configurarlo, se debe dar al programa que se va a ejecutar permiso para que crontab lo ejecute. Se hace mediante la siguiente línea de código estando en el terminal.

```
chmod +x /home/log/FacialRecognitionProject/timbre.py
```

Para configurarlo se escribe en el terminal el siguiente comando y si es la primera vez que se ejecuta habrá que pulsar la tecla Enter de nuevo para usar el editor predeterminado.

```
sudo crontab -e
```

Dentro del fichero, y al final de este se usa el código que viene a continuación. El comando @reboot es lo que hace que se ejecute cuando arranca el sistema. Además, se han creado dos nuevos archivos: stdout.txt para las salidas del programa y stderr.txt como log de errores para facilitar solucionar posibles errores.

```
@reboot python3 / home / log / FacialRecognitionProyect /
timbre.py > / home / log / stdout.txt 2 > / home / log /
stderr.txt
```

Tras guardar y cerrar hay que reiniciar la Raspberry para que los cambios tengan efecto.

6.3. Desarrollo web y base de datos

6.3.1. Configuración IP fija

Antes de entrar en desarrollo web es necesario asignar una IP³³ fija a la Raspberry Pi para asegurar que siempre se podrá encontrar y sea accesible.

Para asignar una IP hay dos formas, se puede hacer desde el router configurándolo para enlazar la dirección MAC de la Raspberry Pi con una dirección IPv4 o desde la propia Raspberry Pi.

Se va a realizar desde la Raspberry Pi, con un servicio³⁴ que se usa como cliente DHCP³⁵.

Para hacer esto posible hay que verificar que DHCPDC está activado poniendo el siguiente comando en el terminal.

```
sudo service dhcpd status
```

Si no está activo, se activa de esta forma:

```
sudo service dhcpd start
sudo systemctl enable dhcpd
```

El paso previo antes de cambiar la IP es ver que IP, Gateway y dirección del servidor DNS se tiene actualmente y con esos datos se puede crear una estática. Para verlo se ejecuta el siguiente comando en el terminal:

```
ifconfig
```

El cambio de IP se gestiona desde el terminal y consisten en modificar un archivo para asignar una dirección IP estática:

```
sudo nano /etc/dhcpd.conf
```

En ese archivo se configura la IP deseada. Hay que asignar una que esté en rango, pero con números altos.

```
interface wlan0
static ip_address=192.168.1.100/24
static routers=192.168.1.1
static domain_name_servers=212.142.173.65 77.26.11.233 192.168.1.1
```

Figure 36. Captura configuración IP fija

³³ https://es.wikipedia.org/wiki/Direcci%C3%B3n_IP

³⁴ [https://es.wikipedia.org/wiki/Daemon_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Daemon_(inform%C3%A1tica))

³⁵ <https://www.redeszone.net/tutoriales/internet/que-es-protocolo-dhcp/>

El último paso es reiniciar y ver que realmente la IP está fija en los valores que se han asignado.

6.3.2. Instalación MariaDB

Django viene por defecto con SQLite como base de datos por defecto. Para mejorar el rendimiento, se va a cambiar por MariaDB.

Para instalar MariaDB solo es necesario ejecutar la siguiente línea en el terminal (con `-y` al final para que haga la instalación por defecto):

```
sudo apt-get install mariadb-server mariadb-client -y
```

Una vez instalada, se configura con la siguiente línea de código, donde pedirá una contraseña para el usuario.

```
sudo mysql_secure_installation
```

Tras esto, ya estaría configurada MariaDB y se podrían crear bases de datos, tablas y hacer consultas y modificaciones desde el terminal.

6.3.3. Instalación y preparación del entorno de desarrollo

Como el alcance del proyecto es crear un solo entorno web no es necesario el uso de entornos virtuales. Sin embargo, si se están llevando a cabo distintos proyectos de Django en el mismo dispositivo sí sería útil la instalación de un entorno virtual por si se necesitan usar diferentes versiones de Django y de los paquetes instalados dependiendo de la necesidad.

Se instala Django ejecutando la siguiente línea de código en el terminal:

```
pip3 install django
```

Una vez instalado, se comprueba que está correctamente instalado mostrando la versión con el siguiente comando, que nos debe devolver la versión instalada actualmente de Django.

```
python -m django --version
```

El siguiente paso será el esqueleto del proyecto. Para ello se crea una carpeta donde irán almacenados los proyectos, en este caso la carpeta creada se ha llamado "djangos". Dentro de esta carpeta, se crea una nueva carpeta que contendrá el proyecto de la página web, que se ha llamado se ha llamado "website".

Para comprobar que todo funciona correctamente, se ejecuta el siguiente comando en el terminal para arrancar el servidor:

```
python manage.py runserver
```

Con el servidor corriendo, se accede a la siguiente URL <http://127.0.0.1:8000/> desde el navegador. Si la instalación de Django ha sido exitosa aparecerá una página como esta:

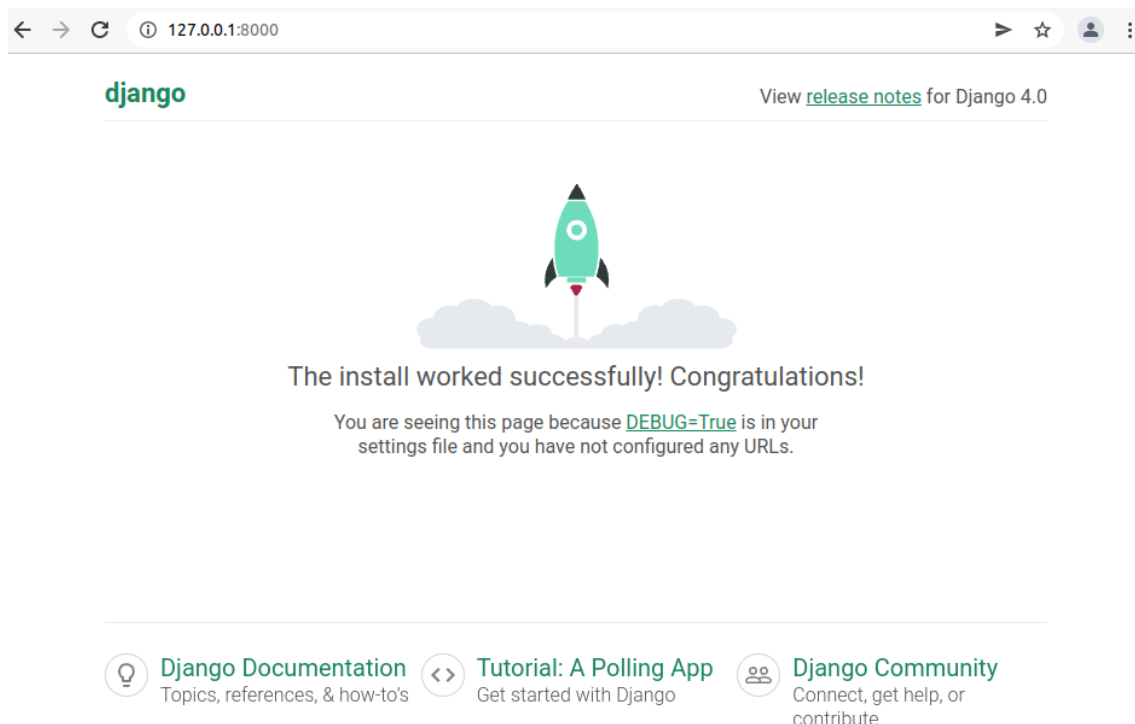


Figura 37. Instalación Django

Una vez comprobado que la instalación se ha hecho correctamente, se procede a crear el nuevo proyecto. En el terminal, se navega dentro de la carpeta “website” y se crea el nuevo proyecto usando el comando “startproject”:

```
cd django
cd website
django-admin startproject website
```

Al ejecutar este código se van a crear una serie de subcarpetas y ficheros. Ahora mismo la estructura del proyecto sería la siguiente:

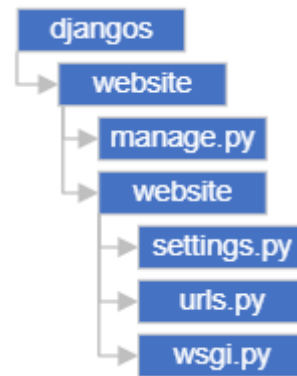


Figura 38. Estructura carpeta "website"

El programa `manage.py` se usa para crear aplicaciones, trabajar con la base de datos y el servidor.

En `settings.py` se encuentra toda la configuración del sitio. Localización de archivos estáticos, base de datos, zona horaria...

El fichero `Urls.py` contiene los "mapeos" (correspondencias) url-vistas. Se suele configurar para enlazar el *mapeo* a las aplicaciones en vez de tenerlo todo aquí.

`Wsgi.py` ayuda en la comunicación con el servidor web.

Completada la estructura del proyecto, se creará una aplicación para la página web llamada "catalog". Para ello se usa el comando "startapp" en el terminal bajo la carpeta "website".

```
python manage.py startapp catalog
```

Como consecuencia, se crea una nueva subcarpeta y ficheros para conformar la aplicación. La nueva distribución quedaría de esta forma:

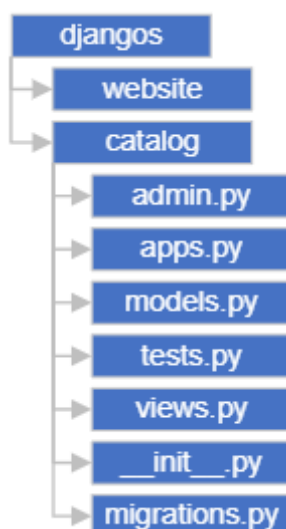


Figura 39. Estructura carpeta "catalog"

El fichero "admin.py" contiene la configuración del sitio web y es donde se registran los modelos creados.

En "apps.py" se registran las aplicaciones y se configuran sus atributos.

"models.py" es uno de los ficheros más importantes, ya que es donde se crean los modelos. Aquí se registran las distintas tablas y columnas de la base de datos.

"tests.py" es para las pruebas, pero no se le dará uso en este proyecto.

El fichero "views.py" también es muy importante ya que contiene las vistas, y es aquí donde se le da funcionalidad a la web (muestra video, enciende LED...).

"__init__.py" no es más que un fichero vacío para que se reconozca la carpeta como un paquete Python.

Por último, el fichero "migrations.py" guarda información sobre las migraciones. Es lo que permite hacer efectivo los cambios y modificaciones efectuados en la base de datos.

6.3.4. Configuración del entorno

A continuación, se llevarán a cabo una serie de modificaciones a los distintos ficheros para adaptarlos a las necesidades del proyecto.

En el fichero "settings.py" habrá que hacer varios cambios:

- En "installed_apps" hay que añadir la aplicación "catalog", el paquete "widget_tweaks" para ayudar a ajustar las propiedades del formulario, y "django_bootstrap5" para hacer más fácil la creación de interfaces:


```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'catalog.apps.CatalogConfig',
    'widget_tweaks',
    'django_bootstrap5',
]

```

- Cambiar la base de datos por MariaDB (comparte *backend* con MySQL):

```

# Change sqlite3 for MariaDB
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'FacialRecognition',
        'USER': 'phpmyadmin',
        'PASSWORD': '1234',
        'HOST': 'localhost',
        'PORT': '3306',
        'OPTIONS': {
            'init_command': "SET
sql_mode='STRICT_TRANS_TABLES'"
        }
    }
}

```

- Cambio zona horaria:

```

TIME_ZONE = 'Europe/Madrid' # Changed to Spanish timezone

```

- Debug: se deja habilitado mientras se hacen las pruebas, una vez terminadas se quita. Esto es muy importante por razones de seguridad.

```

# SECURITY WARNING: don't run with debug turned on in
production!
DEBUG = True

```

La aplicación “catalog” viene con un fichero `urls.py`, y desde aquí se manejarán las url-vistas. Es necesario mapear este fichero en el fichero `urls.py` de la carpeta del proyecto. Además, hay que importar “include” para poder mapearlo a la aplicación:

```

from django.urls import include

urlpatterns += [

```

```
    path('catalog/', include('catalog.urls')),
]
```

Así todas las peticiones que vengan con el patron 'catalog/' se gestionaran desde el fichero urls.py de la aplicación.

Ahora se redirigirá la URL del sitio, que es la `http://127.0.0.1:8000/`, a `http://127.0.0.1:8000/catalog/` ya que esta será la única aplicación que se usa en este proyecto:

```
from django.views.generic import RedirectView

urlpatterns += [
    path('', RedirectView.as_view(url='/catalog/',
    permanent=True)),
]
```

Además, se va a habilitar también el servicio de archivos estáticos como CSS y JavaScript con el siguiente código:

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns += static(settings.STATIC_URL,
    document_root=settings.STATIC_ROOT)
```

Teniendo configurado el fichero urls.py, se puede crear dentro de la aplicación un fichero urls.py. En él vamos a plantear la estructura que tendrá, que será la siguiente:

```
from django.urls import path
from . import views

urlpatterns = [

]
```

Aquí se irán añadiendo las URL-vistas necesarias.

Con esto ya estaría listo el esqueleto del proyecto. Llegados a este punto, se prueba a ejecutar nuestro sitio web. No va a funcionar nada puesto que aún no hay nada creado, pero merece la pena para comprobar que todo va bien y no se ha desconfigurado nada al hacer las modificaciones.

Antes de ejecutarlo, se deben crear las migraciones. Se ejecuta el comando "makemigrations", que es el encargado de crear las migraciones necesarias, pero no las ejecuta. Esto da la posibilidad de comprobar y modificar el código de las migraciones antes de que se apliquen (para usuarios con un conocimiento muy avanzado).

```
python manage.py makemigrations
```

Una vez ejecutado el comando `makemigrations`, ejecutamos el comando “`migrate`”, que aplica la migración creada a la base de datos para incluir los cambios que se hayan ido haciendo durante las pruebas y desarrollo.

```
python manage.py migrate
```

Aplicadas las migraciones, ya se puede ejecutar el servidor

```
python manage.py runserver
```

De nuevo, en se navega a la URL <http://127.0.0.1:8000/> para cargar la página.

Se puede comprobar que devuelve una página de error, lo que es perfectamente normal porque no existe ninguna página definida en el módulo `catalogs.url`. Además, se puede observar que la página que devuelve es <http://127.0.0.1:8000/catalog/>, es decir, la redirección hacia “`catalog`” que se ha aplicado funciona correctamente.

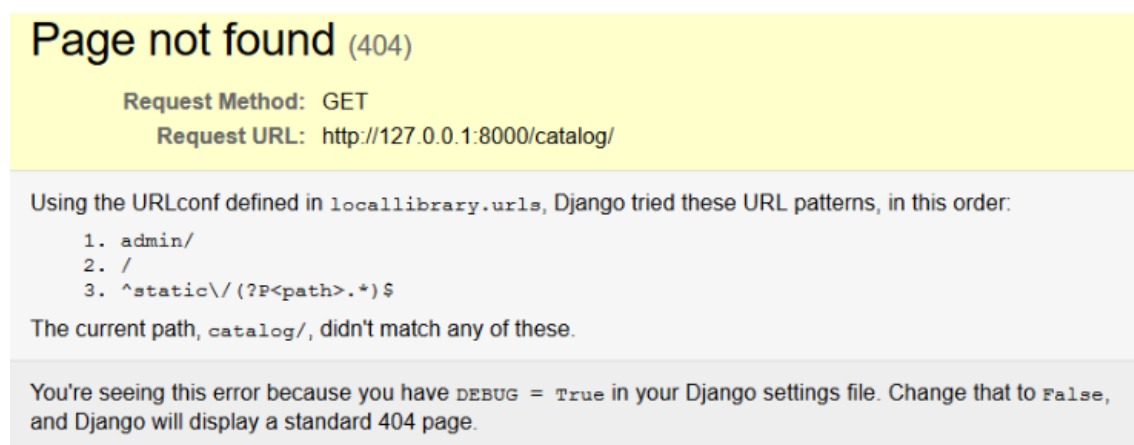


Figura 40. Fallo al cargar página

6.3.5. Desarrollo del entorno web

Teniendo el entorno y el esqueleto del proyecto configurado y listo, se procede a crear el modelo en “`models.py`”, las vistas en “`views.py`” incluir las URL necesarias en “`urls.py`” y crear la página web.

Lo primero será crear el modelo en “`models.py`”. En el modelo, llamado “`doorRelease`”, se va a crear un campo correspondiente con cada columna de la tabla en la base de datos, así se podrá interactuar con ella, leer y escribir en esas columnas.

“`reccdate`” indicará la fecha y hora a la que se ha llevado a cabo el reconocimiento.

“`userid`” será el usuario que se ha reconocido, si es que se ha reconocido alguno.

“success” es booleano, y su valor será uno o cero dependiendo de si el reconocimiento ha sido exitoso.

```
from django.utils import timezone

class doorRelease(models.Model):
    recdate = models.DateTimeField(null=True,
    verbose_name="date")
    userid = models.CharField(null=True, blank=True,
    max_length=20, verbose_name="user")
    success = models.BooleanField(null=True, blank=True,
    verbose_name="success")
```

Con esto ya estaría programado todo lo referente al modelo.

A continuación, se va a estructurar la funcionalidad de la página web, para así poder definir las vistas y URLs necesarias.

La página web constará de una sola página con las siguientes características: aparecerá una tabla con los últimos registros en la base de datos, aparecerá el vídeo de lo que ve la cámara en ese momento y tendrá un botón de apertura que abrirá la puerta (encenderá el LED).

Para llevar esto a cabo se necesitarán tres vistas, aunque solo una de ellas devolverá una URL, que será la de la página web. Esta URL será la que cargue la tabla de últimas entradas. Habrá otra vista encargada de recuperar el vídeo en tiempo real, y otra más para gestionar la pulsación del botón de apertura.

Es decir, se necesitan tres vistas en “views.py” y tres URL en “urls.py”.

El código completo de “urls.py” se puede encontrar en el anexo 6.

Se va a añadir en “urls.py” las tres URL necesarias, cada una de las cuales llamará a una vista:

```
urlpatterns = [
    path('', views.index, name='index'),
    path('camera/', views.livefe, name="live_camera"),
    path('apertura/', views.abrir, name="apertura"),
]
```

En “views.py” se crean las vistas para que la página web funcione de la forma deseada. Se importarán primero todos los módulos y funcionalidades necesarias y se irán creando las distintas vistas. El código completo del fichero “views.py” se encuentra en el anexo 7, aquí se explicará cada vista y su funcionalidad.

La primera vista es la encargada del vídeo, crea un objeto de una clase que se ha creado previamente encargada de hacer posible que se vea el vídeo y devuelve el mismo a la página web.

```
@gzip.gzip_page
def livefe(request):
    try:
        cam=VideoCamera()
        return StreamingHttpResponse(gen(cam),
content_type="multipart/x-mixed-replace;boundary=frame")
    except:
        pass
```

La segunda vista ejecuta el programa “apertura.py” que encenderá el LED cuando se pulsa el botón.

```
def apertura(request):"    if request.method=='POST':
    apertura.abrir()
    response_data='successful'
    return JsonResponse(response_data)
else:
    return HttpResponse(json.dumps({"nothing to see":
"this is not happening"}), content_type="application/json")
```

Esta parte tiene algo particular y es que no recarga la página cuando devuelve la respuesta a la página web, optimizando el funcionamiento de esta. Esto se ha conseguido con JavaScript y Ajax, y se explicará más adelante cuando se hable sobre el código de página web.

La tercera vista devuelve la página de inicio y muestra tabla de entradas recientes obteniéndola a través del modelo.

```
def index(request,*args,**kwargs):
    datalist = doorRelease.objects.all()
    return render(request,'index.html',context=
{'datalist':datalist})
```

Ya solo faltaría la plantilla. La plantilla no es más que un fichero con estructura HTML que se creará en la carpeta “template”, ya que Django la buscará allí.

Como la página web va a tener una estructura definida y solo cambiará el cuerpo, se va a usar una plantilla base que luego se importa en las diferentes plantillas para así no tener que duplicar el código cada vez y hacer más sencillas futuras modificaciones. En este caso solo se usa una plantilla ya que solo hay una página web y no hay redireccionamientos, pero se ha hecho de esta forma por si en el futuro se deseara dar más funcionalidad a la página web.

Se crea la carpeta “templates” dentro de “catalog”. Dentro de “templates” se crea el archivo “base_generic.html” y se le da forma. El código se puede encontrar en el anexo 8, aquí se comentará solo lo más relevante:

Se añade la funcionalidad de Bootstrap para facilitar la correcta visualización de la página web independientemente del dispositivo en el que se esté viendo (pc, table, teléfono móvil...)

```
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
```

Se añade jQuery para poder usar JavaScript y Ajax y así evitar que se recargue la página bajo determinadas acciones. Se puede hacer que no se recargue o que se recargue solo una parte.

```
src="https://code.jquery.com/jquery-3.1.1.js"
```

Por último, se carga el contenido de un archivo estático que se ha creado previamente y aporta el componente de CSS para hacer más atractiva la página.

```
src="{% static 'css/styles.css' %}"
```

Para crear componentes estáticos, se crea la carpeta "static" en "catalog". Dentro de ella se crea otra carpeta, "css" en este caso, y dentro esta nueva carpeta se crea el fichero "styles.css" que es el que se ha importado en "base_generic.html".

Ya solo faltaría elaborar el fichero "index.html" dentro de la carpeta "templates".

Una vez creados todos los ficheros y carpetas, la estructura de la carpeta "catalog" debería ser la siguiente.

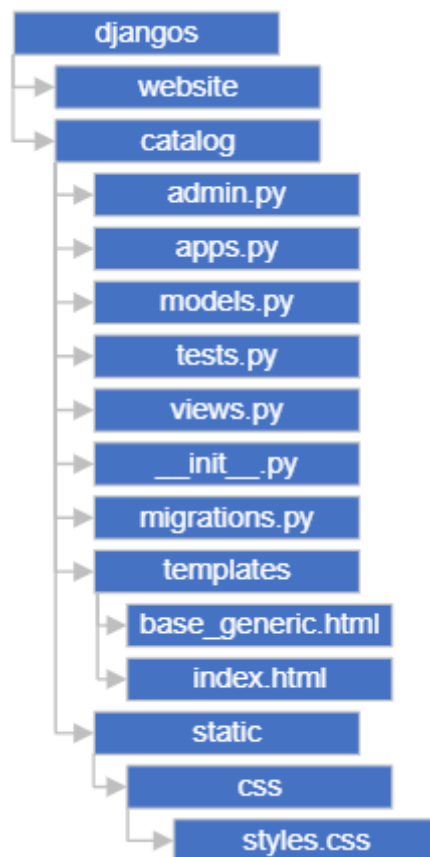


Figura 41. Estructura final carpeta “catalog”

El código de la plantilla “index.html” está en el anexo 9, pero se van a explicar las características más importantes del código.

Se crea un formulario con un botón para que cuando se pulse se llame a la vista que ejecuta el programa de encender el LED.

```

<form method="POST" id="post-form">
  {% csrf_token %}
  <input type="submit" value="Abrir puerta"></button>
</form>

```

Como se indicó anteriormente, esta funcionalidad tiene la particularidad de no actualizar la página cuando se llama a la vista. Esto se consigue con el siguiente programa, que intercepta la petición y es el quién lo manda a la vista.

Con la línea “event.PreventDefault();” es como se evita que se recargue la página y Ajax es el encargado de mandar la petición.

```

<programa>
$(document).ready(function() {
  $('#post-form').on('submit', function(event) {
    event.preventDefault();
    $.ajax({

```

```

        url: "{% url 'apertura' %}",
        type:"POST",
        data:{

csrfmiddlewaretoken:${('input[name=csrfmiddlewaretoken]').va
l()},
        },
        success:function(){
        },
    });
    });
    });
    });
</programa>

```

Para recuperar el video se usa la siguiente línea de código

```

```

Se puede observar que se está buscando la URL “live_camera”, que aparece en el archivo “urls.py” y es donde se llama a “views.livefe”.

```
path('camera/', views.livefe, name="live_camera"),
```

En “views.py” se observa como la vista para recuperar video se llama “livefe”.

```
def livefe(request):
```

Para la tabla de últimos registros se crea una tabla y con un bucle se van recorriendo todos los datos.

```

{% for datalist in datalist %}
  <tr>
    <th scope="row">{{ datalist.id }}</th>
    <td>{{ datalist.userid }}</td>
    <td>{{ datalist.recddate }}</td>
    <td>{{ datalist.success }}</td>
  </tr>
{% endfor %}

```

Se busca en la variable “datalist”. Cuando se carga la página, se llama a la vista “index”

```
path('', views.index, name='index'),
```

Dentro de la vista “index”, se encuentra el *mapeo* de los distintos valores de las columnas de la base de datos, y devuelve a la web estos valores para que se muestren.


```
datalist = doorRelease.objects.all()
return render(request, 'index.html', context=
{'datalist':datalist})
```

Completados todos los pasos, no quedaría más que ejecutar el comando “makemigrations” y “migrate” de la forma que se ha visto antes y ejecutar el servidor con el comando “runserver”.

Al navegar a <http://127.0.0.1:8000/>, cargará la página web creada y estarán disponibles las funcionalidades.

Para poder ver la página web desde otros dispositivos conectados a la misma red, quedaría modificar el fichero “settings.py” para añadir la IP permitida.

```
ALLOWED_HOSTS = ['192.168.1.100']
```

Cuando se ejecute el comando “runserver” es necesario añadirle “0.0.0.0:8000” para que se pueda ver la web desde cualquier dispositivo conectado a la red.

```
python manage.py runserver 0.0.0.0:8000
```

6.3.6. Registro de entradas

Teniendo la base de datos operativa y la página funcionando ha llegado el momento de comenzar a mandar entradas a la tabla desde el reconocimiento facial. De nuevo, se va a modificar el código de FaceRecognition.py para adaptarlo a esta nueva necesidad.

En el mismo condicional que se añadió anteriormente para ejecutar el programa de apertura se añadirá el código necesario. Se crea la variable “percent” para obtener el porcentaje de confianza, si se ha abierto la puerta automáticamente la variable “success” estará a uno, y si no a cero. Se llama “passdata”, la función que se encarga de pasar los parámetros pasándole como argumento si ha sido exitoso el reconocimiento y el id reconocido (si no se ha reconocido a ningún usuario el id será “Unknow”).

```
if (confidence < 50):
    abrir(id) # Open the door (turn on LED)
    percent = 100 - confidence
    success = 1
    passdata(success, id)
    break
else:
    if (time.time() > end_time):
        percent = 100 - confidence
        success = 0
        passdata(success, id)
        break
```

La función “passdata” se encuentra en el fichero “pymdb.py”. El código de este fichero se encuentra en el anexo 10, pero se analizarán a continuación las partes más importantes.

Primero conecta con la base de datos, si hubiese algún problema lanzaría un error.

```
conn=mariadb.connect(
    user="phpmyadmin",
    password="1234",
    host="localhost",
    port=3306,
    database="FacialRecognition"
)
```

Se crea un cursor, que es lo que permite ejecutar los comandos en la base de datos.

```
cur = conn.cursor()
```

Se procede a escribir. Se usa NOW() para obtener la fecha y hora del registro, y los parámetros “result” y “userid” son los que vienen como argumento desde el programa de reconocimiento facial.

```
sql = "INSERT INTO
catalog_doorrelease(recdate,success,userid)
VALUES (NOW(),%s,%s) "
val = (result,userid)
cur.execute(sql,val)
```

La siguiente línea de código es muy importante, ya que la actualización de la tabla de forma automática no viene por defecto, hay que hacerlo mediante este comando. Seguidamente se cierra el puntero y estaría completo.

```
conn.commit()
conn.close()
```

6.3.7. Envío de notificación a Telegram

Ya está funcionando por un lado el reconocimiento facial cada vez que se pulsa el pulsador y el envío de registros a la base de datos desde el reconocimiento facial, y por otro lado la página web donde se muestra el vídeo, esos registros y el botón de apertura. Lo único que falta es una forma de alertar cuando falla el reconocimiento para que se pueda gestionar la apertura o no de la puerta desde la página web.

Se ha escogido Telegram³⁶ por ser una aplicación de mensajería de uso muy extendido. Ha sido necesario crear un bot³⁷, que será la conversación donde lleguen los mensajes, y estos se lanzan desde Python.

Un bot es una herramienta automatizada para ejecutar tareas específicas. Telegram permite integrarlos en las conversaciones para aportar funcionalidades. Los bots son muy populares en Telegram, desde los que ayudan con el seguimiento de los paquetes de mensajería simplemente pasando los números de seguimiento como @MyTracking³⁸ hasta @pdfbot³⁹, que permite pequeñas modificaciones de archivos PDF.

Telegram está disponible para todas las plataformas (iOS, Android, Microsoft, Linux...). Lo ideal es instalarlo en un dispositivo móvil así llegan las notificaciones directamente allí, pero también se puede usar Telegram Web desde el navegador.

En el dispositivo móvil, en la aplicación Play Store (si es Android, en AppStore si es iOS) se puede encontrar la aplicación Telegram, para ello solo habrá que buscarla en el buscador de la parte superior y darle al botón de instalar.

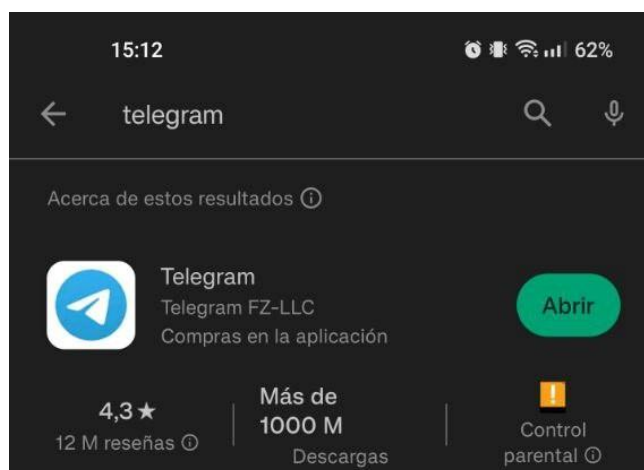


Figura 42. Captura de pantalla Play Store

Abrir la aplicación una vez instalada y seguir los pasos (país y número de teléfono). Se enviará un SMS con un código al número de teléfono introducido y esa será la clave para iniciar sesión. Una vez iniciada se podrá usar Telegram.

El siguiente paso es crear el bot. Esto es muy sencillo, ya que existen en Telegram bots encargados de crear otros bots. En la aplicación de Telegram, en el buscador de la parte superior, se escribe @BotFather. Se selecciona BotFather, el que tiene el tic azul de verificación.

³⁶ <https://es.wikipedia.org/wiki/Telegram>

³⁷ <https://es.wikipedia.org/wiki/Bot>

³⁸ <https://telegram.me/MiTrackingBot>

³⁹ <https://telegram.me/pdfbot>



Figura 43. Captura pantalla del buscador de Telegram

Una vez dentro de la conversación, se usa el comando `/newbot`, a lo que te contesta preguntando cual será el nombre del bot. El nombre escogido ha sido "Videoportero", y tras esto pide un usuario, "VideoporteroUS_bot".

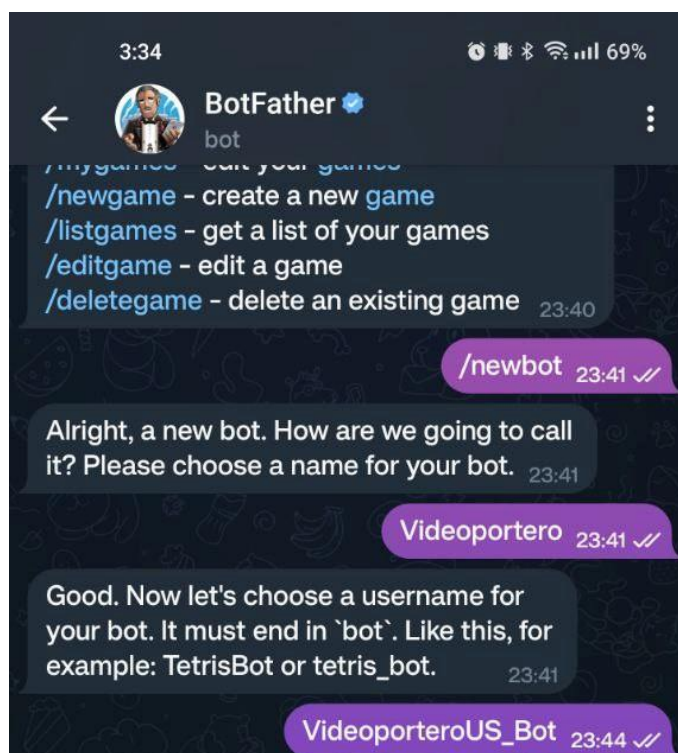


Figura 44. Creación de un bot de Telegram

El bot indicará que se ha creado el nuevo bot de forma exitosa, pasará el enlace para la conversación con él e indicará un token, que será la "api_key" necesaria para el envío del mensaje desde Python.

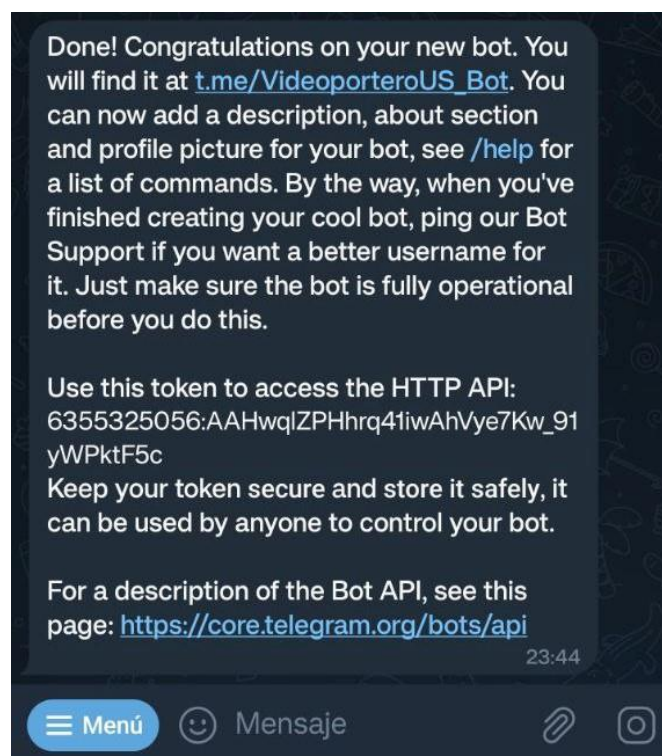


Figura 45. Captura pantalla BotFather

Será necesario también encontrar el id del chat, que se puede obtener mediante otro bot. De nuevo en la aplicación de Telegram, se busca @IDBot⁴⁰ en el buscador superior y se inicia conversación. Aparecerá un mensaje de bienvenida donde explica cómo se usa el bot y qué comandos acepta.

Se manda el comando `/getid` y el bot contestará con un ID numérico que también será necesario para el correcto envío de la notificación.

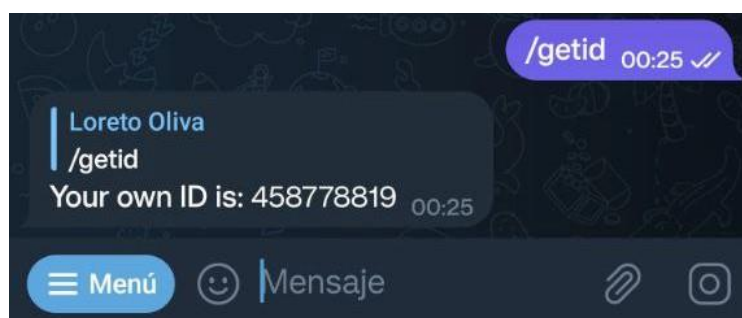


Figura 46. Captura de pantalla IDBot

El envío de la notificación será gestionado por una función llamada `send_telegram_message`, perteneciente al fichero `telegram.py`. El código se encuentra en el anexo 11.

⁴⁰ <https://telegram.me/myidbot>

En este código se envía el mensaje deseado mediante una petición a la API⁴¹ de Telegram pasándole como parámetro la “api_key” y la id del chat.

```
import requests
import json
(...)
data_dict = {'chat_id': chat_id, 'text': message,
'parse_mode': 'HTML', 'disable_notification': True}
data = json.dumps(data_dict)
url = f'https://api.telegram.org/bot{api_key}/sendMessage'
response = requests.post(url, data=data, headers=headers,
proxies=proxies, verify=False)
```

Una vez más, se procede a modificar el programa de reconocimiento facial para añadirle la nueva funcionalidad. En la zona del código que se ejecuta cuando ha pasado el tiempo estipulado y no se ha reconocido a ningún usuario con la confianza suficiente se le añade el siguiente código para mandar la notificación. La variable “chat_id” es el ID obtenido de @MyIDBot, y la variable “api_key” fue proporcionada por @BotFather al crear el bot.

```
if (time.time() > end_time):
    percent = 100 - confidence
    success = 0
    passdata(success, id)
    chat_id = "458778819"
    api_key =
"6355325056:AAHwqlZPHhrq41iwAhVye7Kw_91yWPktF5c"
    send_telegram_message("Hay alguien en la puerta!",
chat_id, api_key)

send_telegram_message("http://192.168.1.100:8000/catalog/",
chat_id, api_key)
    break
```

⁴¹ <https://core.telegram.org/>

7. PRUEBAS REALIZADAS

Se van a mostrar fotos y capturas de las pruebas realizadas, tanto del reconocimiento facial, como de la notificación al dispositivo y la página web.

En cuanto al reconocimiento facial, se han tenido que hacer una serie de pruebas para parametrizar las funciones de una forma óptima. Para ello, existen los siguientes parámetros que se le pasan a la función `detectMultiScale`, explicados en el capítulo 6, apartado 6.2.3:

```
faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.2,
    minNeighbors=5,
    minSize=(20, 20)
)
```

`ScaleFactor`: con un número muy grande se pueden perder detecciones, y un número muy pequeño implica mayor tiempo de procesamiento y posibilidad de falsos positivos. Se ha establecido en 1.2, que implica un 20% de reducción cada vez.

En la siguiente captura se puede ver como con un `ScaleFactor` alto no detecta la cara:

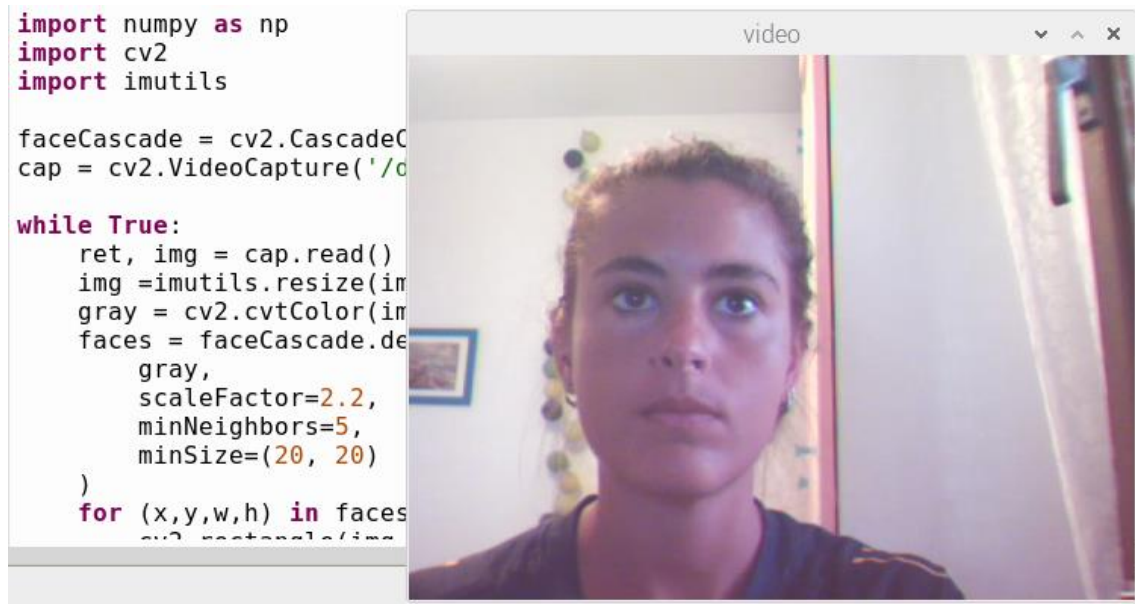


Figura 47. Captura `ScaleFactor` alto

Sin embargo, si se reduce a valor normal sí la detecta:

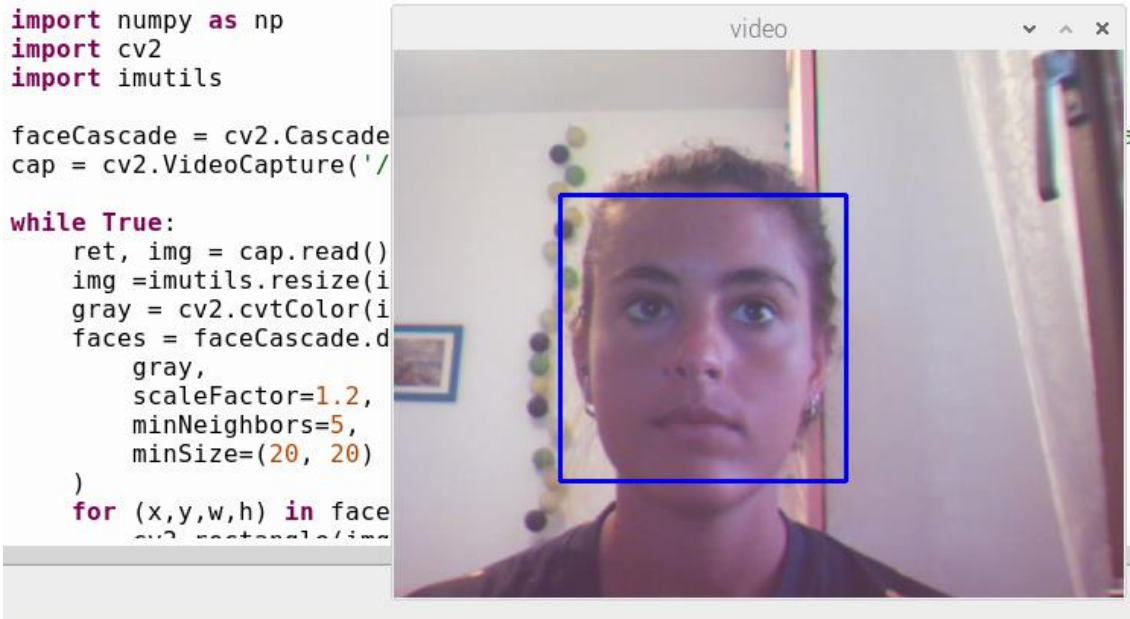


Figura 48. ScaleFactor normal

MinNeighbors: un número muy bajo en este parámetro implicaría falsos positivos y un número demasiado alto podría dar falsos negativos y no reconocer alguna cara real. Se ha establecido un número de 5 rectángulos.

Con un valor muy alto no detecta la cara:

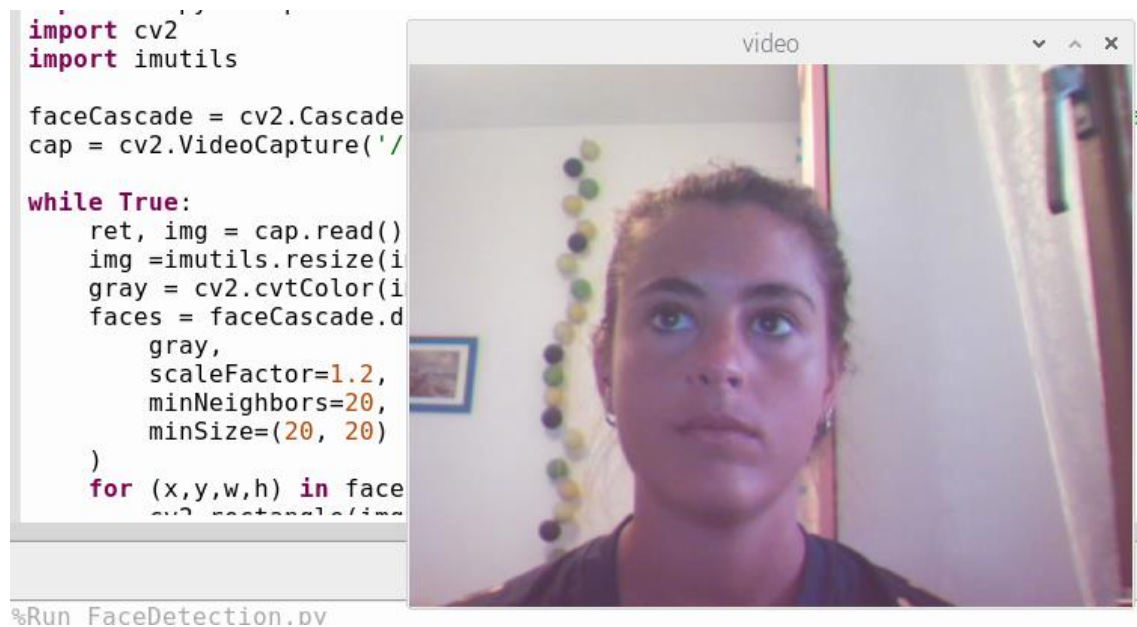


Figura 49. MinNeighbors alto

Con un valor muy bajo detecta varias caras:

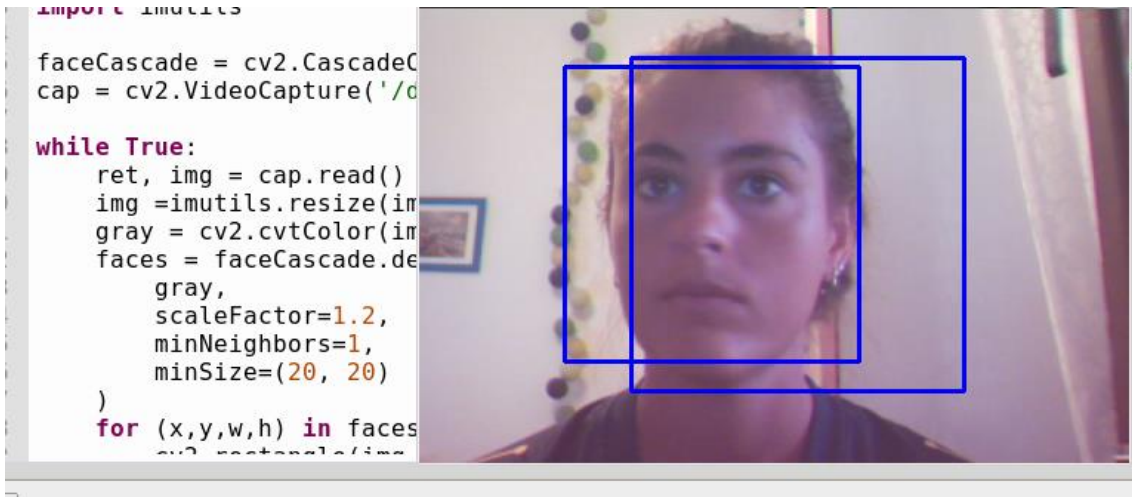


Figura 50. MinNeighbors bajo

MinSize: un número muy pequeño hace que caras más pequeñas puedan ser ignoradas. Teniendo en cuenta el uso que se le va a dar, personas que estarán relativamente cerca de la cámara y esta enfocará a la cara, se ha establecido como (20,20).

En cuanto al porcentaje de confianza para abrir la puerta, se ha llegado a la conclusión de que un 70% es suficiente para reconocer a la persona, dando así un margen para cuando las condiciones de luz no son las óptimas. Al ser un porcentaje alto, la probabilidad de que se confunda a una persona extraña con un usuario registrado es mínima.

Para hacer las pruebas se ha incluido algo de información para mostrar en pantalla e ir verificando que todo funciona correctamente.

Se muestra por pantalla el contenido de la tabla de la base de datos, información sobre si se da por válido el reconocimiento o no (es válido cuando el usuario es reconocido al menos el 70% de confianza) y se muestra la confianza.

```

Parametros: (41, datetime.datetime(2023, 7, 10, 16, 20, 14), 0, 'Loreto')
Parametros: (42, datetime.datetime(2023, 7, 10, 16, 31, 55), 0, 'desconocido')
Parametros: (43, datetime.datetime(2023, 7, 10, 16, 36, 52), 1, 'Loreto')
Parametros: (44, datetime.datetime(2023, 7, 10, 16, 38, 2), 1, 'Loreto')

```

```

[INFO] Reconocido
71.16542878768736

```

Lo que se muestra arriba es una muestra exitosa con las últimas 4 entradas de la base de datos.

EL primer valor de “Parámetros” corresponde al ID, es automático y se va incrementando con cada entrada.

El segundo valor corresponde a la fecha y hora y va separado por comas.

El tercero indica si el reconocimiento ha sido exitoso o no. Se puede ver que cuando el usuario es “desconocido” este parámetro siempre es cero, pero cuando el usuario es “Loreto” a veces es cero y otras uno. Las veces que es cero es debido a que no se ha llegado a ese 70% de confianza y no se ha lanzado el programa de apertura directamente.

El último parámetro corresponde con el usuario.

Se muestra un ejemplo de reconocimiento fallido. Se puede observar que el vez de “Reconocido” aparece “Tiempo” para indicar que ha salido por tiempo y no por reconocimiento.

```
Parametros: (41, datetime.datetime(2023, 7, 10, 16, 20, 14), 0, 'Loreto')
Parametros: (42, datetime.datetime(2023, 7, 10, 16, 31, 55), 0, 'desconocido')
```

```
[INFO] Tiempo
31.16542878768736
```

```
[INFO] Saliendo del programa
```

Cuando esto sucede se envía la notificación al dispositivo móvil para indicar que hay alguien en la puerta y manda el enlace para entrar en la web. La notificación aparecerá en la pantalla de bloqueo y en la barra de notificaciones.

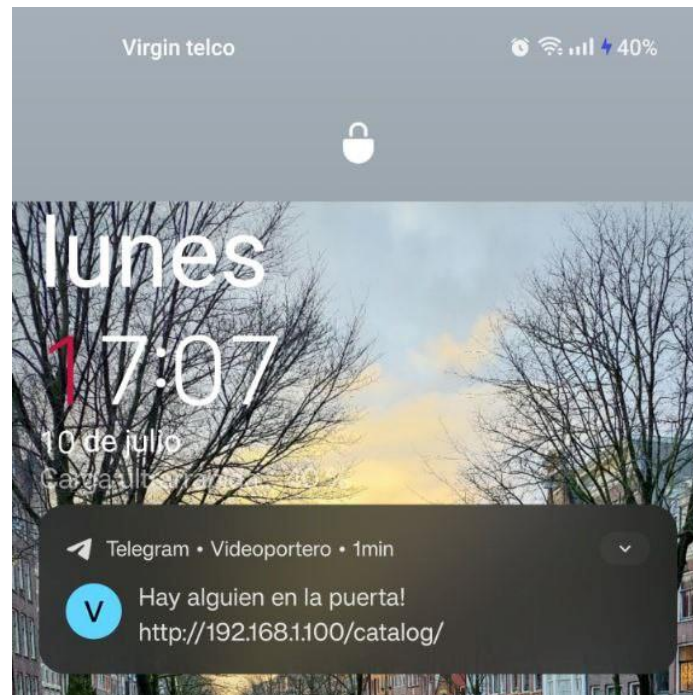


Figura 51. Captura pantalla bloqueo

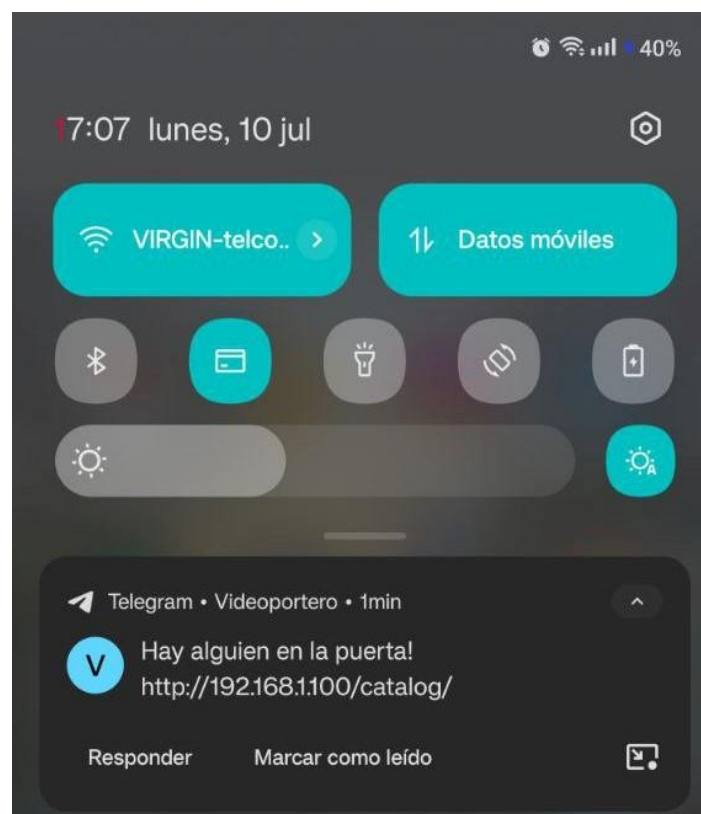


Figura 52. Captura barra notificaciones

En la conversación de Telegram aparecerá de la siguiente forma:



Figura 53. Captura Telegram

Al pinchar en el link llevará a la página web, desde donde se podrá gestionar la apertura o no de la puerta.

La estructura de la página web se ve de la siguiente forma:

En la parte superior de la pantalla aparece el botón para abrir la puerta. Al pulsarlo la puerta se abrirá.

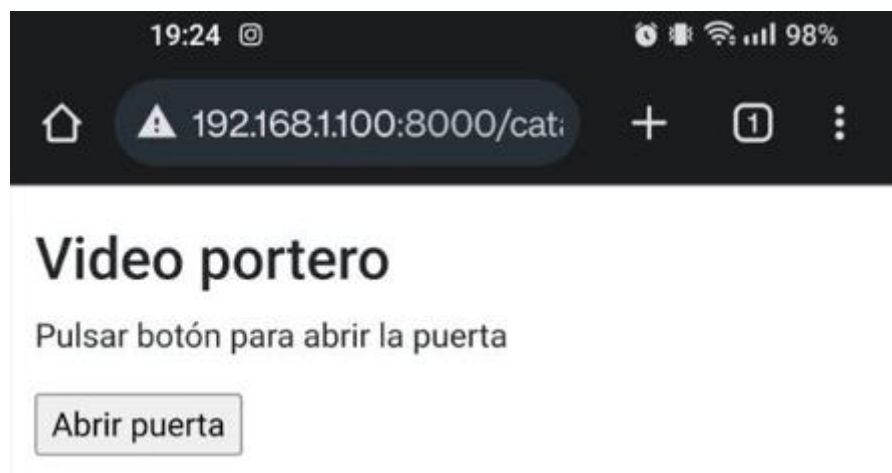


Figura 54. Página web, botón apertura

En la mitad de la pantalla se encuentra la recopilación del vídeo en tiempo real. Se puede ver en quién hay en la puerta en tiempo real.

Cámara



Figura 55. Página web, vídeo en tiempo real

En la parte inferior de la pantalla se observa un registro con las últimas entradas, la fecha del intento de acceso, quién ha entrado (si se ha detectado) y si ha sido exitoso el reconocimiento.

Tabla acceso

ID Usuario	Fecha	Éxito
	None	None
Lore	June 16, 2023, 4:08 p.m.	True
Lore	July 4, 2023, 3:09 p.m.	True
Lore	July 4, 2023, 3:22 p.m.	True
Loreto	July 4, 2023, 3:33 p.m.	False
Loreto	July 4, 2023, 3:35 p.m.	False
Loreto	July 4, 2023, 3:36 p.m.	False
Loreto	July 4, 2023, 3:43 p.m.	True

Figura 56. Página web, tabla de accesos

El conjunto queda de esta forma:

21:47 31%

192.168.1.100:8000/cat:

Video portero

Pulsar botón para abrir la puerta

Abrir puerta

Cámara




Tabla acceso

ID Usuario	Fecha	Éxito
	None	None
Lore	June 16, 2023, 4:08 p.m.	True
Lore	July 4, 2023, 3:09 p.m.	True
Lore	July 4, 2023, 3:22 p.m.	True

Figura 57. Página web

8. ANÁLISIS ECONÓMICO DE LA SOLUCIÓN

Para llevar a cabo el proyecto se ha reutilizado todo el material, por lo que el coste ha sido de cero euros, pero se procede a hacer una lista con opciones para comprar cada uno de los elementos necesarios para la implementación del proyecto.

Raspberry Pi 2 Model B: se puede encontrar en Amazon por 61,86€.

<https://www.raspihc.es/index.php?ver=tienda&accion=verArticulo&idProducto=1751&src=raspberrypi>

También se podrían usar modelos posteriores de Raspberry (el último modelo es Raspberry Pi Model 4) por un precio similar:

<https://www.raspihc.es/index.php?ver=tienda&accion=verArticulo&idProducto=1751&src=raspberrypi>

También existen lotes que vienen con componentes necesarios para la Raspberry como el cable HDMI, fuente de alimentación, carcasa... pero son algo más caros.

<https://amzn.eu/d/fUowWUI>

Fuente alimentación micro USB: como se comentó anteriormente, es recomendable que sea de al menos 1mA. Al margen de eso cualquier cargador micro USB es válido, el cargador del móvil o tableta bastaría.

Se puede encontrar en Amazon por 9,88€

<https://amzn.eu/d/hinyVeR>

Tarjeta de memoria Micro SD: se recomienda que sea de clase 10 y de 32 GB, se puede encontrar en Amazon por 6,90€.

<https://amzn.eu/d/eJpptFO>

Adaptador WiFi USB inalámbrico: se puede encontrar en Amazon por 7,99€

<https://amzn.eu/d/1Yj62eU>

Teclado: cualquiera que funcione por USB, se puede encontrar uno en Amazon por 13€

<https://amzn.eu/d/dRzcfmY>

Ratón: al igual que el teclado, es válido cualquier ratón que funcione por USB. En Amazon se puede encontrar uno por 13€

<https://amzn.eu/d/hm2im6A>

Monitor: se puede encontrar en Amazon por 96.99€

<https://amzn.eu/d/hxv0wKp>

Cable HDMI: se puede encontrar en Amazon por 6,73€

<https://amzn.eu/d/4ohBRIK>

Cámara web: se puede encontrar en Amazon por 19.99€

<https://amzn.eu/d/e7ZxhtR>

Teléfono móvil: es válido cualquier teléfono móvil o tableta con conexión Wifi y acceso al navegador.

Un teléfono móvil que cumple estos requisitos se puede encontrar en MediaMarkt por 130€

https://www.mediemarkt.es/es/product/_movil-galaxy-a04s-samsung-verde-32-gb-3-gb-65-exynos-850-8nm-100467945.html?utm_source=new%20owned&utm_medium=email-other%20email&utm_term=webshare&utm_campaign=webshare

LED: se puede encontrar un lote de 100 diodos LED por 4,25€ en kumótica.

<https://kumotica.es/componentes-robotica/135-lote-100-diodos-led-5-mm-azul.html>

Resistencia: se puede encontrar en Amazon un lote de 200 piezas 7,55€

<https://amzn.eu/d/2KUGFyk>

Pulsador: se puede encontrar un lote de 20 unidades en AliExpress por 0,62€

<https://es.aliexpress.com/item/1005004159746274.html?channel=twinner>

Placa de pruebas y cables: se puede encontrar en Amazon un lote por 14,99€

<https://amzn.eu/d/4CRQpe2>

Los LED, las resistencias, los pulsadores, la placa de prueba y los cables se pueden comprar por separado en los lotes mostrados anteriormente, pero también existen lotes de elementos electrónicos que traen varios tipos de componentes y sale bastante más económico que comprarlos por separados. Por ejemplo, este de AliExpress es muy completo y trae todo lo necesario por 11,28€.

<https://es.aliexpress.com/item/1005001321232974.html?channel=twinner>

Router wifi: se puede encontrar en Amazon por 19,89€

<https://amzn.eu/d/j1Wlx0b>

Además del aparato, se necesita tener contratado internet con alguna compañía. Normalmente son las compañías las que aportan el router y no hace falta comprarlo.

En este enlace se puede encontrar las tarifas de solo fibra de la compañía Digi desde 15€/mes.

<https://www.digimobil.es/fibra-optica/>

Por lo general, es necesario contratar la tarifa del teléfono móvil también con la compañía que se contrate internet y suele ser más barato de esta forma. Una buena opción para fibra y móvil sería la compañía Pepephone, desde 38,90€/mes.

<https://www.pepephone.com/adsl-fibra-movil>

Para calcular el coste de los elementos necesarios se parte de la suposición de que se dispone de internet contratado (y router) y dispositivo móvil. Haciendo esta suposición, se pueden presentar varias opciones:

Si se compra todo lo demás, comprando los componentes electrónicos por separado, el precio total sería de en torno a 265€.

Si se compra el lote de componentes electrónicos en vez de los componentes por separado, el coste sería de unos 250€.

Existe otra opción más, si no se dispone de monitor, cable HDMI, ratón y teclado, o no se desea comprar, la opción más económica es hacer todas las operaciones necesarias en la Raspberry Pi (estas operaciones se explicarán en el manual más adelante) mediante el protocolo SSH⁴², que permite acceder a máquinas remotas (la Raspberry Pi en este caso) a través de una red. Por tanto, con un pc u ordenador portátil no sería necesario comprar los elementos descritos. En este caso, el coste del proyecto comprando el lote de componentes electrónicos sería de unos 120€.

Elemento	Precio (en euros)
Raspberry Pi y accesorios	86,63
Equipo informático	149,71
Componentes electrónicos	11,28
Total	247,62

Figura 58. Tabla de precios

⁴² https://es.wikipedia.org/wiki/Secure_Shell

En cuanto a las horas de trabajo necesarias en el desarrollo del proyecto, se han empleado unas 135 horas. El sueldo medio de un ingeniero en España es de unos 34.865€ brutos al año⁴³, lo que equivale a unos 18,16€/h brutos. Usando esta base, el coste del proyecto ascendería a 2.451€.

⁴³ https://www.glassdoor.es/Sueldos/ingeniero-electronico-sueldo-SRCH_KO0,21.htm

9. ANÁLISIS TEMPORAL

Se procede a desglosar el número de horas, indicando a qué se han empleado.

Unas 18 horas fueron dedicadas a la búsqueda de información sobre reconocimiento facial e implementación de este en una máquina virtual con Raspberry Pi OS en un portátil.

Las instalaciones e implementación del reconocimiento facial en la Raspberry Pi tomaron unas 12 horas.

Por tanto, las horas dedicadas al reconocimiento facial ascienden a 30.

La búsqueda de información, instalación, implementación de la base de datos y su forma de operar con Python para poder escribir y leer desde el programa se llevó a cabo en 12 horas.

En cuanto a la página web, se emplearon unas 12 horas en recopilar información sobre la mejor forma y plataforma para su desarrollo, búsqueda de información sobre Django, instalaciones y configuración del esqueleto del proyecto.

Unas 6 horas fueron empleadas en elaborar una primera versión de la página web e ir probando funcionalidades, así como la investigación de cómo hacer consultas a la base de datos y mostrarlo en la web.

En torno a 12 horas fueron empleadas en la búsqueda de información acerca de mostrar el video en directo en la web y su implementación.

Fueron necesarias unas 36 horas para conseguir lanzar el programa de apertura al pulsar el botón. Se estudió como funcionaban los botones en Django y como llevar a cabo la ejecución del programa desde la vista. Además, se investigó la forma de hacer que no se actualizara la página web al pulsar el botón y de esta forma se encontró la funcionalidad de Ajax y JavaScript para hacerlo posible.

Se emplearon 2 horas en hacer posible el acceso de la web desde cualquier dispositivo de la red y sus correspondientes pruebas con el portátil y dispositivo móvil.

El desarrollo de la página web tomó unas 68 horas en total.

Una vez la estructura del proyecto de videoportero estaba creada, se desarrolló el programa "apertura.py" para encender el LED y el programa "timbre.py" para detectar si se pulsa el pulsador y lanzar el reconocimiento facial si lo hace. Se integraron tanto en el programa de reconocimiento facial como en la página web con sus pruebas correspondientes. Esto tomó unas 7 horas.

En el estudio de las opciones para hacer que se ejecute el programa "timbre.py" al arrancar la Raspberry Pi y su implementación y pruebas se han empleado 5 horas.

Para la creación del bot de Telegram, integración del código y pruebas se han empleado 2 horas.

La limpieza y depuración del código, modificación de la tabla de datos para recopilar la información deseada y estructuración final del conjunto del proyecto tomó unas 9 horas.

En cuanto a la documentación, se han empleado 6 horas para redactar un guion con la información necesaria y la estructura de la implementación de la solución.

En el desarrollo de la memoria se han empleado 78 horas.

En los siguientes gráficos se muestra el peso de cada parte sobre el total del tiempo empleado, uno dividido por las diferentes tecnologías implementadas (figura 58) y otro por las fases generales del proyecto (figura 59).

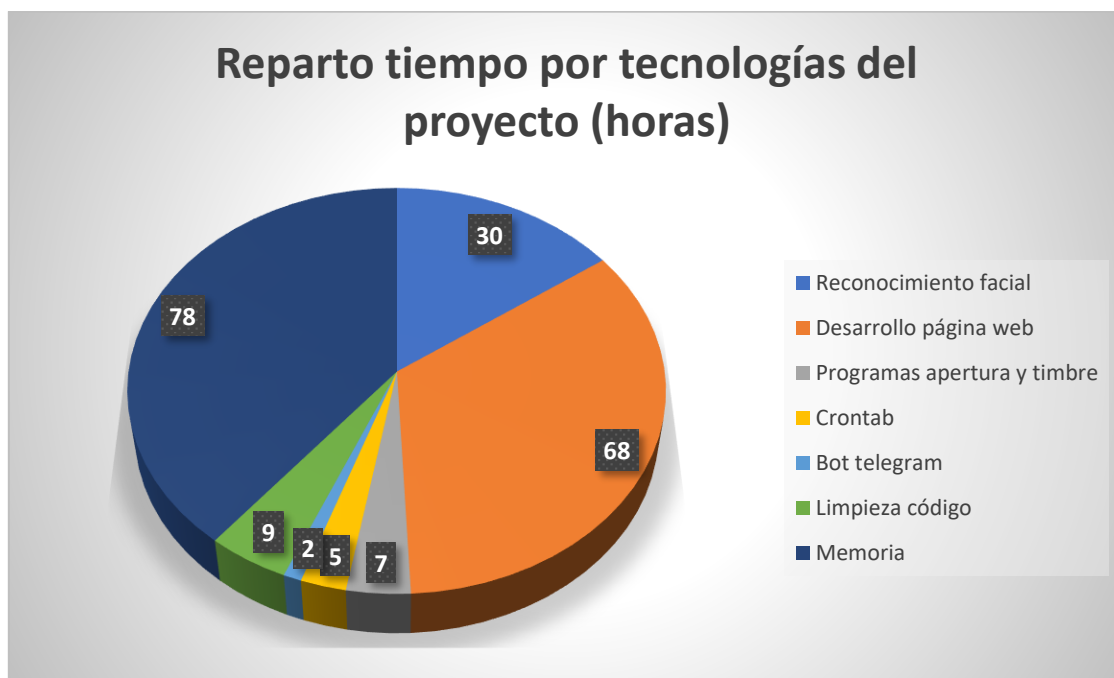


Figura 58. Reparto tiempo por tecnologías del proyecto (horas)

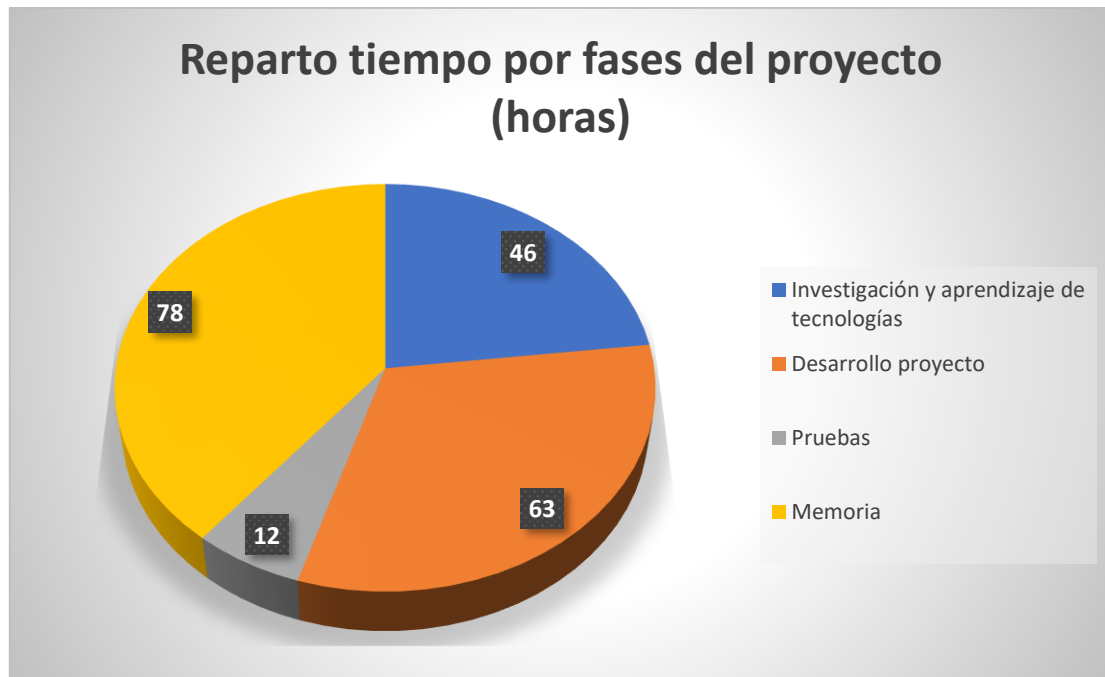


Figura 59. Reparto tiempo por fases proyecto (horas)

9. MANUAL DE DESPLIGUE DE LA APLICACIÓN Y USUARIO

9.1. Despliegue e instalación de la aplicación

Como se indicó en el apartado de análisis económico (capítulo 8), hay dos formas de implementar la solución. Se puede hacer conectando la Raspberry Pi a los periféricos (monitor, ratón y teclado) y usarla como un PC normal, o bien desde un PC o portátil accediendo a la Raspberry Pi mediante protocolo SSH. En este proyecto se ha llevado a cabo conectando la Raspberry a los periféricos y de esta misma forma se desarrollará el manual de usuario.

Sin embargo, si se desea acceder a la Raspberry Pi desde SSH es necesario saber que hay que activarlo en la Raspberry ya que viene desactivado por defecto. Se puede seguir siguiente tutorial para activarlo:

<https://www.ionos.es/digitalguide/servidores/configuracion/activar-ssh-en-raspberry-pi/>

Para copiar y pegar archivos a Raspberry Pi mediante SSH se pueden seguir las indicaciones de la siguiente web:

<https://geekytheory.com/tutorial-raspberry-pi-14-como-transferir-archivos-por-ssh/>

Se procede con las indicaciones para implementar en proyecto conectando la Raspberry Pi a los periféricos.

El primer paso es la preparación del sistema operativo, donde conectando la tarjeta de memoria al PC se le da el formato correcto, se descarga el sistema operativo Raspberry Pi OS y se instala en la tarjeta de memoria. Este procedimiento está explicado paso por paso en el apartado 6.1 del capítulo 6.

Una vez instalado, introduce la tarjeta de memoria en la Raspberry Pi, se conectan los periféricos y la alimentación y se arranca, haciendo la configuración inicial. Este procedimiento está explicado paso por paso en el apartado 6.1 del capítulo 6.

El siguiente paso es la implementación del reconocimiento facial. Para ello no hay más que seguir paso por paso el apartado 6.2.1 del capítulo 6, donde se instalan las librerías necesarias para la puesta en marcha del proyecto.

Completadas las instalaciones, se copia la carpeta "FacialRecognitionProject" (incluida en el soporte informático) dentro del directorio `/home/<usuario>`.

En este punto, todo estará listo para registrar a los usuarios a los que reconocer. Para ello hay que ejecutar el programa “FaceDataset.py”, que tomara 30 fotos al usuario a registrar.

Lo ideal es que se hagan en el mismo sitio donde se va a reconocer a la persona (la puerta de la casa) y que la persona vaya cambiando de expresión según vaya tomando las muestras el programa.

Para ejecutarlo sólo es necesario implementar la siguiente línea en el terminal:

```
python3 FaceDataset.py
```

A ejecutarlo pedirá un número para asignarle al nuevo usuario y posteriormente tomará las capturas. Una vez tomadas se cerrará solo.

Una vez tomadas las fotos, hay que entrenar al modelo para que reconozca a la persona. Para hacerlo no se necesita más que ejecutar el fichero “FaceTrainer.py” en el terminal y se hará automáticamente.

```
python3 FaceTrainer.py
```

Hay que tener en cuenta que habrá que realizar este procedimiento cada vez que se quiera añadir un usuario nuevo para su reconocimiento.

Para montar el circuito hay que hacerlo como indica el esquema del apartado 6.2.7 del capítulo 6. El pin 15 se conectará al LED y el 24 al pulsador.

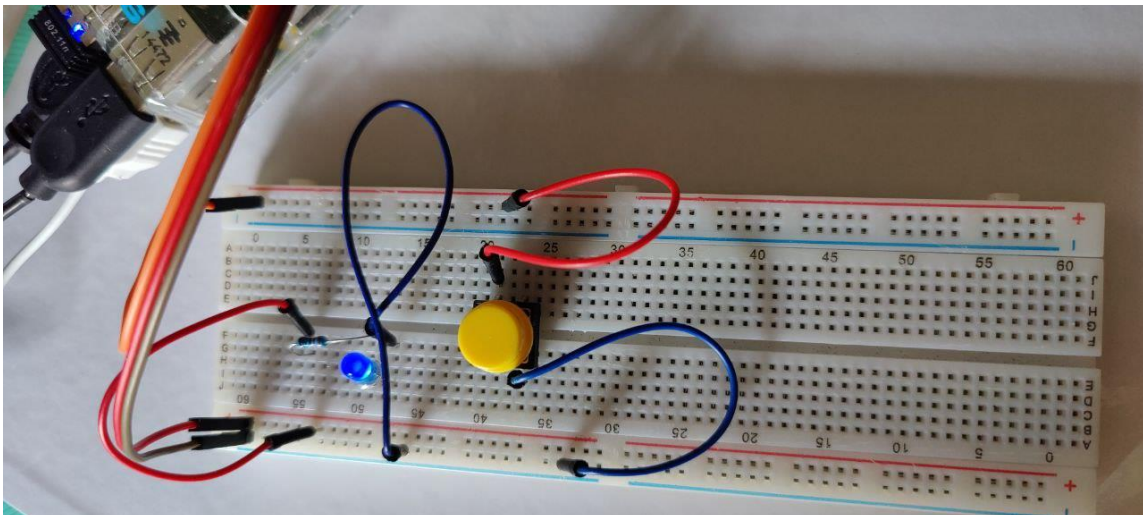


Figura 60. Montaje real LED y pulsador

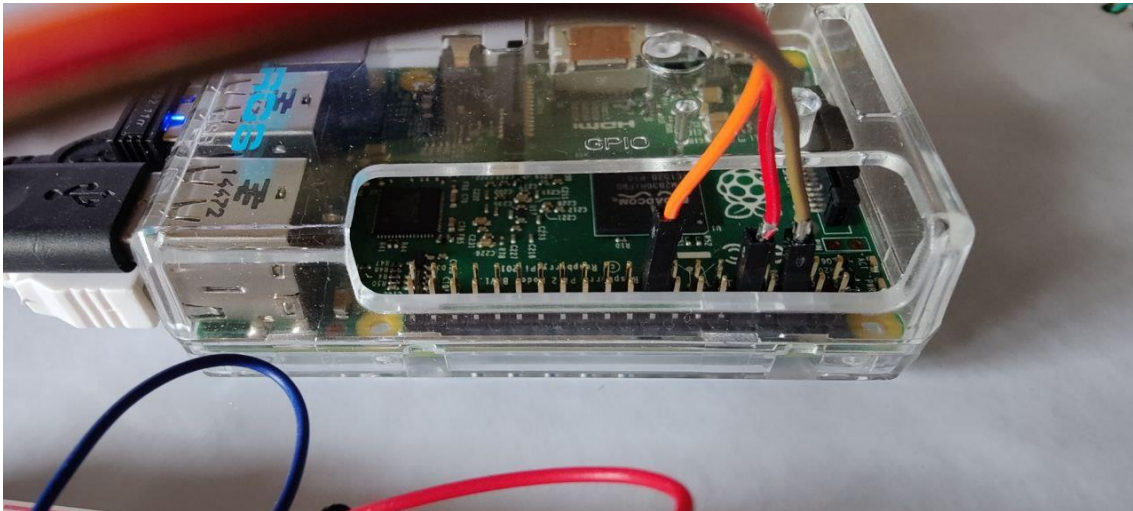


Figura 61. Montaje real Raspberry Pi

También será necesario dar permiso de ejecución al programa “timbre.py” y modificar el fichero “crontab” para que “timbre.py” se ejecute automáticamente desde el arranque de la Raspberry Pi. Este procedimiento se encuentra en el apartado 6.2.7 del capítulo 6.

Hay que tener en cuenta el nombre de usuario que se haya elegido, en este caso ha sido “log”, por tanto, la ruta del proyecto está en `/home/log/`, pero para otro nombre de usuario distinto debería cambiarse “log” por el usuario en cuestión. Hay que tener esto en cuenta para las rutas que aparezcan en todo el código.

A continuación, es necesario configurar la IP fija en la Raspberry Pi. Para ello, se seguirán el apartado 6.3.1 del capítulo 6, donde viene explicado con detalle.

Una vez se configure y se compruebe que la IP está fija, se instala MariaDB. El procedimiento viene desarrollado en el apartado 6.3.2 del capítulo 6.

Para el entorno web, el primer paso será la instalación y preparación del entorno de desarrollo, que se explica en el apartado 6.3.3 del capítulo 6.

Creado el esqueleto del proyecto, será necesario modificar algunos de los ficheros existentes y añadir otros nuevos.

El fichero `/website/website/settings.py` se modifica de la forma que se indica en el apartado 6.3.4 del capítulo 6. Hay que añadir la nueva aplicación a “INSTALLED_APPS” cambiar la base de datos en “DATABASES”, modificar “TIME_ZONE” y poner “DEBUG” a “False”.

En el archivo “settings.py” también habrá que añadir la IP fija de la Raspberry en “ALLOWED_HOSTS”:

```
ALLOWED_HOSTS = ['192.168.1.100']
```


El archivo `/website/website/urls.py` se puede sustituir por el archivo correspondiente que se encuentra la ruta del soporte informático `djangos/website/website/urls.py`.

Es necesario copiar los ficheros del soporte informático `djangos/website/catalog/` “urls.py”, “apertura.py” y “camera.py” dentro de la carpeta del proyecto `/website/catalog/`.

Se copian las carpetas del soporte informático `djangos/website/catalog/` “templates” y “static” en la carpeta del proyecto `/website/catalog/`.

Por último, se copia el contenido de los ficheros del soporte informático `djangos/website/catalog/` “__init__.py”, “admin.py”, “apps.py”, “models.py” y “views.py” dentro de sus homólogos en la carpeta del proyecto `/website/catalog/`.

Tras esto, es necesario arrancar el servidor también, ejecutando previamente los comandos “makemigrations” y “migrate”.

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver 0.0.0.0:8000
```

Tras estos pasos, el reconocimiento facial estará listo. El programa “timbre.py” está constantemente ejecutándose en segundo plano y listo para lanzar el programa de reconocimiento facial cuando se pulse el pulsador y abrir la puerta si reconoce al usuario.

Lo último por configurar sería el envío de notificaciones a Telegram. Para ello es necesario instalar Telegram en el dispositivo móvil, registrarse (si ya se dispone de la aplicación y/o la cuenta no sería necesario) y seguir los pasos para crear el bot, obtener la “api_key” y la id del chat. Todos estos pasos vienen explicados de forma detallada en el apartado 6.3.7 del capítulo 6.

Hay que tener en cuenta que la “api_key” y la id del chat serán distintas dependiendo del dispositivo, así que es necesario hacer un pequeño cambio en el código. En la carpeta “FacialRecognitionProject” se abre el programa “FaceRecognition.py” con un entorno de desarrollo, Thonny por ejemplo, que ya viene instalado.

Habrá que modificar los parámetros “chat_id” y “api_key” por los correspondientes que se hayan obtenido siguiendo los pasos del apartado 6.3.7. También es necesario cambiar la IP por la IP fija de la Raspberry.

Una vez cambiado, se guarda el proyecto y se cierra.

```

if (time.time() > end_time):
    percent = 100 - confidence
    success = 0
    passdata(success, id)
    chat_id = "458778819"
    api_key =
"6355325056:AAHwqlZPHhrq41iwAhVye7Kw_91yWPktF5c"
    send_telegram_message("Hay alguien en la
puerta!", chat_id, api_key)

send_telegram_message("http://192.168.1.100:8000/catalog/",
chat_id, api_key)
    break

```

Tras este paso, ya estaría todo preparado para el reconocimiento facial.

9.2. Manual de uso de la aplicación

Una vez realizado el despliegue de la aplicación, ya se puede operar con la interfaz de usuario. La interfaz de uso del sistema es bastante sencilla.

Para que se lance el reconocimiento facial solo hay que pulsar el pulsador. Durante un tiempo intentará reconocer a la persona y si lo hace, abrirá la puerta de forma automática.

Por otra parte, si llaman a la puerta (se pulsa el pulsador) y no se reconoce a la persona, llegará al dispositivo móvil una notificación avisando de que hay alguien en la puerta y un enlace para la página web tal como se muestra en el capítulo 7 (ver figuras 51, 52 y 53).

En la página web se puede ver la cara de la persona que está intentando acceder, el registro de entradas y un botón de apertura que sirve para abrir la puerta manualmente (ver figuras 54, 55, 56 y 57).

Como información adicional, la URL para acceder a la página web será la IP de la Raspberry Pi añadiéndole :8000. Es decir, http://<IP_Raspberry>:8000/. En el mensaje que llega a Telegram aparece el enlace directamente, pero se puede acceder desde el navegador de cualquier dispositivo que esté conectado a la misma red wifi.

10. CONCLUSIONES

Tras el desarrollo de este proyecto y todo el proceso de documentación que conlleva para su elaboración, se hace evidente que existen muchísimas soluciones parecidas en el mercado de videoportero, pero ninguna incluye el reconocimiento facial ni abrir la puerta automáticamente.

Cabe preguntarse por qué algo tan relativamente sencillo de implementar como es el reconocimiento facial no está extendido a la hora de permitir la entrada a sitios, siendo mucho más populares otro tipo de tecnología como es la huella digital.

La causa de esto es que hoy en día la seguridad sigue siendo un hándicap para el reconocimiento facial. Desde una fotografía o vídeo de la persona, o incluso una máscara impresa en 3D son formas en las que actualmente se puede “engañar” al reconocimiento facial.

Incluso en los teléfonos móviles, donde está muy extendido el reconocimiento facial para desbloquear el dispositivo, existen otros niveles de autenticación como una contraseña segura para realizar ciertos cambios en el sistema aún con el teléfono desbloqueado. Esta doble seguridad busca evitar que otra persona que haya conseguido pasar el reconocimiento facial ya sea por los métodos descritos antes o bien forzando a la persona pueda hacer cambios de importancia en el teléfono.

También influye mucho la iluminación y calidad de la imagen, ya que debido a esto podría llegarse a no reconocer a una persona que el sistema sí está preparado para reconocer.

En resumen, debido a los problemas de seguridad y a los fallos en reconocimientos derivados de la calidad y la iluminación, aún queda un largo camino por recorrer para que un videoportero pueda integrar la función de reconocimiento facial y apertura automática de forma segura y fiable.

11. TRABAJO FUTURO

En cuanto a trabajo futuro, hay varios aspectos en los que se podría trabajar para seguir mejorando e incluyendo funcionalidad al modelo actual.

Se podría implementar un asistente de voz para que el reconocimiento facial se lanzase mediante una frase en vez de pulsar el pulsador. De esta forma, al pulsar el pulsador se avisaría directamente mediante la notificación al dispositivo de que hay alguien en la puerta, y solo intentaría el reconocimiento si se activa mediante voz.

La ventaja es que de esta forma no se intentaría reconocer a todo el mundo que llama al timbre, ya que personas como repartidores de correo, mensajería u otras personas que vayan al domicilio que no están registrados como usuarios no van a ser reconocidos y por tanto no tiene mucho sentido que se intente lanzar el reconocimiento facial.

Aplicar técnicas biométricas multimodales⁴⁴ incrementan bastante la seguridad, ya que hace que el sistema sea mucho más robusto y fiable. Por ejemplo, en el control de fronteras (donde se requiere mucha seguridad y fiabilidad) se usa el reconocimiento facial en el control de pasaporte para los pasaportes electrónicos, pero ya existen proyectos como el “eBorder”⁴⁵ que pretenden implementar técnicas biométricas multimodales (rasgos faciales y huella dactilar en este caso) para aportar mayor seguridad.



Figura 62. Control pasaporte en frontera

⁴⁴ <https://www.computerworld.es/archive/autenticacion-multimodal-una-tecnica-biometrica-precisa#:~:text=La%20idea%20es%20que%20la,captura%20m%C3%A1s%20de%20una%20vez.>

⁴⁵ <https://www.casadomo.com/2022/07/04/proyecto-eborder-desarrollara-nuevo-dispositivo-biometrico-multimodal-inalambrico>

Otra nueva funcionalidad que se podría incluir sería unos sensores de luminosidad y una luz, para que cuando se vaya a lanzar el reconocimiento se encienda la luz si está oscuro para mejorar las condiciones de luminosidad y hacer más efectivo el reconocimiento facial.

También podría incluirse el audio, que pueda hablar la persona que está en la puerta y se reproduzca en el dispositivo que tiene la web abierta y que se pueda contestar desde el dispositivo se reproduzca en un pequeño altavoz que se integre. Esto sería muy ventajoso cuando llaman a la puerta personas que no son conocidas.

En cuanto a la página web, se podría implementar la misma funcionalidad en aplicaciones desarrolladas para Android y iOS.

Por último, se podría hacer accesible desde cualquier lugar, no solo estando dentro de la misma red Wifi. De esta forma se podría ver quien hay fuera, comunicarse con esa persona y abrir la puerta aunque no se esté en el domicilio.

Este proyecto es sólo un prototipo, pero cabría la posibilidad de hacer un estudio para una industrialización de la solución a gran escala. Usando una placa de bajo coste como es Raspberry Pi, podría convertirse en una solución más económica de las que hay actualmente en el mercado.

12. BIBLIOGRAFÍA

[1] BEREZGOV, Anzor, 2022. De Python a Telegram: crea tu propio sistema de notificación. En: *hackernoon* [en línea]. Disponible en:

<https://hackernoon.com/es/de-python-a-telegram-construye-tu-propio-sistema-de-notificaciones>

[2] CABRERABENITO, Luis, 2020. Raspberry y Python: encender y apagar LED. En: *Parzibyte's blog* [en línea]. Disponible en:

<https://parzibyte.me/blog/2020/02/09/raspberry-python-encender-apagar-led/>

[3] DE LUZ, Sergio, 2023. Utiliza Cron y Crontab para programar tareas en tu servidor. En: *redes zone* [en línea]. Disponible en:

<https://www.redeszone.net/tutoriales/servidores/cron-crontab-linux-programar-tareas/>

[4] *DIY Usthad* [en línea]. How to get your Telegram chat ID. Disponible en:

<https://diyusthad.com/2022/03/how-to-get-your-telegram-chat-id.html>

[5] FABIEN, Maël, 2019. “A guide to Face Detection in Python (With Code)”. En: *Medium* [en línea]. Disponible en:

<https://towardsdatascience.com/a-guide-to-face-detection-in-python-3eab0f6b9fc1#:~:text=scaleFactor%20%3A%20Parameter%20specifying%20how%20much,maxSize%20%3A%20Maximum%20possible%20object%20size.>

[6] FLORES MORENO, Dante Maximiliano, 2018. Django — Conecta tu proyecto con la base de datos MySQL. En: *Medium* [en línea]. Disponible en:

<https://medium.com/@a01207543/django-conecta-tu-proyecto-con-la-base-de-datos-mysql-2d329c73192a>

[7] FROMAGET, Patrick. ¿Cómo Instalar MariaDB en Raspberry Pi? (Servidor MySQL). En: *RaspberryTips* [en línea]. Disponible en:

<https://raspberrytips.es/instalar-mariadb-raspberry-pi/>

[8] FROMAGET, Patrick. “Get Started The Right Way With OpenCV On Raspberry Pi.” En: *RaspberryTips* [en línea]. Disponible en:

<https://raspberrytips.com/install-opencv-on-raspberry-pi/>

[9] FROMAGET, Patrick ¿Raspberry Pi: ¿Cómo Iniciar Automáticamente Un Programa? En: *RaspberryTips* [en línea]. Disponible en:

<https://raspberrytips.es/iniciar-un-programa-raspberry-pi/#:~:text=Hay%20varias%20soluciones%20para%20iniciar,Settings%C2%BB%20para%20hacer%20lo%20mismo.>

[10] HEDGPETH, Rob, 2020. “*How to connect Python programs to MariaDB*” En: *MariaDB* [en línea]. Disponible en:

<https://mariadb.com/resources/blog/how-to-connect-python-programs-to-mariadb/>

[11] Ionos [en línea], 2022. Asignar una dirección IP fija a Raspberry Pi. Disponible en:

<https://www.ionos.es/digitalguide/servidores/configuracion/como-asignar-una-ip-fija-a-raspberry-pi/>

[12] Ley Orgánica 3/2018, de Protección de Datos Personales y garantía de los derechos digitales (BOE núm. 294, de 06/12/2018) [en línea]. Disponible en:

<https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673>

[13] LLAMAS, Luis, 2018. Configurar IP estática en Raspberry Pi. En: *Luis Llamas* [en línea]. Disponible en:

<https://www.luisllamas.es/raspberry-pi-ip-estatica/>

[14] MACHO, Juan Carlos. Condicionales y pulsadores. En: *PROMETEC* [en línea]. Disponible en:

<https://www.prometec.net/condicionales-y-pulsadores/>

[15] MAITHANI, Aniket, 2018. A “*Opencv Live Stream from camera in Django Webpage.*” En: *stack overflow* [en línea]. 12 abril 2018 21:04 [consulta 5 abril 2018]. Disponible en:

<https://stackoverflow.com/questions/49680152/opencv-live-stream-from-camera-in-django-webpage>

[16] *MDN Web Docs* [en línea], 2023. Tutorial Django: El Sitio Web de La Biblioteca Local. Disponible en:

https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Tutorial_local_library_website

[17] *OpenCV* [en línea]. “*Cascade classifier*”. Disponible en:

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[18] *OpenCV* [en línea]. “*Face Detection using Haar Cascades.*” Disponible en:

https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html

[19] ROVAI, Marcelo, 2018. “*Real-Time Face Recognition: An End-to-End Project.*” En: *hackster.io* [en línea]. Disponible en:

<https://www.hackster.io/mjrobot/real-time-face-recognition-an-end-to-end-project-a10826>

[20] *Senstar* [en línea]. Problemas del reconocimiento facial. Disponible en:

<https://senstar.com/es/senstarpedia/problemas-del-reconocimiento-facial/>

[21] SOLANO, Gabriela, 2020. DETECCIÓN DE ROSTROS con Haar Cascades Python – OpenCV. En: *OMES* [en línea]. Disponible en:

<https://omes-va.com/deteccion-de-rostros-con-haar-cascades-python-opencv/>

[22] *SuperDataScience* [en línea], 2017. Face recognition using OpenCV and Python: A beginner's guide. Disponible en:

<https://www.superdatascience.com/blogs/opencv-face-recognition>

[23] TATXEN, 2019. A “Ejecutar archivo.sh nada más encender Raspberry Pi”. En: *ForoRaspberry.es* [en línea]. 22 abril 2019 15:14 [consulta 22 abril 2019]. Disponible en:

<https://www.fororaspberry.es/viewtopic.php?t=12966>

[24] *Thales* [en línea]. Reconocimiento facial: ¿cuáles son los problemas de seguridad? Disponible en:

<https://www.thalesgroup.com/es/countries/americas/latin-america/dis/gobierno/inspiracion/problemas-seguridad-reconocimiento-facial>

[25] VILLALPANDO, Juan Antonio. Servidor Web Apache. PHP. MySQL (MariaDB). PhpMyAdmin. En: *KIO4.COM* [en línea]. Disponible en:

http://kio4.com/raspberry/20_apache_php_mysql.htm

[26] VIOLA, Paul y JONES, Michael, 2021. “*Rapid Object Detection using a Boosted Cascade of Simple Features.*” En: *CMU School of Computer Science* [en línea]. Disponible en:

<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>

[27] *Wikipedia* [en línea]. Bot. Disponible en:

<https://es.wikipedia.org/wiki/Bot>

[28] *Wikipedia* [en línea]. Secure Shell. Disponible en:

https://es.wikipedia.org/wiki/Secure_Shell

ANEXO

Anexo 1. Toma de muestras

Recolección de muestras del usuario a añadir. "FaceDataset.py"

```
import cv2
import numpy as np
import os
import imutils
from apertura import *

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
# Load the classifier
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX

id = 0 # Iniciate id counter
names = ['None', 'Loreto'] # Names related to ids

# Initialize and start realtime video capture
cam = cv2.VideoCapture('/dev/video0')

# Define min window size to be recognized as a face
minW = 0.1 * cam.get(3)
minH = 0.1 * cam.get(4)

while True:
    ret, img = cam.read()
    img = imutils.resize(img, width=480)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Detect faces on the image
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(int(minW), int(minH)),
    )

    # Mark rectangle in the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255,
0), 2)
        # Analyze face and return owner
```

```

        id, confidence = recognizer.predict(gray[y:y + h,
x:x + w])

        # Check if confidence is less than 100."0" is
perfect match
        if (confidence < 100):
            id = names[id]
            confidence = " {0}%".format(round(100 -
confidence))
            abrir(id) # Open the door (turn on LED)
        else:
            id = "desconocido"
            confidence = " {0}%".format(round(100 -
confidence))
            # lanzar notificacion

            cv2.putText(img, str(id), (x + 5, y - 5), font, 1,
(255, 255, 255), 2)
            cv2.putText(img, str(confidence), (x + 5, y + h -
5), font, 1, (255, 255, 0), 1)

            cv2.imshow('camera', img)
            k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting
video

            if k == 27:
                break
# Cleanup
print("\n [INFO] Saliendo del programa")
cam.release()
cv2.destroyAllWindows()

```

Anexo 2. Entrenamiento del modelo

Entrenamiento del modelo. "FaceTrainer.py"

```
import cv2
import numpy as np
from PIL import Image
import os
import imutils

# Path for face image database
path = 'dataset'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml"
);

# function to get the images and label data
def getImagesAndLabels(path):
    imagePath = [os.path.join(path, f) for f in
os.listdir(path)]
    faceSamples = []
    ids = []

    for imagePath in imagePath:
        PIL_img = Image.open(imagePath).convert('L') #
convert it to grayscale
        img_numpy = np.array(PIL_img, 'uint8')
        id = int(os.path.split(imagePath)[-
1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)

        for (x, y, w, h) in faces:
            faceSamples.append(img_numpy[y:y + h, x:x + w])
            ids.append(id)

    return faceSamples, ids

print("\n [INFO] Entrenando caras. Tomara unos segundos,
espere por favor ...")

faces, ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))
recognizer.write('trainer/trainer.yml') # Save the model
into trainer/trainer.yml

# Print the numer of faces trained and end program
print("\n [INFO] {0} Caras entrenadas. Saliendo del
programa".format(len(np.unique(ids))))
```

Anexo 3. Reconocimiento facial

Reconocimiento facial. "FaceRecognition.py"

```

import cv2
import numpy as np
import os
import imutils
import time
from apertura import *
from pymdb import *
from telegram import *

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('/home/log/FacialRecognitionProject/trainer/
trainer.yml')
# Load the classifier
cascadePath =
"/home/log/FacialRecognitionProject/haarcascade_frontalface
_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX

confidence = 200 # Initialize confidence
id = 0 # Initialize id counter
names = ['None', 'Loreto'] # Names related to ids

# Initialize and start realtime video capture
cam = cv2.VideoCapture('/dev/video0')

# Define min window size to be recognized as a face
minW = 0.1 * cam.get(3)
minH = 0.1 * cam.get(4)

ini_time = time.time()
end_time = ini_time + 30
while True:
    ret, img = cam.read()
    img = imutils.resize(img, width=480)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Detect faces on the image
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(int(minW), int(minH)),
    )

    # Mark rectangle in the faces

```

```

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255,
0), 2)
        # Analyze face and return owner
        id, confidence = recognizer.predict(gray[y:y + h,
x:x + w])
        # Check if confidence is less than 100."0" is
perfect match
        if (confidence < 100):
            id = names[id]
            confidencep = " {0}%".format(round(100 -
confidence))
        else:
            id = "Desconocido"
            confidencep = " {0}%".format(round(100 -
confidence))
            # lanzar notificacion

            cv2.putText(img, str(id), (x + 5, y - 5), font, 1,
(255, 255, 255), 2)
            cv2.putText(img, str(confidencep), (x + 5, y + h -
5), font, 1, (255, 255, 0), 1)

            cv2.imshow('camera', img)

            if (confidence < 30):
                abrir(id) # Open the door (turn on LED)
                percent = 100 - confidence
                success = 1
                passdata(success, id)
                break
            else:
                if (time.time() > end_time):
                    percent = 100 - confidence
                    success = 0
                    passdata(success, id)
                    chat_id = "458778819"
                    api_key =
"6355325056:AAHwqlZPHhrq41iwAhVye7Kw_91yWPktF5c"
                    send_telegram_message("Hay alguien en la
puerta!", chat_id, api_key)

                    send_telegram_message("http://192.168.1.100:8000/catalog/",
chat_id, api_key)
                    break

# Cleanup
cam.release()
cv2.destroyAllWindows()

```

Anexo 4. Encendido de LED

Encendido del LED. “apertura.py”

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(15,GPIO.OUT)

# While loop
def abrir(self):
    # set GPIO15 pin to HIGH
    GPIO.output(15,GPIO.HIGH)
    # pause for one second
    time.sleep(3)
    # set GPIO15 pin to HIGH
    GPIO.output(15,GPIO.LOW)
```

Anexo 5. Comprobación de pulsación pulsador

Código para vigilar si se pulsa el pulsador y lanzar el reconocimiento facial. "timbre.py".

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(24,GPIO.IN,pull_up_down=GPIO.PUD_UP)

while(True):
    status=GPIO.input(24)
    if status == False:
        exec(open("/home/log/FacialRecognitionProject/FaceRecognition.py").read())
```


Anexo 6. Fichero que contiene las URL de Django

Django. "urls.py"

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('camera/', views.livefe, name="live_camera"),
    path('apertura/', views.abrir, name="apertura"),
]
```

Anexo 7. Fichero que contiene las vistas de Django

Django. "views.py"

```

from django.shortcuts import render

# Create your views here.
from django.views import generic
from .models import doorRelease
from django.urls import reverse
from django.contrib import messages
from django.contrib.messages.views import
SuccessMessageMixin
from django import forms
from .camera import *
from .apertura import *
from django.http import JsonResponse
from django.http import HttpResponse
from django.http import StreamingHttpResponse
from django.views.decorators import gzip
import json

@gzip.gzip_page
def livefe(request):
    try:
        cam=VideoCamera()
        return StreamingHttpResponse(gen(cam),
content_type="multipart/x-mixed-replace;boundary=frame")
    except:
        pass

def apertura(request):
    if request.method=='POST':
        apertura.abrir()
        response_data='successful'
        return JsonResponse(response_data)
    else:
        return HttpResponse(json.dumps({"nothing to see":
"this is not happening"}), content_type="application/json")

def index(request,*args,**kwargs):
    datalist = doorRelease.objects.all()
    return render(request,'index.html',context=
{'datalist':datalist})

```

Anexo 8. Estructura página web

Django. "base_generic.html"

```
<!DOCTYPE html>
<html lang="en">
  <head>
    {% block title %}
      <title>Local Library</title>
    {% endblock %}
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1" />

    <link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css
/bootstrap.min.css"
    rel="stylesheet"
    integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ
6jIW3"
    crossorigin="anonymous" />
    <script src="https://code.jquery.com/jquery-3.1.1.js"
integrity="sha256-
16cdPddA6VdVInumRGo6IbivbERE8p7CQR3HzTBuELA="
crossorigin="anonymous"></script>
    <!-- Add additional CSS in static file -->
    {% load static %}
    <link rel="stylesheet" href="{% static 'css/styles.css'
%}" />
    <script src="{% static 'css/styles.css' %}" /></script>
  </head>
  <body>
    <div class="container-fluid">
      <div class="row">
        <div class="col-sm-2">
          {% block sidebar %}
            <ul class="sidebar-nav">
            </ul>
          {% endblock %}
        </div>
        <div class="col-sm-10 ">{% block content %}{%
endblock %}</div>
      </div>
    </div>
  </body>
</html>
```

Anexo 9. Página web

Django. "index.html"

```
{% extends "base_generic.html" %}

{% block content %}
<h1>Video portero</h1>
<p>
  Pulsar botón para abrir la puerta
  {% block button %}
<form method="POST" id="post-form">
  {% csrf_token %}
  <input type="submit" value="Abrir puerta"></button>
</form>
  {% endblock %}
<script>
  $(document).ready(function() {
    $('#post-form').on('submit',function(event) {
      event.preventDefault();
      $.ajax({
        url: "{% url 'apertura' %}",
        type:"POST",
        data:{

csrfmiddlewaretoken:${'input[name=csrfmiddlewaretoken]'.val(),

        },
        success:function() {
        },
      });
    });
  });
</script>
</p>
<h2>Cámara</h2>
<div>
  
</div>
<h2>Tabla acceso</h2>
<table class="table-striped">
  <thead>
    <tr>
      <th scope="col">ID</th>
      <th scope="col">Usuario</th>
      <th scope="col">Fecha</th>
      <th scope="col">Éxito</th>
    </tr>
  </thead>
  <tbody>
```

```
{% for datalist in datalist %}
  <tr>
    <th scope="row">{{ doorrelease.id }}</th>
    <td>{{ datalist.userid }}</td>
    <td>{{ datalist.recddate }}</td>
    <td>{{ datalist.success }}</td>
  </tr>
{% endfor %}
</tbody>
</table>
{% endblock %}
```

Anexo 10. Registro de entrada a base de datos

Programa que escribe en la base de datos. "pymdb.py"

```
import sys
import time

def pasodata(result,userid):
    try:
        conn=mariadb.connect(
            user="phpmyadmin",
            password="1234",
            host="localhost",
            port=3306,
            database="FacialRecognition"
        )

        except mariadb.Error as e:
            print(f"Error conectando: {e}")
            sys.exit(1)

        cur = conn.cursor()

        try:
            sql = "INSERT INTO
catalog_doorrelease(recdate,success,userid)
VALUES (NOW(),%s,%s) "
            val = (result,userid)
            cur.execute(sql,val)
        except mariadb.error as e:
            print(f"Error insert: {e}")
        conn.commit()

        conn.close()
```

Anexo 11. Envío mensaje a Telegram

Programa para enviar mensajes a Telegram. “telegram.py”

```
import requests
import json

def send_telegram_message(message: str, chat_id: str,
    api_key: str, proxy_username: str = None, proxy_password:
    str = None, proxy_url: str = None):
    responses = {}
    proxies = None

    if proxy_url is not None:
        proxies = { 'https':
f'http://{username}:{password}@{proxy_url}', 'http':
f'http://{username}:{password}@{proxy_url}' }

    headers = {'Content-Type': 'application/json', 'Proxy-
Authorization': 'Basic base64'}
    data_dict = {'chat_id': chat_id, 'text': message,
'parse_mode': 'HTML', 'disable_notification': True}
    data = json.dumps(data_dict)
    url =
f'https://api.telegram.org/bot{api_key}/sendMessage'
    response = requests.post(url, data=data,
headers=headers, proxies=proxies, verify=False)
    return response
```