Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Lectura en tiempo real de MRZ (Zona de lectura mecánica) en dispositivos móviles

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Jonás Baptiste Escrivá García

Tutor: María José Castro Bleda

Tutor externo: Hilbert Pieter Mostert

Curso 2020-2021

Resumen

El procesamiento de imágenes se ha visto optimizado drásticamente en los últimos años, de forma que actualmente es posible realizar este procesamiento en tiempo real para, por ejemplo, leer el MRZ de un documento de identidad o un pasaporte. El MRZ, en el caso de los documentos de identidad españoles (DNI) se encuentra en la parte trasera del mismo y consta de 3 líneas de 30 caracteres cada una y contiene información importante del ciudadano en cuestión.

En este trabajo se ha desarrollado una librería en C++ para la detección y lectura del MRZ. Adicionalmente, se ha diseñado e implementado una aplicación Android que usa la cámara del móvil para detectar el MRZ y extraer su información. El trabajo se ha completado con una aplicación web con la que mediante el uso de una web cam y de la misma librería se detecta el MRZ.

Palabras clave: documento de identidad, MRZ, librería, procesamiento de imágenes, Android, aplicaciones web.

Abstract

Image processing has been so drastically improved over the last years that it has now become possible to do this processing in real time. This can be used, for example, to read the MRZ in an identification document or a passport. The MRZ, on Spanish identification documents (DIN) is on the back side and consists of 3 lines of 30 characters each and contain important information about the citizen.

In this work, a library in C++ has been developed for the detection and reading of the MRZ. Additionally, an Android application has been designed and implemented, it uses the mobile camera to detect the MRZ and extract its information. The work has been completed with a web application that, using the webcam and the same library, detects the MRZ.

Keywords: identity document, MRZ, library, image processing, Android, web applications.

Resum

El processament d'imatges s'ha vist optimitzat dràsticament els últims anys, de tal forma que actualment es possible realitzar aquest processament en temps real per a, per exemple, llegir el MRZ de un document de identitat o un passaport. El MRZ, en el cas dels documents de identitat espanyols (DNI) es troba en la part de darrere del mateix i consta de 3 línies de 30 caràcters cadascuna i contenen informació important sobre el ciutadà en qüestió.

En aquest treball s'ha desenvolupat una llibreria en C++ per a la detecció i lectura del MRZ. Addicionalment, s'ha dissenyat i implementat una aplicació Android que utilitza la càmera del mòbil per a detectar el MRZ i extraure la seva informació. El treball s'ha completat amb una aplicació web amb la qual mitjançant el ús de una web cam i de la mateixa llibreria es detecta el MRZ.

Paraules clau: document de identitat, MRZ, llibrera, processament de imatges, Android, aplicacions web.

Tabla de contenidos

Tabla de figuras

1 Introducción

En la actualidad con el uso más y más frecuente que se hace de la tecnología hay un tema que siempre preocupa al usuario a la hora de utilizar programas donde se tenga que verificar su identidad o utilizar su información, la seguridad. Todos conocemos el clásico sistema de identificación de usuario conformado por el típico usuario y contraseña, no obstante, este sistema no ofrece total seguridad de que realmente es el usuario quien está identificándose y no otro usuario que haya robado la información.

Para evitar este problema de seguridad que tantos problemas ha causado a algunos usuarios lo ideal sería un sistema de identificación que utilizase un elemento único de cada usuario, ahí es donde entran los documentos de identidad, siempre han sido utilizados para identificarnos y nada nos impide ahora utilizarlos en nuestras tecnologías.

Este trabajo pretende conseguir que el usuario pueda leer el MRZ de su documento de identidad o pasaporte y obtener toda la información de él, usando para ello la cámara de su móvil o la de su ordenador. Se ha trabajado con librerías y programas externos como son OpenCV, el cual se centra en el procesamiento de imágenes, Tesseract, encargado de la lectura y conversión de texto procedente de imágenes, Emscripten, encargado de compilar código C a formato webassembly para su posterior utilización en aplicaciones web, se han utilizado varios lenguajes de programación como son Kotlin y Java, utilizados en la aplicación Android, C++, utilizado en la librería que se ha desarrollado, web o html, utilizado en la aplicación web.

Se ha realizado el proyecto con la ayuda y tutorización de la empresa Veridas cuya prioridad es desarrollar programas de identificación de usuario seguras, rápidas y más potentes que las que actualmente se utilizan en la mayoría de los sistemas de identificación con el objetivo de hacerlo lo más accesible posible abarcando todas las plataformas posibles, el proyecto forma parte del trabajo que desarrolla el equipo enfocado a el uso que hará el cliente.

1.1 Motivación

La finalidad de este trabajo es la de conseguir una lectura del MRZ del documento de identidad o pasaporte del usuario lo más eficiente y rápida posible empleando las nuevas tecnologías de procesamiento y lectura de imágenes que se han desarrollado estos últimos años y así conseguir que el usuario tenga una experiencia agradable y satisfactoria empleando este tipo de tecnologías.

Otro de los motivos por los que se ha realizado este trabajo ha sido por la posibilidad que ofrece de trabajar con estas nuevas tecnologías que han experimentado grandes mejoras estos años, lo cual facilita su viabilidad para utilizarlas en dispositivos móviles y además utilizándolas en sistemas que requieren respuestas en tiempo real.

A lo largo de la carrera se han impartido asignaturas que tocaban el tema de procesamiento y lectura de imágenes, poder trabajar con ello y llevarlo a aplicaciones que lo utilicen en tiempo real haciendo uso de algo tan presente en la sociedad actual

como es el dispositivo móvil es una oportunidad para poner en práctica mis conocimientos al respecto y desarrollarlos aún más.

1.2 Objetivos

El objetivo principal de este trabajo es el de conseguir que el usuario pueda obtener la información del MRZ de su documento de identidad o pasaporte en tiempo real de la forma más rápida, eficiente y satisfactoria posible usando su dispositivo móvil o la web cam de su ordenador.

Los objetivos de este trabajo son:

1. Desarrollar una librería en C++ que se encarga del proceso de detección y lectura del MRZ, es decir, esta librería recibe una imagen y devuelve toda la información que obtiene del MRZ.
2. El diseño y desarrollo de una aplicación móvil para el sistema operativo Android que haga uso de la cámara del móvil y de la librería anteriormente mencionada para el reconocimiento del MRZ.
3. El diseño y desarrollo de una aplicación web que haga uso de la web cam del usuario y de la librería anteriormente mencionada para el reconocimiento del MRZ.
4. Emplear las nuevas tecnologías de procesamiento y lectura de imágenes en sistemas que requieran respuestas en tiempo real.
5. Facilitar la información del MRZ al usuario de la forma más eficaz, rápida y satisfactoria.
6. Familiarizar a los usuarios a estas nuevas tecnologías y mostrar todo su potencial y viabilidad, así como comodidad que supone utilizarlas.
7. Poner en práctica todo lo aprendido en la carrera en lo relacionado a procesamiento de imágenes.
8. Adquirir conocimientos de programación sobre aplicaciones móvil y web.

1.3 Impacto esperado

Se espera que el trabajo abra puertas a nuevos sistemas de identificación que usen estas nuevas tecnologías ya que proporcionan mayor seguridad al usuario en comparación a los sistemas clásicos que se suelen utilizar.

También se espera que mejore la experiencia del usuario al usar estos sistemas que emplean tecnologías de reconocimiento de imágenes y vean el potencial que hay detrás de estas tecnologías que han experimentado un gran crecimiento los últimos años.

Se espera asimismo que el sistema desarrollado se use en las aplicaciones de la empresa y así mejore la calidad de su servicio y garantice una mayor seguridad a todos sus usuarios.

Al ser un trabajo realizado con programas, sistemas y lenguajes con los que nunca he trabajado espero que me sirva para aprender mucho acerca de cómo funcionan, sus dificultades, sus pros y sus contras y la metodología que se ha de emplear para desarrollar un proyecto de este tamaño.

1.4 Metodología

Para la realización de este trabajo hubo que seguir unas pautas generales y otras más específicas:

Pautas generales:

1. Realizar la librería que procese y lea las imágenes con OpenCV y Tesseract.
2. Realizar la aplicación Android.
3. Realizar la aplicación web.

Pautas específicas:

1. Crear un archivo de configuración tessdata para que la lectura del MRZ sea lo más acertada posible.
2. Crear la librería utilizando CMake para que en caso de que se realicen cambios en la librería la compilación e inserción de cambios en ambas aplicaciones sea inmediata.
3. Establecer unas configuraciones de cámara que mejoren la calidad de la imagen para facilitar el reconocimiento del MRZ en ambas aplicaciones.
4. Establecer un sistema de cargado de librería lo más rápido y eficiente posible en ambas aplicaciones
5. Mostrar el resultado por pantalla de forma clara y concisa para que el usuario tenga claro cuando el reconocimiento ha sido satisfactorio y cuando no lo ha sido.
6. Que ambas aplicaciones y la librería funcionen de forma rápida y no haya tiempos de espera ni errores que ralenticen su funcionamiento.

1.5 Estructura

En este documento empezaré describiendo el proceso de creación de la librería que procesa y lee las imágenes, describiré también las técnicas empleadas para mejorar este proceso.

Asimismo, se detallará el proceso de creación de ambas aplicaciones Android y web y las técnicas empleadas para mejorar su funcionamiento y rapidez, se describirá además el diseño de estas y cómo se ha incluido la librería en ambas y conseguido que funcione de la mejor forma.

2 Contexto tecnológico

En este capítulo se va a hacer una observación a las aplicaciones existentes que realicen operaciones similares a las de la desarrollada en este proyecto.

Actualmente existen aplicaciones de lectura de MRZ para móvil que mediante la cámara sacan la información del documento de identidad o pasaporte del usuario, como por ejemplo ReadID:

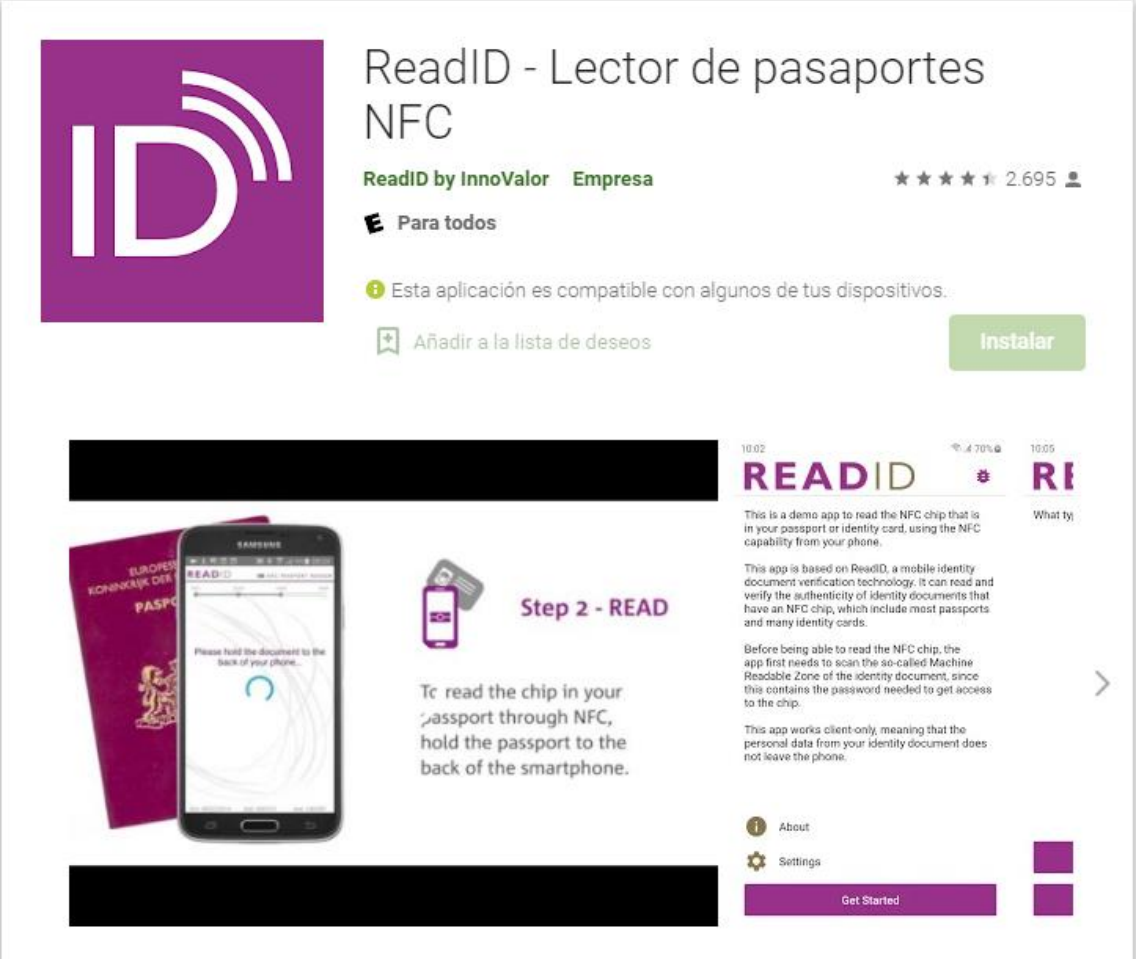

Figura 1. Captura de la aplicación ReadID.

En la imagen anterior (Figura 1) podemos ver una captura de la aplicación ReadID [1], esta lee y obtiene la información del documento de identidad usando el NFC del documento, el problema que presenta es que no todos los dispositivos móviles cuentan con lectura de NFC.

2.1 Propuesta

Este proyecto pretende desarrollar un programa de procesamiento y lectura de imágenes de forma rápida, eficaz y de fiabilidad para el usuario empleando avanzadas técnicas de procesamiento de imágenes y de lectura de estas.

Asimismo, se pretende que el sistema pueda ser actualizado, mejorado e integrado en otros sistemas con absoluta facilidad y compatibilidad.

3 Análisis del problema

Este trabajo ofrece la oportunidad de, utilizando avanzadas técnicas de reconocimiento de imágenes y procesamiento de estas, desarrollar un sistema eficiente y veloz que permita al usuario identificarse y obtener información de su documento de identidad de una forma más segura a los sistemas clásicos de identificación.

Los requisitos de hacer esta aplicación es precisamente que el trato con el usuario garantice rapidez y seguridad, es decir, que sea algo con lo que el usuario se familiarice y pueda emplearlo de forma rápida, es decir, los largos tiempos de espera, la sensación de que la aplicación no progresa etc quedan completamente descartados.

Además, se pretende hacer uso de estos nuevos métodos de reconocimiento de imágenes y procesamiento de estas cuyo rendimiento y eficacia se ha visto mejorada drásticamente en estos años, y que, con el paso del tiempo, si estas tecnologías siguen mejorando y creciendo su actualización en las aplicaciones Android y web sea inmediata y se puedan disfrutar los beneficios.

3.1 Análisis energético o de eficiencia algorítmica

La eficiencia energética y algorítmica han sido un punto importante sobre el que ha girado el proyecto entero, la librería se ha desarrollado con el estricto código necesario teniendo en cuenta su posterior inclusión en una aplicación Android y web, por tanto, se ha invertido gran parte del tiempo en verificar que el código no pierde más tiempo del necesario realizando operaciones ni emplea más memoria de la estrictamente necesaria.

Tanto en las aplicaciones Android y web se ha tenido en cuenta que el usuario ha de usar un dispositivo del que no tenemos control por tanto el guardado de imágenes se hace de la mejor forma y así evitar que se haga un uso excesivo de la batería o de la memoria del dispositivo.

Para la monitorización de estos aspectos se han utilizado las facilidades que proporcionan Android y web. En la siguiente captura (Figura 2) podemos ver el uso de memoria, cpu y energía de la aplicación Android, como se puede apreciar al principio de su ejecución se produce un pico debido a la carga de datos y energía que supone abrir la aplicación, pero se puede ver que una vez en ejecución se mantiene estable y no varía mucho.

A lo largo del desarrollo tanto de la librería C como de las aplicaciones Android y web se han ido controlando estos aspectos para que no dificultasen la ejecución de estas y como consecuencia acabase ralentizando el funcionamiento y obtención de un resultado.

Figura 2. Captura del uso de memoria, cpu y energía de la aplicación Android.

3.2 Análisis dentro del marco legal y ético

En este caso la información que se recopila y se enseña en la aplicación no se guarda ni se utiliza en una base de datos por tanto la información personal del usuario no peligra de ser utilizada para algo con lo que no estén de acuerdo o de ser utilizada por algún programa.

Si esta aplicación se usase junto con una base de datos de clientes u usuarios podría ser un eficiente sistema de asegurar con seguridad que el que realmente se esta identificando es el cliente, no obstante, existe el problema de que en el caso de que alguien robase una identificación de un cliente y la usase en la aplicación no habría forma de saber si realmente es él.

3.3 Análisis de riesgo

El factor de aceptación y satisfacción son muy importantes cuando desarrollamos una aplicación de este tipo enfocada a que lo utilice un usuario por tanto se ha realizado el proyecto teniéndolo muy en cuenta, tanto la interfaz como el proceso de detección y lectura del documento se han adaptado para que el usuario no tenga largas esperas y tenga la sensación de que se está trabajando mientras la aplicación está aún en el proceso de detección y lectura.

La librería se ha realizado pensando precisamente en que ha de ser incluida en otros sistemas Android y web o incluso si desea emplearse en algún otro entorno se tiene la seguridad de que está adaptada para ello.

Ambas aplicaciones están realizadas sin limitaciones con la posibilidad de ser parte de un proyecto más grande y de que se puedan utilizar bases de datos o cualquier otra herramienta sin que dificulte el trabajo de esta.

3.4 Identificación y análisis de soluciones posibles

Para la realización de este proyecto nos vimos limitados en unos aspectos, por ejemplo, para el procesamiento de imágenes y su lectura necesitábamos ambos OpenCV y Tesseract ya que el proyecto iba a abarcar ambas plataformas Android y web y era la mejor opción. No obstante, para la realización de la librería había cierta libertad y se me permitió realizarla con el programa con el que más estuviese familiarizado, en mi caso nunca había trabajado con uno de estos programas y me atreví más con Eclipse, pero cabía la posibilidad de realizarlo con NetBeans entre otros.

A la hora de desarrollar la aplicación web y Android usamos respectivamente Visual Studio Code empleando el lenguaje Typescript y Android Studio empleando el lenguaje Kotlin, tal como requería la empresa ya que aseguraba mejor compatibilidad y funcionamiento de las aplicaciones.

En cuanto al proceso de obtener el texto resultado, es un proceso bastante inalterable y único por lo que no había mucho margen para obtener diferentes soluciones con un mismo resultado. Este proceso consta de:

1. Redimensionar la imagen.
2. Conversión a gris
3. Eliminación de bordes.
4. Eliminar ruido y elementos innecesarios.
5. Buscar los contornos y ordenarnos según la relevancia.
6. Detección del MRZ.
7. Recortar la zona del MRZ y mejorar la calidad de la imagen.
8. Hacer la lectura de este y obtener el texto.

3.5 Solución propuesta

Esta consiste en primero realizar una librería en código C++ que haga el procesamiento y reconocimiento de las imágenes que se le pasan, realizando una serie de operaciones sobre ella para mejorar su calidad y obtener solo la zona de interés.

Una vez obtenido esto lo siguiente es desarrollar la aplicación Android, en Android Studio, hacer una interfaz agradable al usuario y que abra la cámara y pase continuamente imágenes a la librería para que, una vez se detecte el documento de identidad, se obtenga el texto resultado y lo saque por pantalla desglosando toda la información.

Finalmente se desarrolla la aplicación web que haga uso de la librería para obtener el MRZ del mismo modo que la aplicación Android.

3.6 Plan de trabajo

El plan de trabajo consistía en primero desarrollar la librería en C++ y luego realizar el desarrollo de las aplicaciones Android y web como se ha descrito anteriormente, para ello se han dedicado al principio 4 horas diarias, pero al ver que no eran suficientes y el proyecto avanzaba muy lento se ha propuesto realizar 6 o más horas diarias.

3.7 Presupuesto

Para este trabajo ha sido necesario tener un ordenador, una web cam que enfoque objetos cercanos y un dispositivo Android. No se ha empleado software de pago. Contaba con todos los elementos mencionados excepto la webcam, que ha costado sobre unos 40 euros.

4 Diseño de la solución

4.1 Arquitectura del sistema

El proyecto se divide en 3 partes:

1. La librería C++ que se encarga procesar la imagen que se le pasa y leerla devolviendo un texto indicando la información del documento de identidad, esta hace uso de OpenCV y de Tesseract para procesar las imágenes y hacer la lectura de esta respectivamente. Además de obtener el nivel de confianza para cada carácter que se ha leído.

2. El programa Android, desarrollado en Android Studio, que usa la cámara del móvil para enfocar el documento de identidad y la librería desarrollada, junto con las librerías de OpenCV y Tesseract adaptadas para su uso en Android, para obtener el texto resultado, luego, una vez obtenido se saca la información por pantalla de forma clara para que el usuario pueda verla.

3. El programa web, desarrollado en Visual Studio Code utilizando el lenguaje Typescript, que usa la cámara web cam para enfocar el documento de identidad y la librería desarrollada, junto con las librerías de OpenCV y Tesseract adaptadas para su uso en WebAssembly, para obtener el texto resultado, luego saca la información por pantalla de forma clara para que el usuario pueda verla.

4.2 Tecnología utilizada

Ubuntu: utilizo una máquina virtual que simula un sistema operativo Ubuntu donde se realiza la mayor parte del trabajo, como es por ejemplo el desarrollo de la librería, la aplicación Android, la importación de las librerías OpenCV y Tesseract y su adaptación a las respectivas aplicaciones Android y web.

Eclipse: se utiliza para la realización del proyecto que posteriormente se convertirá en una librería para ser utilizada en las aplicaciones Android y web. El proyecto creado se ha programado en C++.

OpenCV: se emplea en la librería y es el encargado de realizar las transformaciones de imágenes y sus modificaciones necesarias además de detectar la zona de interés.

Tesseract: utilizado también en la librería, convierte el texto encontrado en la zona de interés en texto manejable para poder sacar los resultados necesarios del mismo (nombres, apellidos, etc)

CMake: se utiliza para compilar el proyecto resultado de Eclipse en una librería con el formato adecuado para la aplicación Android.

GNU Image: utilizado para recortar y modificar las imágenes para posteriormente poder hacer un reconocimiento entrenado de ellas y poder hacer el fichero tessdata.

QT-Box Editor: se emplea para crear el archivo tessdata que posteriormente utilizará Tesseract para facilitar y mejorar el reconocimiento del texto que se va a leer en las imágenes que se le pasan a la librería.

Android Studio: empleada para realizar la aplicación para el dispositivo Android en el que se utiliza la librería creada con Eclipse y CMake.

Windows10: se emplea para la realización de la aplicación Web ya que la máquina virtual con Ubuntu detecta la webcam, pero no permite un uso adecuado de ella.

Visual Studio Code: entorno de desarrollo utilizado para la realización de la aplicación web.

Emscripten: utilizado para la compilación de las librerías para poder utilizarlas en la aplicación web.

Kotlin: lenguaje de programación empleado por Android Studio para la realización de la aplicación Android.

Typescript: lenguaje de programación empleado por Visual Studio Code para la realización de la aplicación web.

C++: lenguaje de programación empleado en Eclipse para el desarrollo de la librería.

WebAssembly: lenguaje utilizado por el fichero generado por Emscripten al realizar la compilación de la librería, posteriormente usado en el proyecto web.

VM Virtual Box: programa utilizado para crear un entorno Ubuntu sobre el que desarrollar la

HTML: uno de los lenguajes usado para la realización de la aplicación web y la organización de los elementos por pantalla.

5 Librería C++

La librería C++ se ha desarrollado en Eclipse, utilizando el lenguaje C++. Se han utilizado las librerías de OpenCV y Tesseract para el proceso de detección y lectura de imágenes. Además, se ha utilizado CMake para compilarlo todo en una librería utilizable en los proyectos Android y web.

5.1 Programas utilizados por la librería

5.1.1 Eclipse

Para el desarrollo de la librería C++ se ha empleado Eclipse [2], Eclipse es una plataforma de desarrollo software compuesta por un conjunto de herramientas de programación de código abierto, permite programar en diferentes entornos y lenguajes entre ellos podemos encontrar Java y C++.

5.1.2 C++

El lenguaje de programación utilizado para desarrollar la librería es C++ [3], C++ es un lenguaje que proviene de la extensión del lenguaje C para que pudiese manipular objetos, es un lenguaje con muchos años y muchos usuarios activos además de recibir actualizaciones constantes, lo cual lo convierte en un lenguaje muy potente utilizado muy a menudo para programar en alto nivel.

5.1.3 CMake

El proyecto resultante se ha convertido en una librería utilizando CMake [4], CMake es una herramienta multiplataforma de generación o automatización de código, permite definir y administrar compilaciones de código principalmente para C++, mediante una serie de definiciones de cómo se construye el proyecto te permite crear librerías y ejecutables. El comportamiento de la compilación se define en el archivo CMakeLists.txt, uno para cada directorio de código fuente, en él se pueden definir las librerías y ejecutables anteriormente mencionados que se quieren generar y también las rutas de los archivos, definición de variables y localización de librerías dependientes.

5.1.4 OpenCV

OpenCV [5] es una librería libre de visión artificial y machine learning que vio la luz a principios de 1999, esta provee una infraestructura para aplicaciones que utilicen visión artificial, al ser una librería libre permite utilizar y modificar el código, tiene una comunidad enorme de más de 47000 usuarios lo cual hace que esté en constante crecimiento y desarrollo. Además, está escrito en C++ lo cual la hace muy manejable.

Algunos de los algoritmos que contiene permiten identificar caras, objetos, clasificar acciones humanas en video, hacer seguimientos de movimientos de objetos, encontrar imágenes similares, seguir movimientos de ojos, etc.

5.1.5 Tesseract

Tesseract [6] es un software libre de reconocimiento óptico de caracteres, es uno de los de código abierto más precisos disponibles y se hizo open-source en 2005. Puede reconocer más de 100 lenguajes y puede entrenarse para reconocer más lenguajes. Permite su uso utilizando cualquier tipografía siempre que se le pasen las configuraciones y entrenamiento adecuado.

5.2 Clases que conforman la librería

La librería C++ consta de distintas clases que realizan todo este proceso de mejoramiento de la imagen y lectura separadas según su función específica, las voy a describir en orden de trabajo desde que se le pasa la imagen hasta que se ofrece el resultado.

5.2.1 Clase MrzReader o clase principal

Esta es la clase principal a la que se le pasa la imagen y llamará a las otras clases para realizar el proceso de detección y lectura, transformado de la imagen.

Empieza haciendo una conversión a blanco y negro de la imagen si es necesario, acto seguido realiza un escalado de la imagen y se le pasa a otra clase que se encargará de realizar el proceso de detección en el que buscará la zona en la que se encuentra el MRZ del documento de identidad o pasaporte. Una vez obtenido el resultado de la detección, si este no encuentra nada y devuelve un resultado no válido directamente se escriben los resultados y se finaliza la ejecución, no obstante, si el resultado es válido se transfiere la zona de la imagen que ha encontrado la clase de detección a la clase de lectura de la imagen.

Antes de pasar este resultado a la clase de lectura se realiza un escalado de imagen que depende de si el MRZ es el de un pasaporte o el de un DNI, este escalado óptimo para mejorar la lectura de la imagen se ha obtenido mediante pruebas con diferentes ejemplos o muestras de imágenes, finalmente, en caso de que la lectura sea errónea se devuelve la respuesta correspondiente y en caso de que sea acertada se devuelven los resultados obtenidos.

5.2.2 Clase Imutils o clase de utilidades

Esta clase es la primera en ser llamada por la clase principal, consta de dos funciones que realizan las transformaciones sobre las imágenes más básicas como rotar la imagen o redimensionarla. Para ello utiliza funciones de OpenCV que facilitan este proceso.

5.2.3 Clase DetectRoi o clase de detección del MRZ de la imagen

Segunda clase en ser llamada es la encargada de detectar la zona de interés, es decir, la zona donde se encuentra el MRZ del documento de identidad. Empieza creando dos kernels, uno de forma cuadrada y otro rectangular, un kernel consiste en definir un área, esta área se utiliza posteriormente para aplicar filtros y modificar los pixeles que los contienen, sobre la imagen pasada utilizando OpenCV, concretamente empieza con un Gaussian Blur [7-10] sobre la imagen convertida a blanco y negro, el cual suaviza la

imagen y permite eliminar los bordes o contornos que dificultan el reconocimiento. La Figura 3 muestra un ejemplo práctico de su funcionamiento. En la imagen de la derecha podemos apreciar el efecto de aplicar un filtro gaussiano sobre la imagen.

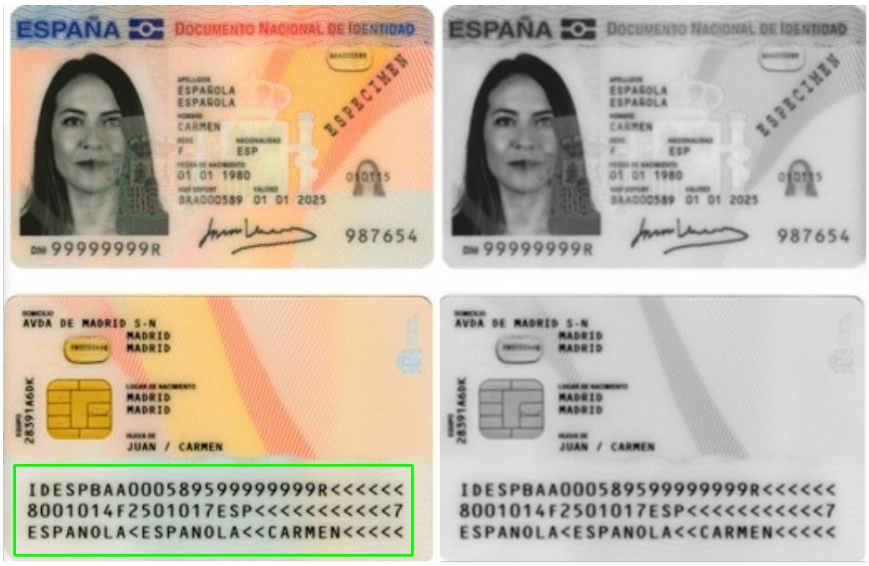

Figura 3. Filtro Gaussian Blur.

Suavizar la imagen permite eliminar todo lo que pueda molestar a la hora de encontrar contornos relevantes para identificar el MRZ, la intensidad del suavizado depende del tamaño del kernel utilizado. También permite quitar todo el ruido que se genera cuando capturas una imagen mediante un dispositivo externo como puede ser una cámara de un móvil o una webcam.

Esto se consigue dando mayor valor de peso al píxel más cercano al centro del kernel que se utiliza y haciendo una media de valores que posteriormente se aplica a todos los pixeles para conseguir ese efecto de difuminado al normalizarse los valores de estos.

También aplicamos un filtro Black Hat [11-13] sobre la imagen después de aplicar el Gaussian Blur. El filtro Black Hat se encarga de aplicar operaciones sobre la imagen habiendo proporcionado un kernel, este destaca los objetos negros sobre fondo blanco que encuentra en la imagen, tal como se aprecia en la Figura 4.

En la imagen de la derecha de la Figura 4 vemos el resultado de aplicar Black Hat sobre la primera imagen. Podemos ver como se destacan en blanco los objetos que en la primera imagen se encontraban rodeados mayoritariamente de blanco. En concreto, sobre la imagen de ejemplo proporcionada vemos como el MRZ es la zona que más destacada aparece y eso es debido a que es la superficie que más rodeada de blanco está en comparación con el resto, nos fijamos también por ejemplo que algunos campos han aparecido casi ilegibles, esto nos ayuda a eliminar distracciones y encontrar la zona del MRZ.

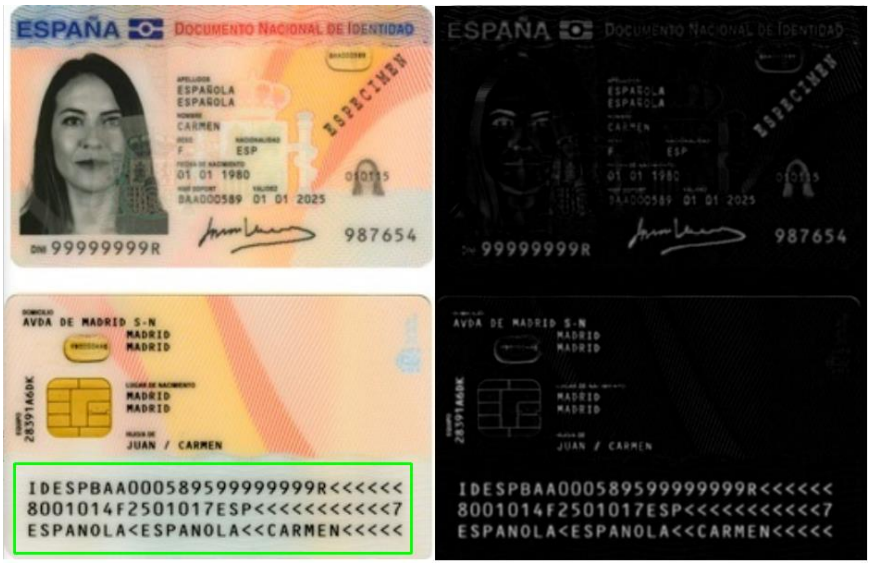

Figura 4. Filtro Black Hat.

La utilidad de este filtro llevado al documento de identidad que vamos a procesar se resume en que al aplicarlo conseguimos que en la zona del MRZ aparezcan todos los contornos de las letras destacados y por tanto esto implica que mediante este filtro se consigue eliminar muchos elementos del documento de identidad que no sea el MRZ, como por ejemplo alguna línea o dibujo, algún carácter u otro tipo de información irrelevante para el reconocimiento del mismo… y así se consigue que poco a poco se vayan eliminando elementos externos para luego asegurarnos de que luego al hacer la búsqueda de contornos persista el MRZ y además se le dé más prioridad a la hora de buscarlo.

La intensidad de este también viene determinado por el kernel por tanto ha sido necesario asegurarse de que no se pierda información valiosa del MRZ al aplicarse de una forma demasiado agresiva o que su efecto fuse irrelevante al aplicarse de una forma demasiado suave ya que el objetivo es eliminar contornos innecesarios y que persistan los del MRZ.

Finalmente aplicamos el filtro Sobel [14, 15] sobre la imagen, que obtiene los contornos de la imagen contemplando los cambios bruscos de intensidad en los pixeles de la imagen. Este filtro permite obtener una imagen en la que se le quita importancia al fondo y se le da importancia a los bordes. Podemos ver su efecto en el ejemplo de la Figura 5.

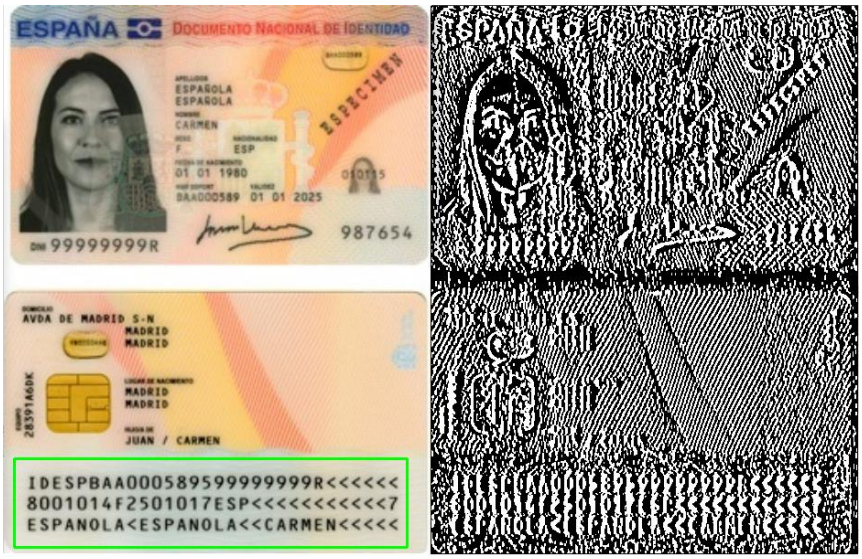

Figura 5. Filtro Sobel.

En la imagen de la derecha de la Figura 5 podemos ver el resultado de aplicar una serie de transformaciones utilizando Sobel y podemos ver como se destacan los bordes y se quita la importancia del fondo. Remarcar los contornos de la imagen nos permite mantener la forma de los caracteres de los MRZ y que por tanto no se deformen o se pierda información de este.

El filtro Sobel puede dar resultados distintos según cómo se aplique, ya que se puede aplicar solo para los contornos del eje X o Y o ambos dependiendo de a qué contornos se les quiera dar prioridad y además también depende del kernel que se le proporcione.

Este filtro combina suavizado gaussiano e inversión de la imagen, por eso podemos ver en la imagen resultado del ejemplo como se pierden los colores y los bordes marcados están suavizados.

Esto aplicado al proyecto nos beneficia en que nos va a proporcionar una imagen en la que se destaca más la figura de los caracteres que conforman el MRZ y tras posteriores modificaciones permitirá una mejor lectura a Tesseract.

A continuación, se eliminan los valores negativos en la matriz de la imagen, estos valores son innecesarios y se les pone un valor neutro para que luego en la detección de bordes no se tengan en consideración. Seguidamente se sacan los valores mínimo y máximo de la matriz de valores de la imagen y hace una media para aplicarla sobre todos ellos.

Lo siguiente es realizar una conversión del tipo de la imagen a CV_8UC1, es decir, una imagen de 8 bits sin signo en un solo canal, utilizado principalmente en imágenes a escala de grises. Y aplicar más filtros sobre la imagen, empezando con un filtro Close [16], el cual se encarga de eliminar el ruido en la imagen de los contornos que contiene. Podemos apreciar su función en la imagen de la Figura 6. En la figura se muestra el efecto de aplicar el filtro Sobel a la imagen original y podemos apreciar cómo se ha eliminado el ruido que contiene.

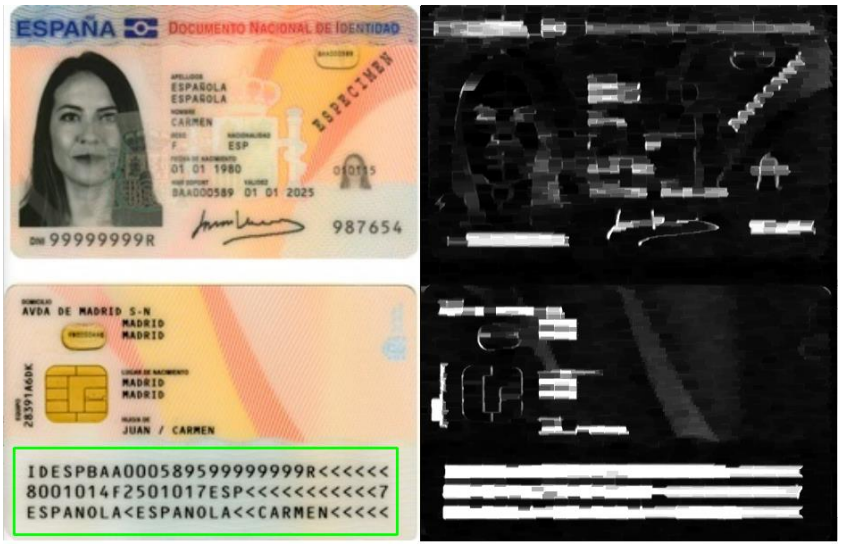

Figura 6. Filtro Close.

El siguiente filtro que se aplica sobre la imagen es el Threshold utilizando las técnicas de Otsu [17-19]. Esta técnica se caracteriza por determinar un valor de umbral global óptimo a partir del histograma de la imagen, por lo que este filtro se encarga de eliminar las escalas de grises remarcando solo los valores necesarios y evitando así que se omitan posibles valores que pueden pasar desapercibidos (véase Figura 7). Podemos apreciar en la figura como en la segunda imagen se elimina la escala de grises y aparecen todos los valores marcados por igual.

Figura 7. Filtro Threshold con técnicas Otsu.

Seguidamente se aplican dos últimos filtros sobre la imagen, otro filtro Close y un filtro Erode [20, 21], el cual desgasta los bordes de los contornos de la imagen y en este caso en concreto sirve para remarcar las figuras de los contornos y facilitar así su lectura a la clase que se encarga de la lectura del MRZ. En la Figura 8 podemos apreciar su funcionamiento. En la figura podemos ver en la segunda imagen el efecto de aplicar el

filtro Erode sobre la primera, como se puede apreciar se elimina parte de la figura de la imagen y obtenemos una imagen más marcada que la segunda, podríamos decir que obtenemos una imagen menos redondeada.

Figura 8. Filtro Erode.

A continuación, se eliminan los valores innecesarios de la matriz de la imagen otra vez y ya podemos finalmente encontrar los contornos de la matriz con la función FindContours de OpenCV y ordenarlos de mayor a menor.

En caso de que se hayan encontrado contornos y se hayan podido ordenar de mayor a menor, cogemos el primero ya que sabemos que el contorno del MRZ siempre va a salir el primero en la ordenación de mayor a menos por ser la zona que más contrasta del documento de identidad, luego lo recortamos de la imagen para devolverlo finalmente a la clase principal y seguir con el procedimiento.

5.2.4 Clase ReadRoi o clase de lectura de la imagen

Última clase en ser llamada, a esta clase se le pasa la imagen con el MRZ obtenido en la clase de detección y una ruta o camino al archivo de configuración de Tesseract llamado también tessdata que se va a utilizar para la lectura de la imagen.

La imagen la recibe Tesseract y se ejecuta su función de reconocimiento, la cual devuelve el texto leído y lo convierto a string para poder manejarlo mejor y realizar operaciones sobre él.

Para poder realizar este reconocimiento Tesseract necesita que se le pase, a parte de una imagen, un fichero con datos entrenados que se tiene que crear acorde con las imágenes que va a recibir el mismo. Para ello, ha sido necesario primero recopilar un conjunto de imágenes ya procesadas por OpenCV, luego, descargar la fuente que se va a utilizar para realizar el reconocimiento, en este caso se llama OCR. Una vez recopilada esta información ejecuto un script que genera las cajas de reconocimiento y los caracteres reconocidos por Tesseract en cada una de las imágenes de entrenamiento y, mediante la aplicación QTboxeditor puedo abrir estas imágenes con

sus cajas y caracteres generados para poder modificar los errores que haya podido tener Tesseract a la hora de leerlas.

Ha sido necesario ir muestra por muestra asegurándome de que no se haya equivocado haciendo la lectura y, en caso de que lo haya hecho, corregir el fallo y guardar los cambios. Una vez todo corregido y guardado con estos datos se genera el fichero traineddata el cual ya se lo puedo proporcionar a Tesseract en el reconocimiento que hace en la librería para que haga lecturas más acertadas.

En la Figura 9 podemos ver el entorno de edición de cajas y caracteres reconocidos por Tesseract en QTBoxEditor. Las cajas aparecen marcadas por las líneas verdes y en el lateral izquierdo podemos ver el carácter asociado a caja una de las cajas, este programa nos permite modificar el tamaño de las cajas para que se ajusten al carácter y, además, modificar el carácter leído en caso de que sea erróneo.

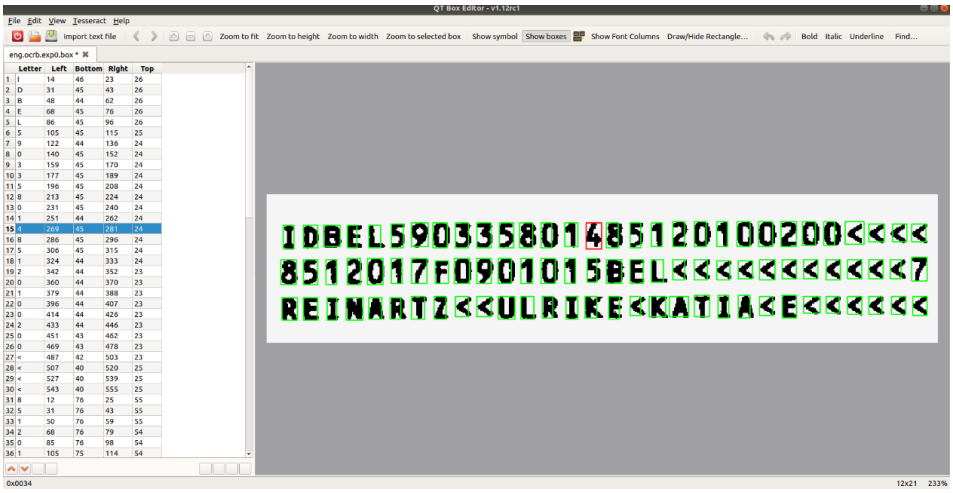

Figura 9. Imagen del entorno QTBoxEditor.

Todos los cambios que se efectúen ayudan al entrenamiento de Tesseract para posteriormente hacer un reconocimiento más efectivo, se han utilizado 13 muestras, pero cuantas más muestras se utilicen mayor será la efectividad de este a la hora de reconocer caracteres.

El programa QTBoxEditor proporciona herramientas muy visuales y sencillas a la hora de hacer la modificación de los datos leídos por Tesseract lo cual facilita mucho la modificación de estos y la creación de datos entrenados.

En la Figura 10 podemos apreciar por pantalla los caracteres que ha leído Tesseract haciendo el reconocimiento de la imagen incluyendo las respectivas cajas.

Además de obtener el texto, mediante un bucle se obtiene el vector de valores de confianza obtenidos para cada carácter leído por Tesseract en su lectura de la imagen.

A continuación, se evalúan un conjunto de condiciones sobre el texto resultado obtenido para clasificarlo y obtener sus resultados, estas condiciones que se evalúan son las siguientes:

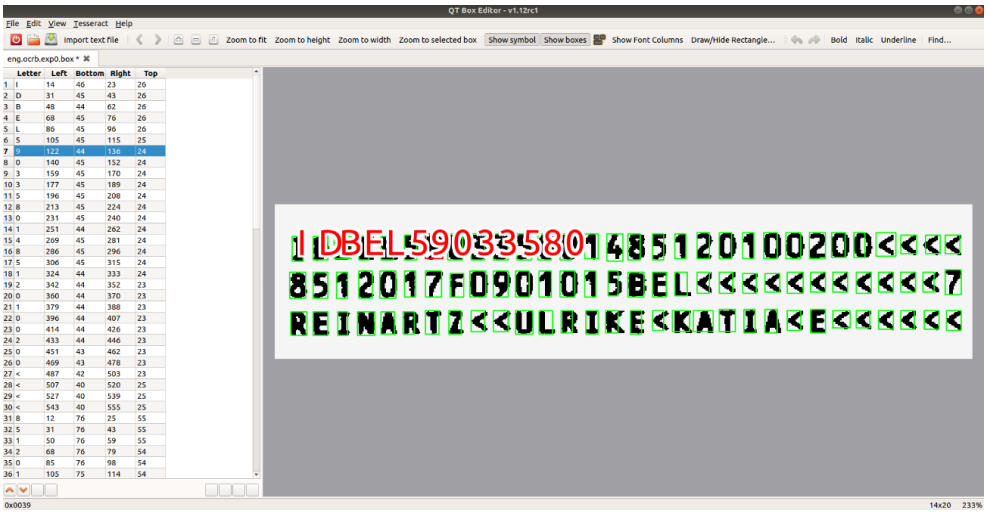

Figura 10. Captura del programa QTBoxEditor enseñando los carácteres.

- Tamaño del texto:

Si el texto resultado contiene 3 líneas sabemos que se trata de un documento de identidad, si este contiene 2 líneas sabemos que se trata de un pasaporte, si no cumple ninguna de las dos condiciones sabemos que el texto leído no es válido y por tanto finalizamos aquí y devolvemos el resultado indicando que ha sido una lectura no válida.

- Número de caracteres leídos:

Si el texto resultado no contiene al menos 93 caracteres (los que contiene el documento de identidad contando los saltos de línea) o 90 caracteres (los que contiene el pasaporte contando el salto de línea) si no contiene al menos 93 o 90 caracteres sabemos entonces que el texto leído es erróneo y que por tanto no nos sirve para obtener información de él, en caso de que sea correcto seguimos con la evaluación de condiciones y en caso de que no lo sea terminamos la evaluación de condiciones y devolvemos el resultado.

-Los dígitos de verificación de información correcta para las fechas de nacimiento y expiración y para el ID:

En los MRZ hay un dígito que sigue a los campos de fecha de nacimiento, expiración y número de ID, este dígito se calcula con los anteriores dígitos correspondientes a estos campos mencionados y mediante una función adecuada se puede hacer el cálculo y verificar si coincide. En caso de que coincida sabemos que la lectura se ha hecho correctamente, en caso de que no coincida sabemos también que se ha realizado una lectura incorrecta y por tanto finalizaríamos aquí y devolveríamos el resultado correspondiente. A continuación, adjunto un ejemplo para hacer una demostración de cómo se calcula el valor de este dígito para entender mejor su funcionamiento:

Example 1 — Application of check digit to date field

Using 27 July 1952 as an example, with the date in numeric form, the calculation will be:

	Date:	5	2	0	7	2	7	
	Weighting:	7	3	1	7	3	1	
Step 1 (multiplication)	Products:	35	6	0	49	6	7	
Step 2 (sum of products)		35	+ 6	+ 0	+ 49	+ 6	+ 7	= 103
Step 3 (division by modulus)		103 / 10 = 10, remainder 3						

Step 4. Check digit is the remainder, 3. The date and its check digit shall consequently be written as 5207273.

Example 2 — Application of check digit to document number field

Using the number AB2134 as an example for coding a 9-character, fixed-length field (e.g. passport number), the calculation will be:

Sample data element:	A	B	2	1	3	4	<	<	<
Assigned numeric values:	10	11	2	1	3	4	0	0	0
Weighting:	7	3	1	7	3	1	7	3	1
Step 1 (multiplication) Products:	70	33	2	7	9	4	0	0	0
Step 2 (sum of products)	70	+ 33	+ 2	+ 7	+ 9	+ 4	+ 0	+ 0	+ 0 = 125
Step 3 (division by modulus)	125 / 10 = 12, remainder 5								

Step 4. Check digit is the remainder, 5. The number and its check digit shall consequently be written as AB2134<<<5.

Figura 11. Ejemplos de evaluación del dígito de verificación.

En la Figura 11 podemos ver dos ejemplos, uno de obtención del dígito de verificación de la fecha de nacimiento (en la primera imagen) que consta únicamente de dígitos numéricos y el otro de obtención del dígito de verificación del número de ID del documento de identidad (segunda imagen) que además de constar de dígitos numéricos contiene caracteres y símbolos [22].

Como podemos ver en el ejemplo para obtener este dígito hay que realizar una serie de pasos que consisten en:

1. Asignar valores numéricos a los caracteres que contiene el elemento que estamos deseando evaluar, si son números no hace falta conversión, pero en caso de que haya letras se asigna un número siendo la A el 10, la B el 11, la C el 12 y así recorriendo todo el alfabeto, en caso de que sea un símbolo < se le asigna el valor 0.

2. Una vez asignados los valores se multiplica cada uno por un valor, empezando por el primero, lo multiplicamos por 7, el segundo se multiplica por 3 y el tercero por 1, si hay más elementos se repite la secuencia de multiplicar por 7, luego por

3 y finalmente multiplicamos por 1. Realizamos esto con todos los elementos del campo.

3. Seguidamente sumamos todos los valores obtenidos en las multiplicaciones.

4. Dividimos el resultado de la suma de los valores entre 10 y cogemos el resto.

5. El resto obtenido es el dígito de verificación el cual aparecerá siempre al final de cada uno de estos campos.

Por tanto, conociendo esta información se han realizado 2 funciones, una que evalué el dígito de verificación de las fechas de nacimiento y expiración y otra función que evalúe el dígito de verificación del id del documento de identidad.

Para ello realizan las conversiones numéricas correspondientes y las multiplicaciones y divisiones necesarias sobre los elementos que se le han pasado, devolviendo un resultado que indique si el campo en cuestión es correcto o es erróneo, en caso de que sea erróneo se termina con la evaluación y obtención de resultados del MRZ y se indica que la lectura no ha sido correcta.

Tras evaluar estas condiciones en caso de que no haya habido ningún problema o indicio de que la lectura es incorrecta se procede a la recopilación de los datos que podemos obtener del MRZ y su almacenamiento en una clase resultado.

Se empieza recogiendo la nacionalidad, el ID, el DNI (en caso de que lo haya), la fecha de nacimiento, la fecha de expiración y el sexo del usuario, todo ello mediante funciones de C++ que permiten la obtención de substrings dentro de un string conociendo la posición del carácter inicial y el número de caracteres que se quieren obtener.

Una vez obtenida toda esta información ahora solo resta obtener los nombres y apellidos del usuario, esta parte es más flexible ya que el usuario puede tener o bien un nombre o dos, puede que por algunos motivos solo aparezca un apellido. Hay que tener todo eso en cuenta y sabiendo el formato que se emplea en el MRZ es fácil de desglosar esta información. La Figura 12 muestra un ejemplo de MRZ para entender cómo se refleja esta información en el MRZ

Como podemos ver en este ejemplo de documento de identidad, fijándonos solo en el apartado de los apellidos y nombres vemos que los apellidos aparecen primero y vienen separados por el símbolo < y que a continuación, los nombres aparecen separados por los símbolos <<. Teniendo esto en cuenta se ha establecido un conjunto de condiciones en los que se encuentran en la línea los símbolos << y <, luego, utilizando las funciones de C++ de recortar substrings en un string se extraen los correspondientes nombres y apellidos y se añaden a la clase de resultados para luego devolverlos.

Finalmente, con los resultados guardados en la clase de resultados y con el vector de valores de confianza guardados también, se devuelve a la clase principal este objeto de clase resultado. En caso de que la lectura sea errónea se devuelve un resultado indicando que la lectura no ha sido satisfactoria.

Figura 12. Foto ejemplo de la parte trasera de un DNI.

5.2.5 Clase Confs o clase de configuraciones establecidas

Clase que establece las configuraciones necesarias para la detección de la imagen, en ella podemos encontrar la ruta o camino para encontrar el archivo de configuración de Tesseract o tessdata, un indicador para conocer si el formato de la imagen está en formato YUV y la orientación con la que se ha obtenido la imagen en cuestión.

Podemos encontrar un objeto de esta clase en la clase principal, ya que junto con la imagen que se le pasa a esta también recibe un objeto de esta clase para conocer las transformaciones necesarias a realizar sobre la imagen antes de empezar con el proceso de detección del MRZ.

Contiene las funciones necesarias para consultar estos valores y para modificarlos en caso de que sea necesario.

5.2.6 Clase Confidence o clase del vector de confianza

Esta clase contiene dos elementos principales, el valor del carácter en cuestión y el valor de la confianza de dicho carácter.

Es la que se utiliza en la clase de lectura del MRZ para construir un vector en el que a cada carácter se le asigna su valor de confianza.

Una vez el vector está construido este se pasa a la clase resultados para que lo almacene y lo devuelva en el resultado final.

Cuenta con las funciones necesarias para consultar estos valores y para construir un objeto que cuente con ambos valores.

5.2.7 Clase Results o clase resultados

En esta clase se van guardando los resultados de la lectura del MRZ en la clase de lectura, concretamente empieza guardándose el vector de valores de confianza y seguidamente se van guardando el texto entero del MRZ, la nacionalidad, la fecha de nacimiento y expiración, el sexo, el DNI, el ID del documento, los nombres y apellidos. En caso de que solo tenga un nombre el segundo campo del nombre aparecería vacío y no afecta al resultado final, y lo mismo para los apellidos.

Si por lo que sea se da el caso que el resultado de la lectura es erróneo no se guardan estos valores y simplemente se devuelve un texto indicando que la lectura no es correcta.

La clase final se encarga de devolver este objeto de la clase resultados con todos los resultados obtenidos en la clase de lectura.

5.3 Entrenando Tesseract.

Anteriormente se ha explicado el proceso de creación de un fichero de configuración o entrenamiento para Tesseract llamado tranieddata pero no se ha entrado mucho en detalle en los factores que facilitan este proceso o garantizan mejores resultados. En este apartado se explicará todo lo que se ha tenido en cuenta a la hora de crear este fichero para que su uso sacase los mejores resultados posibles.

Los factores que más en cuenta se ha tenido a la hora de seleccionar imágenes para que formen parte del entrenamiento han sido los siguientes:

1. Resolución de la imagen: la resolución de las imágenes seleccionadas ha sido examinada con detalle ya que las imágenes con peor resolución que pasan por el procesamiento de OpenCV acaban con letras prácticamente ilegibles y añadirlas a la muestra de entrenamiento podría causar conflicto y reconocimiento de caracteres incorrectos en las imágenes con mayor resulción. Además, se ha tenido en cuenta que se iba a trabajar con dispositivos móviles y webcams por lo que se ha tenido en cuenta un mínimo de resolución aceptable.
2. Nacionalidad de los documentos: cada nacionalidad dispone de un tipo de documento diferente con sus colores y elementos distribuidos de forma diferente, a pesar de que la zona del MRZ no cambia ha sido necesario realizar pruebas con documentos de distintas nacionalidades para asegurarse de que no intervengan elementos externos. Por ejemplo, para el reconocimiento de los pasaportes americanos ha dado la casualidad de que había un dibujo que era incluido en la zona del MRZ y Tesseract trataba de leerlo.
3. Variedad de carácteres: la mayoría de las imágenes disponibles disponen de poca variedad de letras en el MRZ, por ejemplo, un documento de muestra con un DNI con el valor 99999999R va a crear un archivo de entrenamiento peor que un DNI que tenga el valor de 12398765K, y lo mismo para los otros campos, esto es debido a que al contener mayor variedad de letras y números estamos entrenando y preparando Tesseract a saber cómo reaccionar al encontrar estos caracteres en un entorno real, ya que los documentos de identidad reales contienen gran variedad de estos caracteres.

4. Variedad de pasaportes y documentos de identidad: preparar a Tesseract bien para que haga un reconocimiento eficiente de los MRZ de los documentos de identidad no garantiza que vaya a tener la misma eficiencia a la hora de leer los pasaportes. Se han tenido que añadir la misma variedad de documentos de identidad que de pasaportes para asegurarse de que la lectura en ambos sea correcta.
5. Documentos actuales: se ha tenido en cuenta también que los documentos están en cambio constante, el DNI español por ejemplo ha ido cambiando con el paso del tiempo y sus colores y distribución de elementos cambian también, esto en principio no afecta al reconocimiento del MRZ ya que sus valores no cambian de sitio y siempre está en la misma ubicación, no obstante observé que es importante añadir documentos lo más actuales posibles a las muestras de entrenamiento ya que siempre hay algún elemento que por muy discreto que sea, por ejemplo el color del fondo del documento, acaba teniendo su repercusión en el procesamiento que hace OpenCV y posteriormente en la lectura de Tesseract.
6. Confusión de caracteres: un factor muy importante que ha causado muchos problemas en el fichero de entrenamiento. Tesseract tiene muchos problemas para reconocer ciertos caracteres muy parecidos si no se le entrena bien, por ejemplo, observé que al principio confundía mucho el '0' y la 'O' o el 'D', el '<' con la 'C' o la 'c' y el '8' con la 'B' entre otros caracteres, fue necesario añadir más muestras de entrenamiento que tuviesen más de estos caracteres para que aprendiese bien a diferenciarlos, ya que con las muestras que tenía al principio no se encontraban muchos de estos y es lo que causo que luego no consiguiese leerlos bien. Este factor en concreto debió tenerse bien en cuenta ya que al desarrollar la librería y hacer la verificación de fechas o de ID los descartaba a pesar de estar correctos precisamente por confundir un '0' con una 'O' en alguna fecha de nacimiento o expiración y esto marcó luego la diferencia a la hora de hacer la aplicación móvil entre una lectura rápida y una en la que se tardaba un poco más en conseguir el resultado deseado.

La mayoría de estos factores no se tuvieron en cuenta desde el principio, se ha ido descubriendo poco a poco que tenían su impacto en el resultado de lectura y que en su conjunto marcan una gran diferencia en el tiempo que se tarda en conseguir un buen resultado de lectura de MRZ. Por tanto, se podría decir que seguramente existen más factores que si se tuviesen en cuenta harían que el MRZ se lea de una mejor forma y que por tanto deberían investigarse, como por ejemplo el color de la luz con el que se enfoca el MRZ, si hay presencia de elementos externos como una mano, un dedo, si hay suciedad presente en el documento o sufre de ralladuras.

Para este proyecto estos han sido los factores que se han descubierto y prestado atención a la hora de crear el traineddata y así se ha asegurado una lectura eficiente y correcta en el entorno correspondiente en el que se ha usado.

6 Aplicación Android

Android [23] es un sistema operativo inicialmente desarrollado por Android Inc. y posteriormente adquirido por Google, basado en núcleo Linux y otros softwares de código abierto, ha sido diseñado para dispositivos móvil con pantalla táctil.

El sistema se caracteriza por poseer una estructura de capas, lo que permite la realización de todo tipo de aplicaciones que se pueden integrar en el sistema y permiten añadir todo tipo de funciones y utilidades. Estas aplicaciones pueden desarrollarse en lenguaje Java o Kotlin, para este proyecto he utilizado el lenguaje Kotlin.

Tenemos una amplia variedad de programas que permiten la realización de aplicaciones para estos dispositivos Android, para este trabajo se ha empleado Android Studio [24].

6.1 Android Studio

Es un entorno de desarrollo integrado oficial para el desarrollo de aplicaciones para Android, posee un potente editor de código y herramientas de gran utilidad para manejar el mismo. Se anunció en mayo en 2013.

Permite la utilización de emuladores de sistemas operativos Android para probar el funcionamiento de la aplicación que se esté desarrollando y además permite conectar tu dispositivo Android para instalar y poder utilizar y monitorizar la aplicación.

Además, posee herramientas que permiten monitorizar el uso de memoria y recursos que se está haciendo de la aplicación en tiempo de ejecución. Este recurso es muy útil para observar si hay memory leaks, se está abusando de los recursos, se ejecutan los hilos de procesos correctamente…

A continuación, la Figura 13 muestra una imagen que enseña cómo es el entorno visual de Android Studio sobre el que se ha hecho la aplicación:

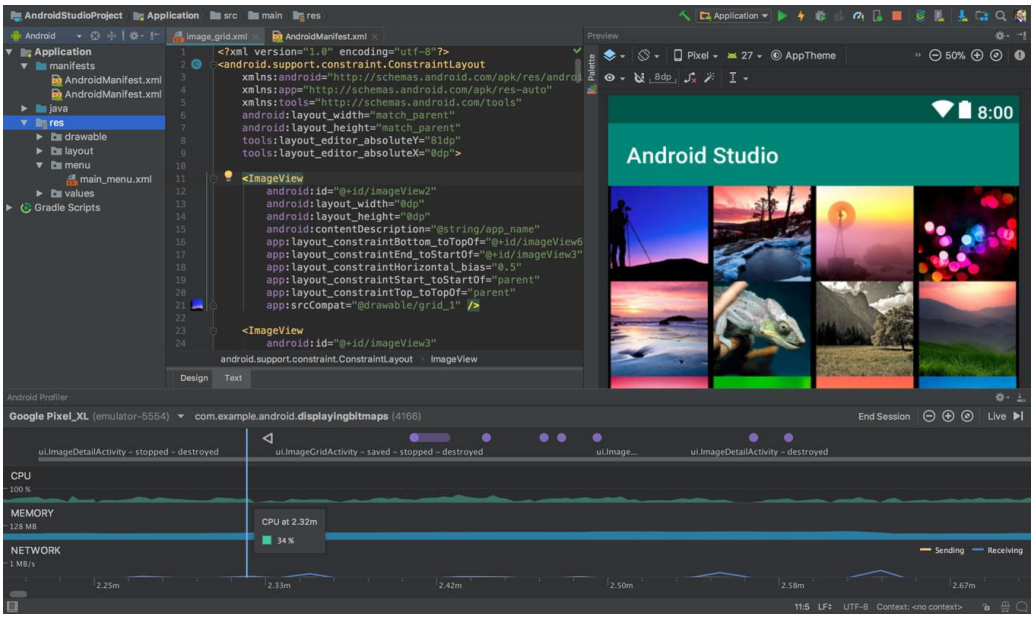

Figura 13. Entorno visual de Android Studio.

6.2 Kotlin

Kotlin [25] es un lenguaje de programación de tipado estático, esto significa que la comprobación de tipificación se realiza durante la compilación, que corre sobre la máquina virtual de Java y además puede ser compilado a código fuente JavaScript.

El objetivo del desarrollo de Kotlin era conseguir un lenguaje de compilación tan rápido como Java, orientado a objetos de calidad industrial y desarrollado para ser interoperable con Java.

Cabe destacar algunas ventajas de usar Kotlin como son la reducción de líneas de código en aproximadamente un 40% con respecto a otros lenguajes y su segura gestión de los nulos, de tal forma que se garantiza que nunca se van a producir errores por tratar con objectos nulos.

6.3 Procesos de la aplicación

Cuando la aplicación se inicia, de forma predeterminada todos los procesos [26] de esta se ejecutan en el mismo proceso, en esta aplicación se han establecido varios procesos con claras funciones que voy a describir a continuación:

6.3.1 Proceso Main o proceso principal

En ella encontramos funciones principales relacionadas con la actividad del dispositivo, tales como onCreate(), onResume() y onPause() [27] que forman parte de la actividad que se ejecuta en el momento y se encargan de escuchar y realizar acciones como consecuencia de las modificaciones que sufra este proceso o actividad.

También encontramos las clases que se encargan de pedir los permisos de apertura de Cámara y escritura de información al usuario cuando ejecute por primera vez la aplicación.

Además, podemos encontrar las clases como Confs, Results y Confidence necesarias para poder ejecutar la librería e interpretar los resultados que esta devuelve y ResourceArchive que se encarga de descomprimir el tessdata para que Tesseract pueda trabajar.

Finalmente cabe destacar las funciones ButtonFrament que se encargan de ejecutar los fragmentos correspondientes según qué botón pulse el usuario.

6.3.2 Proceso fragmento detección

Este fragmento es el que se encarga del proceso de detección del MRZ del usuario, para ello activa la cámara trasera del móvil y enseña por pantalla lo que está viendo, entonces espera a que se obtenga un buen resultado de la lectura del MRZ y aparece un botón que permite al usuario ver los resultados.

El fragmento consta de funciones relacionadas con la actividad del dispositivo (si el usuario sale de la aplicación, gira el móvil, etc se ejecutan estas funciones y reanudan el trabajo o establecen las configuraciones necesarias) y luego consta de las funciones necesarias para poder hacer funcionar la cámara como son getCameraFacingBack() y initCamera() que se encargan de detectar la cámara trasera e iniciarla respectivamente.

Además, cuenta con las clases SurfaceCallback y PreviewCallbackWithBuffer, la primera se encarga de establecer por pantalla la imagen que se está obteniendo continuamente por la cámara trasera y la segunda se encarga de guardar continuamente esta imagen en un buffer y pasarla a la librería a la espera de que devuelva un resultado positivo.

6.3.3 Proceso fragmento resultado

Este fragmento es el que se encarga de mostrar por pantalla el resultado obtenido por el fragmento de detección, cuenta con las funciones relacionadas con la actividad del dispositivo y en ellas ya se encarga de recibir esa información y enseñarla por pantalla al usuario correctamente.

La información que se puede leer consta de: nombres, apellidos, sexo, nacionalidad, id del documento de identificación en cuestión, fecha de nacimiento, fecha de expiración del documento y el texto entero del MRZ.

La Figura 14 muestra un diagrama de la estructura que siguen estos procesos para mayor comprensión y poder verlo y entenderlo de forma visual.

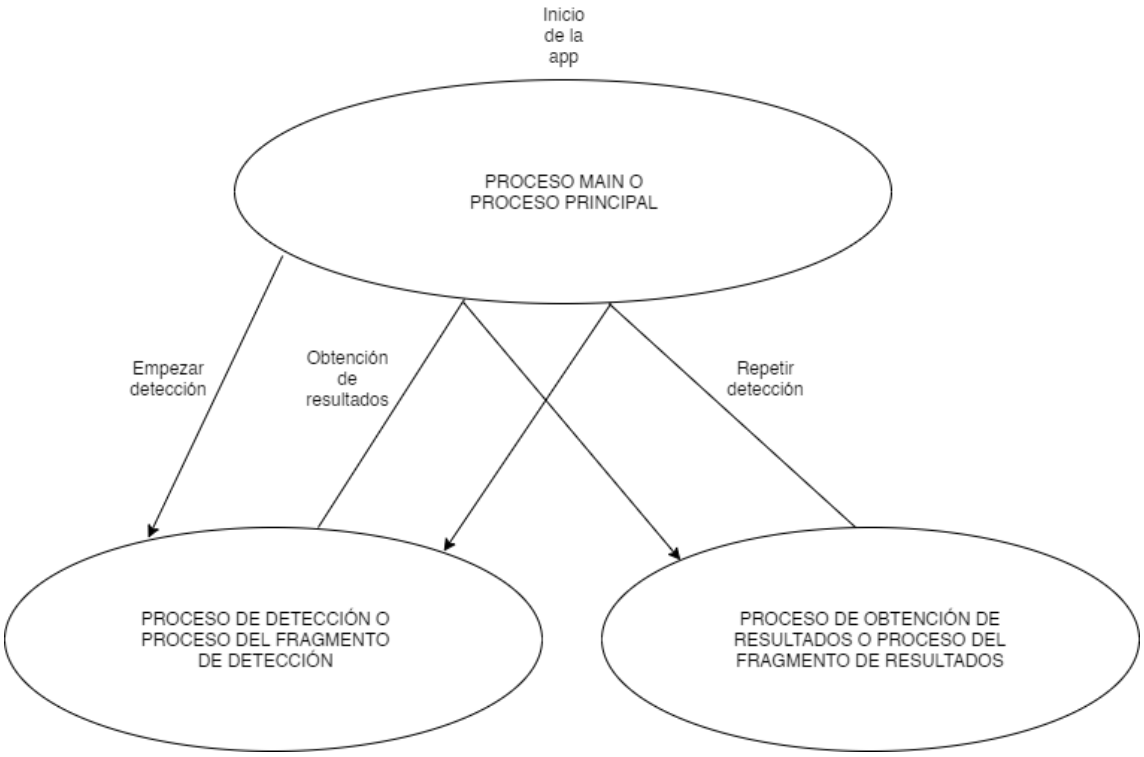

Figura 14. Esquema de procesos.

Como podemos ver para el inicio de la aplicación se ejecuta el proceso main o principal y cuando el usuario decide empezar la detección se ejecuta el proceso de detección, una vez el usuario considera oportuno obtener los resultados de dicho proceso este llama al proceso main y a su vez este ejecuta directamente el proceso de obtención de resultados, podemos ver que, en caso de que el usuario quiera volver al proceso de detección, el de resultados ha de llamar también al main que se encarga de volver a ejecutar el proceso de obtención de resultados.

Queda detallado en el esquema cómo la función del proceso main es la de iniciar la aplicación y hacer de gestora de ambos procesos. Este esquema se ha realizado con la herramienta Draw.io [28].

6.4 Diseño visual de la aplicación

A continuación, se va a disponer un esquema de la estructura visual de la aplicación y una serie de imágenes en las que se podrá ver el diseño actual con el que se encuentra el usuario (véase Figura 15).

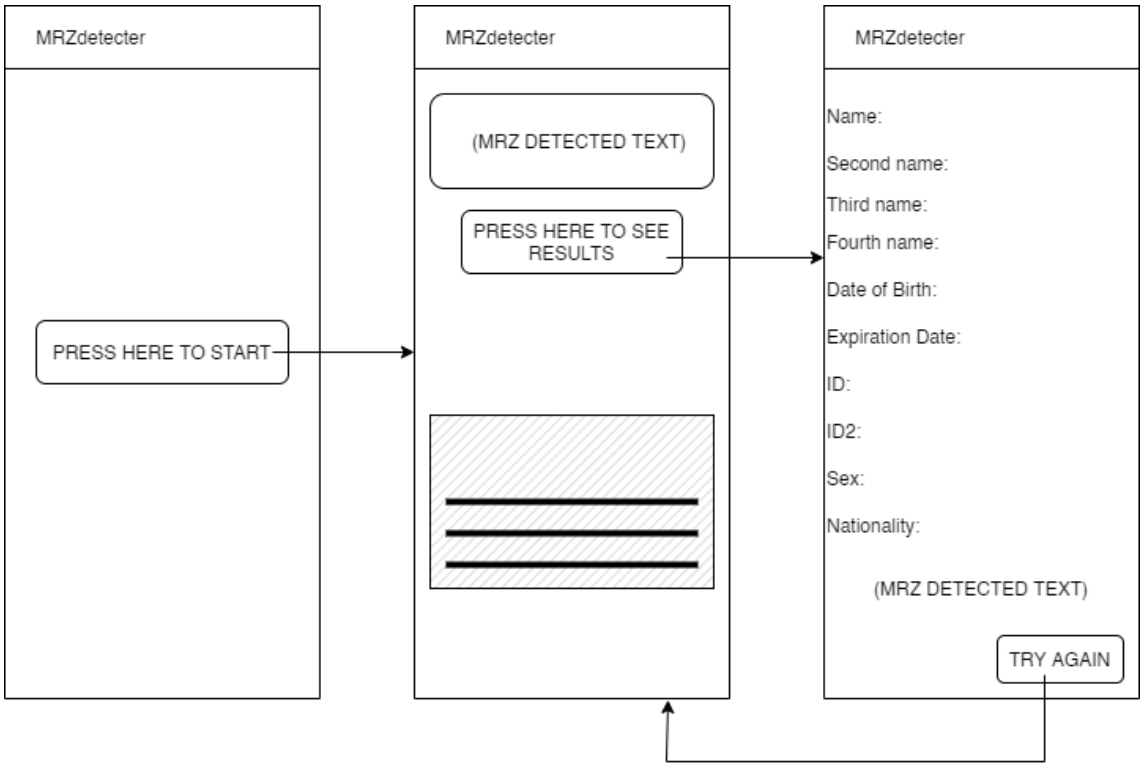

Figura 15. Imagen del diseño visual de la aplicación.

El esquema visual, realizado con la herramienta Draw.io, ilustra la aplicación consta de 3 pantallas, que se describen a continuación.

6.4.1 Primera pantalla

En ella podemos ver un botón en el que se indica al usuario que lo pulse para empezar el proceso de detección. Esta pantalla es llevada por el proceso main o principal el cual se encarga de que, cuando se pulse el botón, se ejecute el proceso del fragmento de detección que podemos ver en la segunda pantalla.

En ella se requieren los permisos de apertura de cámara y almacenaje de información al usuario en caso de que no se hayan dado anteriormente o se hayan revocado.

6.4.2 Segunda pantalla

Se ejecuta al pulsar el botón de la pantalla principal y en esta se abre la cámara del usuario. Al comienzo de esta el texto del MRZ aparece indicando que no se ha encontrado ningún MRZ y una vez el usuario enfoca el documento de identidad empieza llenarse este campo con el texto que se va leyendo.

Asimismo, una vez se ha detectado texto perteneciente al MRZ del documento de identidad aparece un botón en la parte inferior por si el usuario quiere ver la información obtenida del proceso de detección.

El proceso de detección en esta pantalla lo lleva exclusivamente el proceso fragmento que ha lanzado el proceso principal o main.

6.4.3 Tercera pantalla

En ella podemos ver toda la información obtenida en el proceso de detección, esta información aparece desglosada por apartados para que el usuario pueda entenderla fácilmente y ver si el resultado ha sido satisfactorio.

El proceso de obtención de resultados se encarga de llevarlo el proceso fragmento que lanza el proceso principal o main y que no es el mismo que el de detección. Este proceso tiene acceso a la información obtenida en el fragmento de detección gracias a que comparten información entre ellos mediante un view model.

Finalmente, podemos ver un botón en la parte inferior derecha de la pantalla que permite al usuario volver a la segunda pantalla y repetir el proceso de detección en caso de que considere que el resultado obtenido no ha sido satisfactorio.

Cabe destacar que, si giramos el dispositivo móvil, el funcionamiento de la aplicación es el mismo pero la organización de los elementos en pantalla cambia para adaptarse a la nueva orientación establecida.

En la Figura 16 se dispone un esquema de la organización de los elementos por pantalla cuando se gira el dispositivo. El cambio de orientación se puede realizar en cualquier momento, depende del usuario y no altera el proceso de funcionamiento de la aplicación.

Para realizar estos diseños Android Studio cuenta con una interfaz gráfica [29] en la que podemos colocar estos elementos para poder verlos de forma visual ya que resulta más orientativo que establecerlo directamente desde el código, en ella podemos ver como quedan los elementos por pantalla además de establecer algunos parámetros como son el color, posición y proporciones con respecto a la pantalla del dispositivo… todos estos cambios una vez aplicados aparecen reflejados en el código que se encarga de declarar estos elementos y sus características así el programador no ha de preocuparse por declararlos y escribir código.

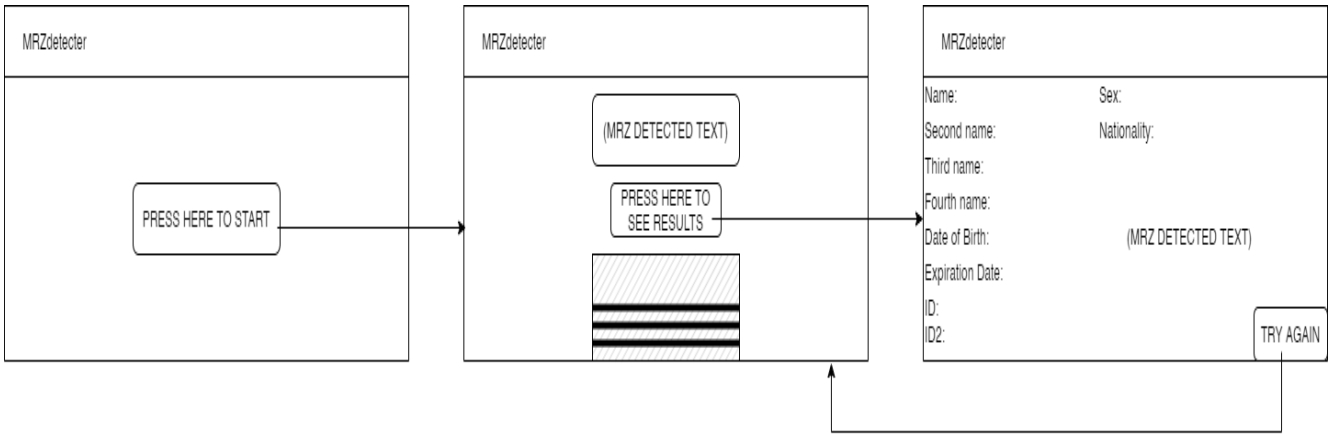

Figura 16. Imagen del diseño de la aplicación con el dispositivo móvil girado.

El hecho de que aparezcan todos elementos reflejados en el código ayuda a que el programador pueda copiar y pegar código para repetir plantillas sin tener que volver a poner todos estos elementos y configurar sus características.

A continuación, en la Figura 17, podemos ver una imagen de cómo es el entorno de desarrollo del diseño de la parte gráfica de la aplicación. Como podemos ver en la parte derecha de la imagen aparece representado gráficamente el código escrito en el fichero xml de la parte izquierda. El editor gráfico nos proporciona herramientas para la modificación de esta como es poner botones, texto u otros elementos y modificar sus características.

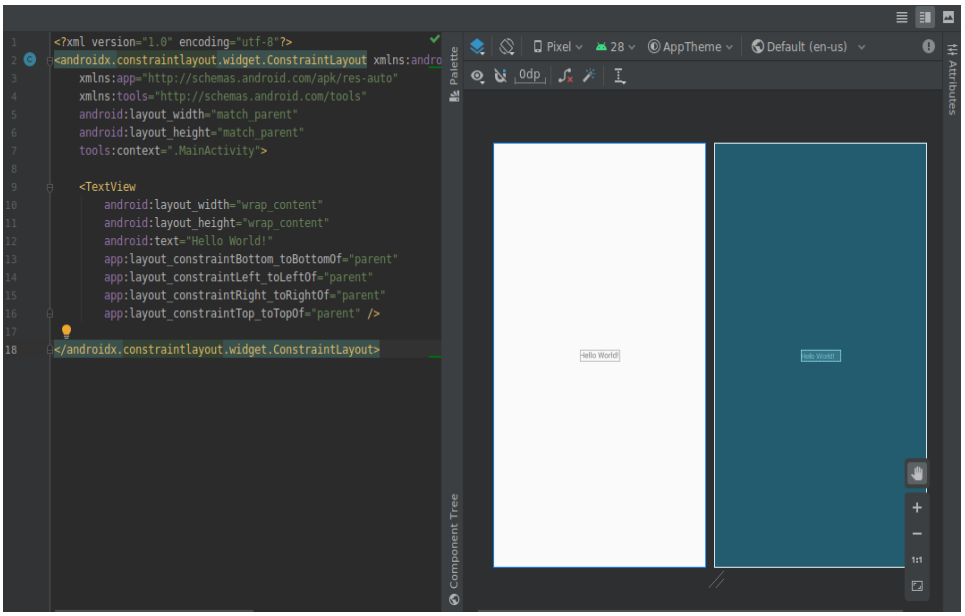

Figura 17. Imagen del entorno de desarrollo gráfico de Android Studio.

7 Aplicación Web

Para el desarrollo de la aplicación web existe un amplio abanico de posibilidades de desarrollo todas y cada una de ellas con sus aspectos positivos y negativos, pero siempre una opción más conveniente dependiendo del tipo de web que se quiera crear. En este caso se ha utilizado el editor de código Visual Studio Code para crear un proyecto React que utilice el lenguaje TypeScript y HTML.

Para la utilización de la librería C++ que se ha creado se ha empleado un compilador Emscripten que devuelve como salida un archivo en el lenguaje JavaScript además de la posibilidad de obtener también un fichero HTML y otro WebAssembly, se puede incluso configurar el compilador para que el código de salida cuente con determinadas características que puedan ser necesarias en nuestro proyecto en el que se vayan a utilizar.

7.1 Desarrollo de la aplicación web

Para el desarrollo de la aplicación web existen muchas posibilidades de elección todas y cada una de ellas con sus aspectos positivos y negativos, pero siempre una opción más conveniente dependiendo del tipo de web que se quiera crear. En este caso se ha utilizado el editor de código Visual Studio Code para crear un proyecto React [30] con el que emplearemos la librería compilada con Emscripten, a continuación, se explicará qué es Visual Studio Code, ReactJS, Typescript, HTML, Emscripten y WebAssembly que son los principales elementos que conforman esta aplicación web.

7.1.1 Visual Studio Code

Para la realización de la misma se ha utilizado el editor de código Visual Studio Code [31], un programa gratuito y de código abierto lo cual ofrece al usuario muchas opciones de personalización como por ejemplo ya sea cambiar el color del tema que utiliza, modificar atajos de teclado y preferencias, también permite crear e integrar extensiones en el mismo que realicen funciones de lo más diversas como por ejemplo resaltar el código erróneo, analizar los métodos creados y determinar si esta hecho de forma óptima o por el contrario no lo es. La Figura 18 muestra el entorno visual de Visual Studio Code.

7.1.3 TypeScript

El lenguaje utilizado para la creación de esta es TypeScript [32], un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft, es un superconjunto de Javascript, que añade tipos estáticos y objetos basados en clases. Es muy útil ya que extiende la sintaxis de JavaScript por tanto cualquier código JavaScript existente debería funcionar sin problemas en el mismo, está pensado para proyectos grandes los cuales mediante un compilador de TypeScript traducen el código a JavaScript original.

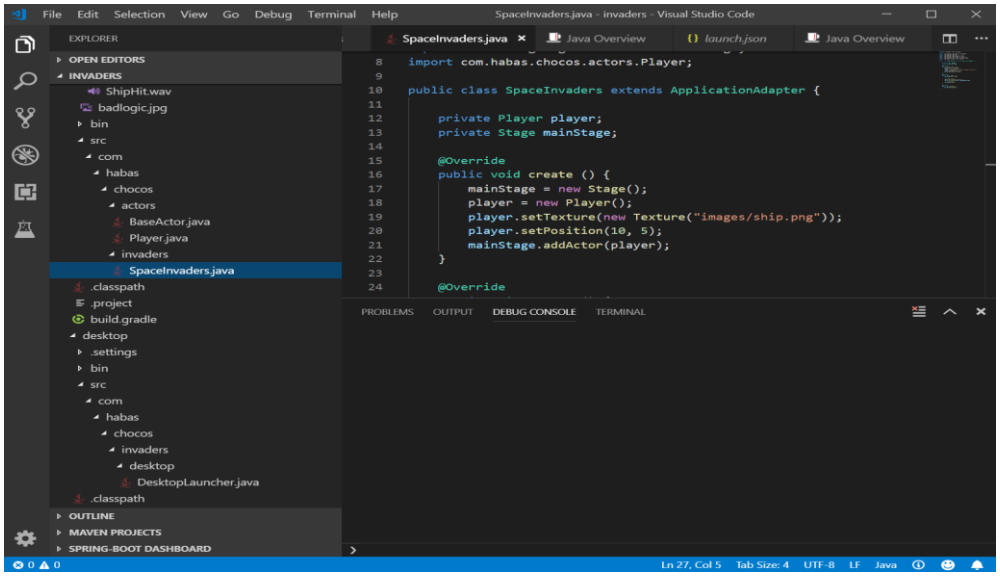

Figura 18. Imagen del entorno de desarrollo de Visual Studio Code.

7.1.4 HTML

HTML [33, 34] se trata de un lenguaje de marcado, es decir, indica como van a ir ordenados los elementos en una página web por medio de marcas de hipertexto o tags. Todos los elementos externos que se utilicen no los crea el HTML sino que se hace referencia a su ubicación, es decir, HTML tiene la tarea de organizar los elementos incorporados y unirlos para permitir su visualización en una página web, pretende ser un estándar para que pueda ser interpretado por todos los navegadores web que estén actualizados.

Ofrece una gran adaptabilidad estructura lógica y es fácil de interpretar tanto por humanos como por máquinas.

7.1.5 Emscripten

Emscripten [35] es un compilador Source-To-Source también denominado transcompilador, puede procesar el bytecode de LLVM normalmente creado al compilar código C o C++ por tanto nos permite compilar código de este tiempo y devolver como salida un fichero en el lenguaje de programación JavaScript además de permitir devolver un fichero HTML y uno WebAssemby y todo ello permitiendo establecer unos parámetros de configuración que faciliten su uso luego según en qué proyecto se vaya a utilizar.

7.1.6 WebAssembly

Webassembly [36, 37, 38] es un formato de código binario portable, creado para la ejecución íntegra en navegador de scripts de lado del cliente se trata de un lenguaje de bajo nivel diseñado inicialmente para compilación de C y C++ y su utilización en Web.

Ofrece mayor velocidad para la ejecución de código a gran escala que JavaScript por lo que pretende cambiar el panorama Web en un futuro cercano.

7.2 Estructura de la aplicación web

La idea de la aplicación web es conseguir una interfaz sencilla y clara para el usuario y una potente y veloz herramienta que realice el trabajo de detección y obtención del texto del MRZ utilizando la librería creada en C++ al comiendo del proyecto.

Al tratarse de un proyecto React podemos identificar varios elementos a destacar:

- Fichero App.tsx, en este fichero se encuentran las clases y funciones que se encargan de abrir la cámara, obtener las imágenes y llamar a las funciones de la librería para empezar el proceso de detección y obtención del resultado.
- Ficheros index.html y index.css, en estos ficheros se establecen algunos parámetros que especifican el comportamiento de la web, su aspecto y organización de los elementos.
- Ficheros librarymrz.js y library.wasm, son los ficheros resultado de compilar la librería C++ con emscripten y así permitir su uso en el proyecto.

7.3 Clases que conforman la aplicación web

La aplicación web se ayuda de las siguientes clases y funciones para poder funcionar:

- AppStreamCam: clase principal que se encarga de la obtención del video de la cámara y su procesamiento para obtener imágenes de ella. Cuenta con una función principal que inicia la cámara y pide permisos al usuario cuando este pulsa el botón de “INICIAR CÁMARA”. Además, cuenta con otra función que se encarga de recopilar imágenes de la cámara y convertirla para permitir su uso en la librería que he creado. Finalmente, cuenta con una función html que permite visualizar la cámara en la web y las imágenes que se están procesando.

- App: esta función se encarga de definir el título y algunos elementos de la página web además de cargar la librería y otros.

- Variables: interfaz compartida en la que se guardan los valores obtenidos de la detección de la imagen procesada por la librería.

7.4 Diseño de la aplicación web

Para el diseño de la aplicación quise utilizar una interfaz sencilla que facilitase el uso de esta y no causase confusión en el usuario, por tanto, el diseño sigue el esquema que muestra la Figura 19.

Figura 19. Imagen del diseño que conforma la app web.

En este esquema realizado con Draw.io se puede ver el diseño final de la aplicación web, consta de un título con el nombre de la aplicación y debajo una instrucción simple que indica al usuario que enfoque el MRZ de su documento de identidad a la cámara.

En medio se enseña la vista de la cámara para que el usuario pueda hacerse a la idea de lo que se está viendo por ella y poder acercar o alejar el MRZ para facilitar su detección.

Abajo podemos ver dos botones, uno de ellos para activar la cámara y el otro se pulsa cuando se quiera dar comienzo al proceso de obtención de información del MRZ del documento de identidad.

Debajo de estos botones se enseña toda la información resultado del proceso de obtención de información.

8 Desarrollo de la solución propuesta

En este apartado se va a detallar todo el proceso de creación del proyecto, desde el principio hasta el producto final y se va a dar información sobre los pasos seguidos sin entrar en mucho detalle en las especificaciones y definiciones anteriormente explicadas.

8.1 Preparando el entorno

Para el desarrollo de este proyecto se ha empezado descargando una máquina virtual concretamente VM VirtualBox, para montar un sistema operativo Linux que utilice Ubuntu, una vez montado el sistema he descargado Eclipse y configurado el entorno para realizar un proyecto en C++

8.2 Creación de la librería

Una vez creado el proyecto Eclipse lo siguiente ha sido descargar las librerías de OpenCV y Tesseract e importarlas al proyecto y asegurarme de su correcto funcionamiento haciendo pruebas con los mismos, una vez estaba todo funcionando empecé haciendo una aplicación que, proporcionada una imagen, realizase las transformaciones y modificaciones necesarias sobre ella para poder encontrar la zona de interés. Para ello se han utilizado imágenes de muestra de documentos de identidad que se han descargado directamente de internet. Me pude asegurar de que todo el proceso iba bien después de añadir las funciones haciendo uso de otras funciones de ayuda que enseñaban la imagen en cada momento requerido.

Una vez el proceso de detección de la zona de interés, en este caso el MRZ, funcionaba correctamente se da por concluida la tarea de OpenCV y paso a utilizar Tesseract para que devuelva el texto que encuentre en la zona de interés. No obstante, el texto resultado no era muy acertado ya que utilizaba un fichero de configuración muy básico.

Para mejorar el resultado que obtuviese Tesseract tenía que hacer un fichero de configuración tessdata que se adaptase a las necesidades del entorno en el que se iba a utilizar, para ello he usado QT-Box Editor, un programa capaz de crear estos archivos de configuración, para su uso fue necesario recoger muestras de imágenes de documentos de identidad procesados por OpenCV y descargar la tipografía que emplean los documentos de identidad, una vez conseguido esto puedo crear el archivo de configuración y proporcionárselo a Tesseract, el texto resultado tras incluir este fichero se vio drásticamente mejorado.

Una vez conseguido el texto resultado deseado, el MRZ del documento de identidad, corregí algunos errores que se daban cuando ejecutaba el proyecto en determinados documentos de identidad de los que tenia de prueba, errores relacionados con imágenes con el MRZ muy cerca del borde, lo que causaba problemas a la hora de recortar la zona de interés y otros errores relacionados con el escalado de la imagen, tenía que ser diferente para pasaportes y documentos de identidad ya que 2 líneas necesitaban un mayor escalado para que se pudiese ver mejor el texto.

Cuando por fin el proyecto daba el resultado deseado sin fallos hice un esquema de clases para determinar la organización y estructura que iba a seguir el proyecto, por eso hice diferentes clases para diferentes funciones: una clase principal que llamase a las otras clases secundarias que realizarían las tareas de detección y lectura, además de las clases que contendrían información sobre los resultados, parámetros o estructuras de información.

Una vez establecida la estructura de clases separé el código correspondiente a su respectiva clase y tras asegurarme que todo seguía funcionando correctamente me dispuse hacer las clases que almacenarían toda esta información.

El texto resultado que devolvía el proyecto tenía que ser interpretado, así que en la propia clase de lectura del MRZ interpreto estos resultados y los guardo además de hacer la verificación de que la información obtenida es correcta, y finalmente devuelvo ese resultado, esto incluye el texto del MRZ, los nombres, los apellidos, el sexo, la nacionalidad, los id del documento, la fecha de nacimiento y de vencimiento y el vector de confianza de los caracteres que se han leído.

Una vez el texto resultado y los datos interpretados eran correctos para todas las muestras que le proporcionaba al proyecto di por finalizado el proyecto, lo siguiente fue verificar asegurarse de que todo estaba hecho de la forma óptima y que no quedase nada por mejorar, además de buscar posibles fallos que pudiesen ocasionar problemas al llevarlo a los sistemas Android y Web.

Finalmente, tras verificarlo todo y llegar a la conclusión de que ya tenía algo cuyo funcionamiento era adecuado para las aplicaciones Android y Web decido compilarlo utilizando CMake, para ello se tuvo que hacer un fichero CMakeLists.txt en el que se especificaría todos los ficheros a utilizar, las dependencias y el formato de salida que se requería. Una vez hecho esto ya disponía de una librería que podía utilizar en mi aplicación Android y además al haberla hecho con CMake cualquier cambio o modificación necesaria no me llevaría mucho de tiempo de realizar y aplicar al proyecto y la aplicación.

8.3 Creación de la aplicación Android

Lo siguiente era abordar la parte del proyecto que realizaba la aplicación Android, para ello descargo la aplicación Android Studio y monto un proyecto que utilice Kotlin, el primer paso en esta aplicación era conseguir que abriese la cámara trasera y sacase por pantalla lo que estaba viendo la cámara, para ello se tuvo que montar la interfaz que emplearíamos en la aplicación así que se hizo un diseño previo de cómo debería verse y tras tenerlo diseñado y montado se hicieron las pruebas para que abriese la cámara.

Una vez estuvo todo montado observé que la imagen que se sacaba era una pantalla en negro y descubrí que eso se debía a que aún no había cedido los permisos a la aplicación para que la pudiese utilizar, así que se incluyó el código necesario para ello y tras ceder permisos a la aplicación para poder utilizar la cámara por fin se veía por pantalla la imagen que estaba capturando la cámara.

No obstante, esta imagen sacada por pantalla era muy rígida, es decir, no enfocaba ni se adaptaba a la luz del entorno, para ello tuve que incluir en el código unas configuraciones que la cámara aplicase mientras estuviese en funcionamiento en caso de que pudiese utilizarlas ya que algunos dispositivos Android no disponen de dichas funciones. En mi caso mi dispositivo Android sí que contaba con ello así que después de aplicar los cambios la cámara enfocaba correctamente y se adaptaba a la luz entorno

y por fin podía obtener una imagen nítida que se pudiese utilizar para obtener el texto resultado.

Tras obtener una imagen nítida a través de la cámara, empleo un buffer donde voy guardando continuamente frames de la imagen que está capturando, la imagen que se obtiene de la cámara está en formato YUV por lo que en el buffer puedo almacenar solo la cantidad de información necesaria, solo se guardan determinados pixeles, concretamente los que conforman la imagen en blanco y negro.

Lo siguiente era hacer uso del proyecto que desarrollé al principio, la librería, para poder procesar esa imagen que obtenía de la cámara y obtener el texto resultado.

Tras asegurarme de que todas las librerías funcionan correctamente y no dan problemas en el proyecto de Android Studio lo siguiente era pasar de forma continua a esta librería las imágenes que iban entrando al buffer y quedarme sólo con los resultados satisfactorios.

No obstante, observo que los resultados de la librería son incorrectos debido a que el fichero de configuración tessdata en este proyecto se incluye de forma distinta, así que se decide que lo mejor es tener el fichero configuración comprimido y descomprimirlo al ejecutar la aplicación, incluyendo en el código las operaciones correspondientes y el de la obtención del path 0 camino a este fichero para pasárselo a la librería.

Una vez empezamos a obtener resultados satisfactorios estos se guardan. Para poder enseñar estos resultados obtenidos por pantalla creo un fragmento cuya función es la de enseñarlos, para ello se diseña el fragmento, qué apariencia va a tener y cómo se le iba a pasar esa información, así que para que se pudiese transmitir esa información al fragmento se crea un view model, el cual se encarga de almacenar esa información y permite que otros procesos la consulten.

Tras finalmente poder sacar los resultados por pantalla corrijo algunos errores de ejecución del programa, la aplicación debía funcionar también cuando se girase la cámara del dispositivo así que se añadieron los cambios necesarios para ello, además se corrige un error que ocasionaba que la aplicación se cerrase cuando salías de la aplicación y volvías a entrar con el dispositivo en una orientación distinta. Hubo que transmitir la información de la rotación del dispositivo a la librería para que girase la imagen y la orientase correctamente para poder hacer la lectura correctamente.

Finalmente, la aplicación conseguía capturar de forma continua imágenes de la cámara y obtener resultados satisfactorios además de poder enseñarlos cuando el usuario decidiese. Tras tener esto funcionando decido cambiar la estructura del proyecto y hacer una actividad principal que gestionase ambos fragmentos de detección y de enseñar los resultados.

Para poder entender el funcionamiento de los fragmentos y como llevarlo a cabo en mi proyecto que ya de por si estaba prácticamente finalizado tuve que hacer una sencilla aplicación con un proceso principal main y otros dos fragmentos, en la aplicación principal ponía un texto cualquiera y luego se abría el primer fragmento que me pedía otro texto cualquiera, una vez introducido se enseñaban ambos textos por el último fragmento.

La creación de este subproyecto me ayudó a entender mejor como funcionaban los fragmentos y poder llevarlo luego a mayor escala en mi proyecto principal para que funcionase todo correctamente.

Después de implementar los fragmentos en mi proyecto principal y que todo funcione correctamente di por concluida la aplicación Android.

8.4 Creación de la aplicación web

Para el desarrollo de la aplicación web fue necesario instalar Visual Studio Code y realizar el proyecto en Windows10 ya que así se podía detectar la webcam sin problemas, tras tener el todo instalado se crea un proyecto React que utilice Typescript, lo primero era sacar por pantalla la imagen que la webcam obtuviese de forma continua.

Para ello se hizo una clase que activase la webcam cuando el usuario pulsase un botón y empezara a guardar frames de la imagen para pasarlos a la librería de forma continua.

También se hizo una clase con las correspondientes variables del texto que se iba a obtener del MRZ para posteriormente enseñarlas por pantalla.

Con todo esto funcionando solo quedaba importar la librería al proyecto y pasar la imagen para que realizase la detección de esta.

Al tratarse de una aplicación web se ha tenido que usar Emscriten para crear las librerías correspondientes a partir del proyecto que desarrollé al principio con Eclipse y de las librerías de OpenCV y Tesseract. Este proceso fue largo ya que se tuvo que hacer todo de forma manual, primero se crearon los objetos del código cpp, luego de los objetos se hizo una librería o archivo y finalmente de este archivo se tuvo que obtener los ficheros js y wasm que se iban a utilizar en el proyecto para ejecutar las funciones de detección.

No obstante, no resultó como esperaba y no permitía ejecutar nada por tanto algo del proceso de compilación había fallado. Para asegurarme de que lo estaba haciendo bien decidí hacer un proyecto a pequeña escala, hice código C++ que realizase sumas y restas y lo compilé con Emscripten siguiendo el proceso anteriormente descrito y resulto que tampoco funcionaba.

Tras estar investigando observé que había que especificar unos determinados valores en la compilación y tras hacerlo en mi proyecto a pequeña escala pude conseguir que los ficheros resultantes de la compilación ejecutasen las funciones de C++ que había creado.

Por tanto, ahora solo quedaba seguir estos mismos pasos en mi proyecto a gran escala, al intentarlo observé que daba problemas con las librerías correspondientes a OpenCV y Tesseract, por tanto, fue necesario compilar todo el código C++ de estas a Emscripten para posteriormente unirlo a la compilación de mi librería y que no diese errores.

Una vez conseguido que mi librería compile y exporte las funciones correctamente junto con OpenCV y Tesseract llega el momento de incluirlo en mi proyecto Web para poder llamar a sus funciones y que haga la detección de las imágenes correctamente.

Tras incluirlo en mi proyecto observo que llamar a las funciones no es tan fácil como se especificaba así que después de estar probando diferentes métodos y formas de utilizar la librería en el proyecto no consigo dar con alguna que saque resultados esperados.

Tras estos resultados negativos, en vez de utilizarlo en mi proyecto React utilizo un fichero html bastante básico y sencillo que carga una imagen de un pasaporte y se lo pasa a la librería compilada.

El fichero html no dio problemas para cargar la librería y sorprendentemente permitía la ejecución de las funciones de la librería de una forma bastante sencilla y clara, esto ayudó bastante a la hora de determinar si funcionaba correctamente.

Una vez conseguido que todo cargase correctamente se empiezan a hacer pruebas con la imagen del pasaporte que se había cargado, tras pasarlo a la librería observo que el proceso tarda mucho y que no conseguía realizar todo el proceso de detección y lectura.

Gracias a que podía emitir mensajes desde el código de mi librería poco a poco pude depurar el proceso por el cual pasaba la imagen y conseguir que pasase por todo el recorrido hasta devolver un resultado de lectura.

No obstante el resultado de la lectura era incorrecto, devolvía muchos caracteres confusos y sin sentido alguno, por tanto determiné que seguramente el error provenía o bien del formato en el que se pasaba la información a la librería o bien de alguna transformación que no resultaba bien en el proceso, decidí investigar más al respecto y ver si conseguía encontrar una forma de ver las transformaciones que sufría la imagen en el proceso de detección, además de informarme de todas las opciones de formato de imagen que tenía disponible para pasarle a mi librería.

Tras estar realizando muchas pruebas, entre ellas por ejemplo intentar guardar la imagen que se obtenía en un jpg compilando y utilizando libjpeg, en un png compilando y utilizando libpng, en un base64 compilando y utilizando libbase64 para su posterior visualización en la web, reinterpretar el buffer en el que se guardaba la imagen y volver a visualizar su información, cambiar el tipo de información que se pasaba utilizando un blob … al final conseguí dar con el problema y al cambiar el formato con el que pasaba la imagen a un tipo puntero de const char conseguí el primer resultado positivo.

Una vez obtenido el resultado positivo compilo el proyecto otra vez eliminando todo el ruido que se había generado para realizar las pruebas y encontrar el problema y así tener una compilación limpia que solo mostrase por pantalla lo necesario.

Me aseguré de que seguía funcionando de la forma correcta y lo añadí a mi proyecto React. Aquí empezaron los problemas de nuevo ya que React me dio muchas complicaciones para poder cargar mi librería y utilizarla debido a que no permitía un uso de Module para ejecutar las funciones de la librería de forma tan libre como el fichero html que utilizaba para realizar las pruebas.

Finalmente conseguí hacer que el proyecto React utilizase Module para ejecutar las funciones de una forma similar a la que se utilizaba en el html, una vez hecho esto hice la prueba de lectura de mi documento de identidad con la webcam y la librería me devolvió el resultado esperado, una lectura con éxito del mismo.

Tras finalizar esto ahora solo quedaba decorar el entorno para el usuario y para ello se cambió la forma en la que se mostraba el MRZ (en tres líneas separadas en vez de una entera), esconder el canvas que enseña la imagen capturada por la webcam, y otros detalles que facilitasen todo el proceso.

9 Implementación

A continuación, se van a enseñar los resultados obtenidos por la librería desarrollada, la aplicación Android y por la aplicación web.

9.1 Resultados obtenidos por la librería

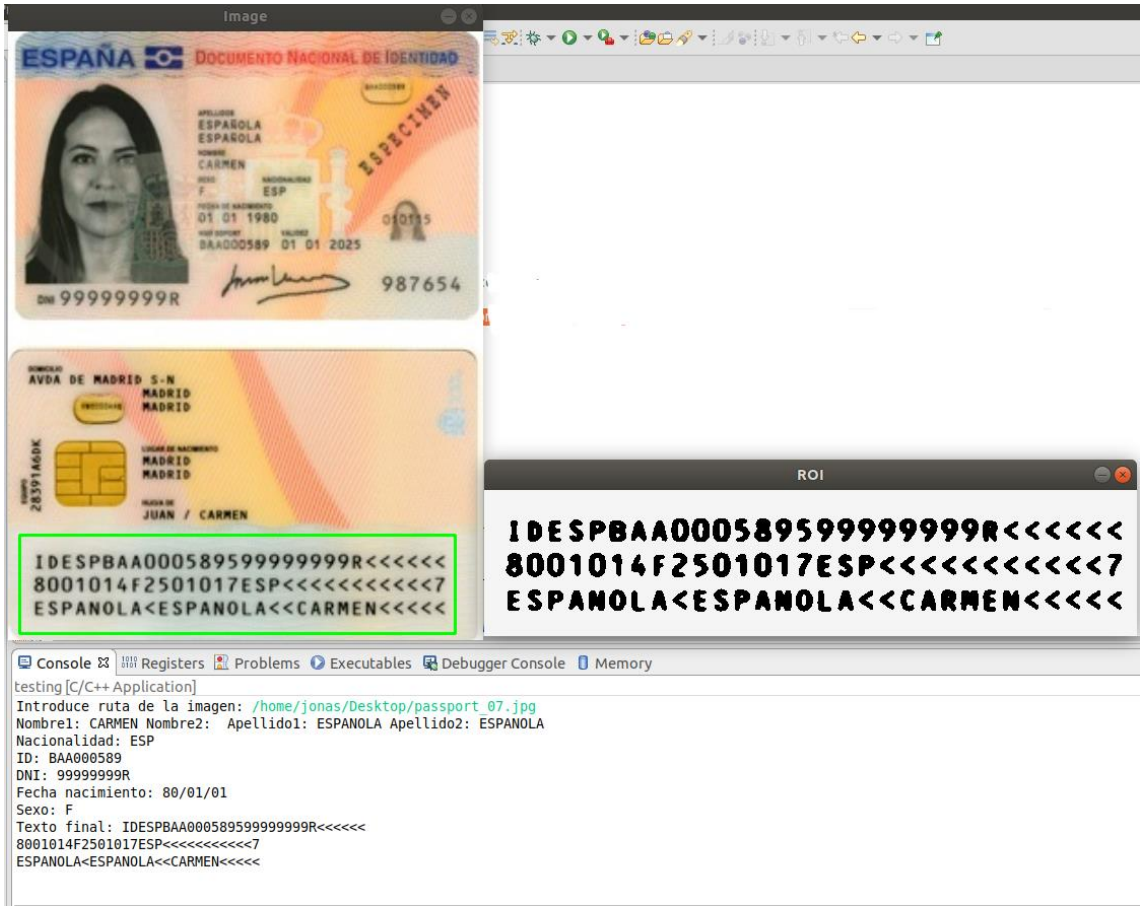

Figura 20. Resultado obtenido cuando la librería procesa un DNI.

En la imagen de la Figura 20 podemos ver el resultado que nos devuelve el proyecto Eclipse cuando procesa una imagen que contiene un MRZ, posteriormente de este proyecto se ha hecho la librería

9.2 Resultados obtenidos por la aplicación Android

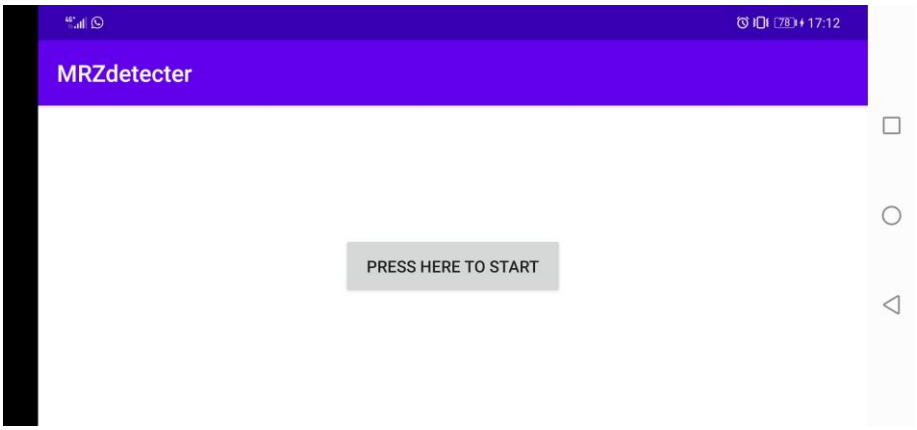

Figura 21. Primera pantalla de la aplicación con el móvil en horizontal.

Figura 22. Primera pantalla de la aplicación con el móvil en vertical.

Lo que podemos observar en las dos imágenes de las Figuras 21 y 22 es la pantalla principal de la aplicación Android, donde se solicita al usuario que pulse el botón para empezar con la detección. La primera imagen se ha tomado con la rotación del móvil activada y en horizontal, la segunda en vertical.

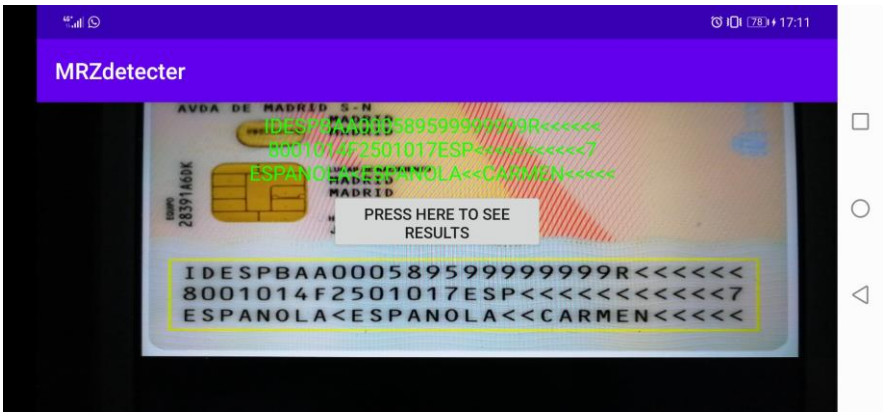

Figura 23. Segunda pantalla de la aplicación con el móvil en horizontal

Figura 24. Segunda pantalla de la aplicación con el móvil en vertical.

En las dos imágenes mostradas en las Figuras 23 y 24 podemos ver el proceso de detección de la aplicación en el que va apareciendo el texto resultado arriba además de un botón que da la opción al usuario de enseñar los resultados hasta ahora obtenidos. La primera imagen se ha tomado con la rotación del móvil activada y en horizontal, la segunda en vertical.

Figura 25. Tercera pantalla de la aplicación con el móvil en horizontal.

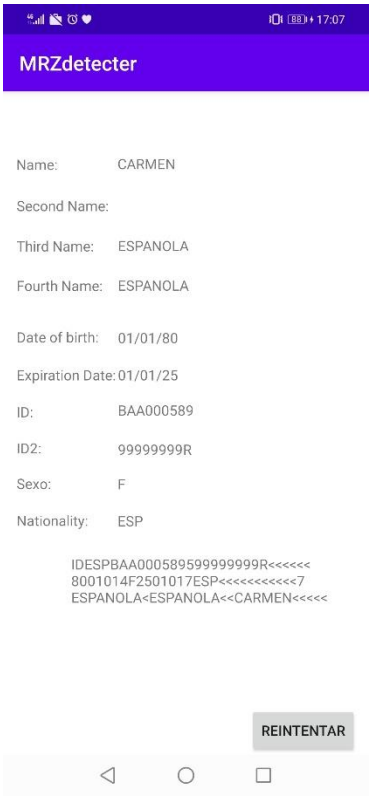

Figura 26. Tercera pantalla de la aplicación con el móvil en vertical.

En las dos imágenes de las Figuras 25 y 26 podemos ver la pantalla con todos los datos resultados obtenidos del proceso de detección además de un botón que da la opción al usuario de empezar de nuevo con el proceso de detección en caso de que los datos no hayan sido satisfactorios. La primera imagen se ha tomado con la rotación del móvil activada y en horizontal, la segunda en vertical.

9.3 Resultados obtenidos por la aplicación web

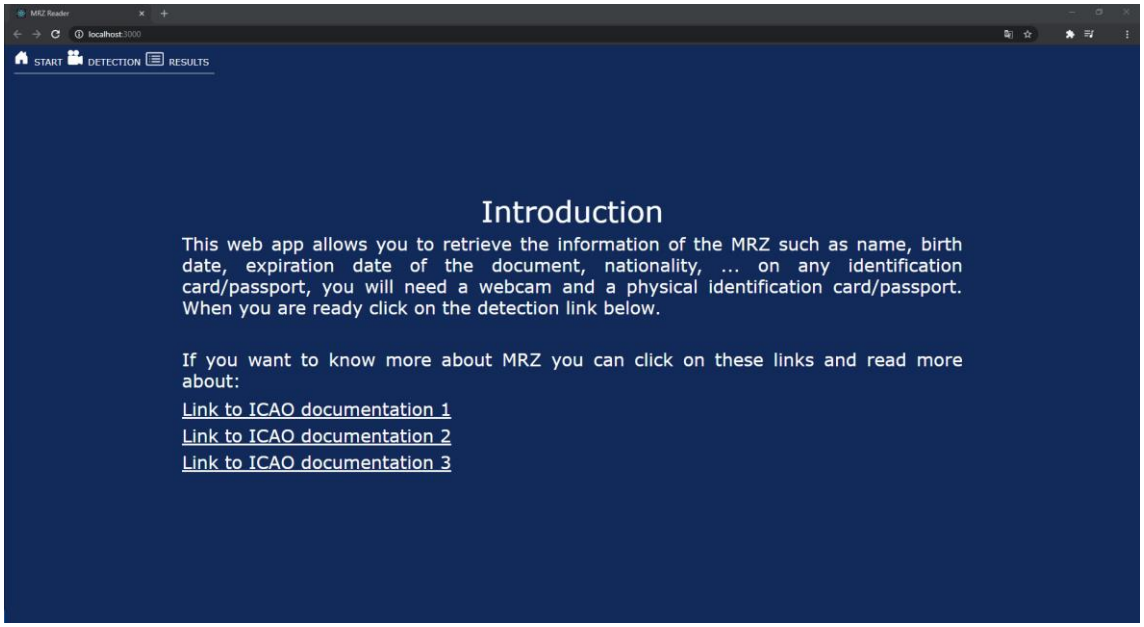

Figura 27. Captura de la pantalla principal de la aplicación web.

En la imagen de la Figura 27 podemos ver una captura de la pantalla principal de la aplicación web en la que observamos una breve descripción del funcionamiento de esta para guiar al usuario. También se pueden ver tres enlaces abajo que puede utilizar el usuario para obtener más información acerca de los MRZ.

Arriba a la derecha se disponen tres enlaces que usuario puede utilizar para alternar entre las diferentes páginas disponibles dentro de la aplicación web, se puede observar “START” que hace referencia a la página principal en la que se encuentra la breve introducción para guiar al usuario, también vemos “DETECTION” donde el usuario iniciará el proceso de detección y finalmente “RESULTS” donde podrá observar los resultados del proceso de detección.

La Figura 28 ilustra una captura de la pantalla de detección de la aplicación web, en ella podemos ver que se dispone de dos botones, uno para activar la webcam y otro para empezar a detectar el MRZ del documento.

Además, podemos ver un texto debajo de la imagen que nos indica que hay resultados disponibles para visualizar en caso de que los haya. Al cabo de 10s se enseña un mensaje por pantalla en la que avisamos al usuario de que recargue la pagina en caso de que no el proceso de detección esté tardando demasiado.

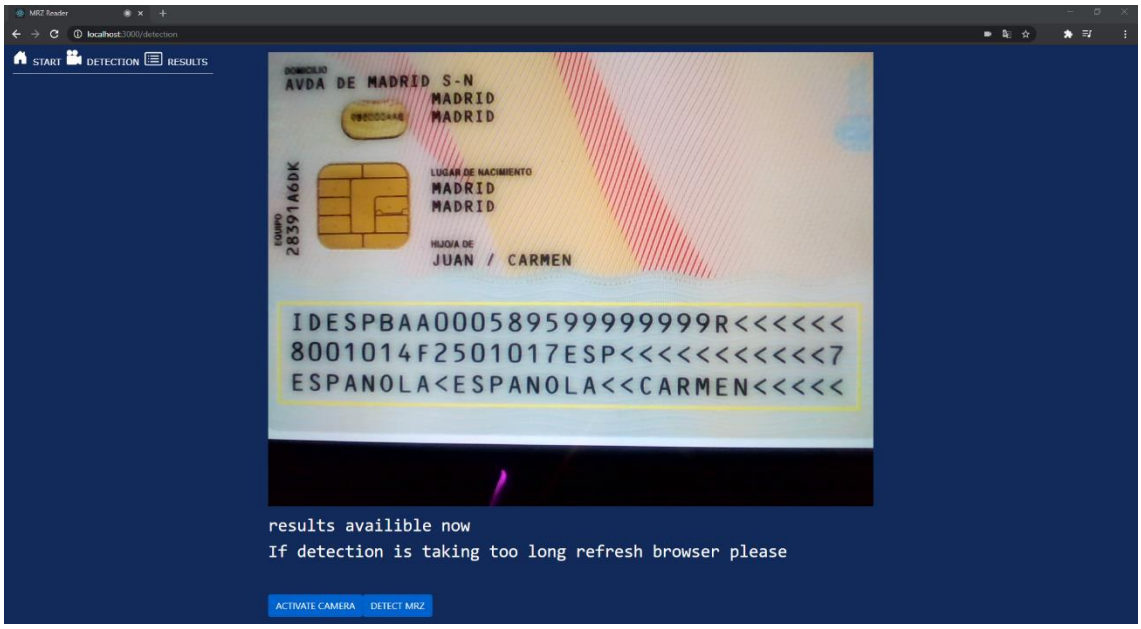

Figura 28. Captura de la pantalla de detección de la aplicación.

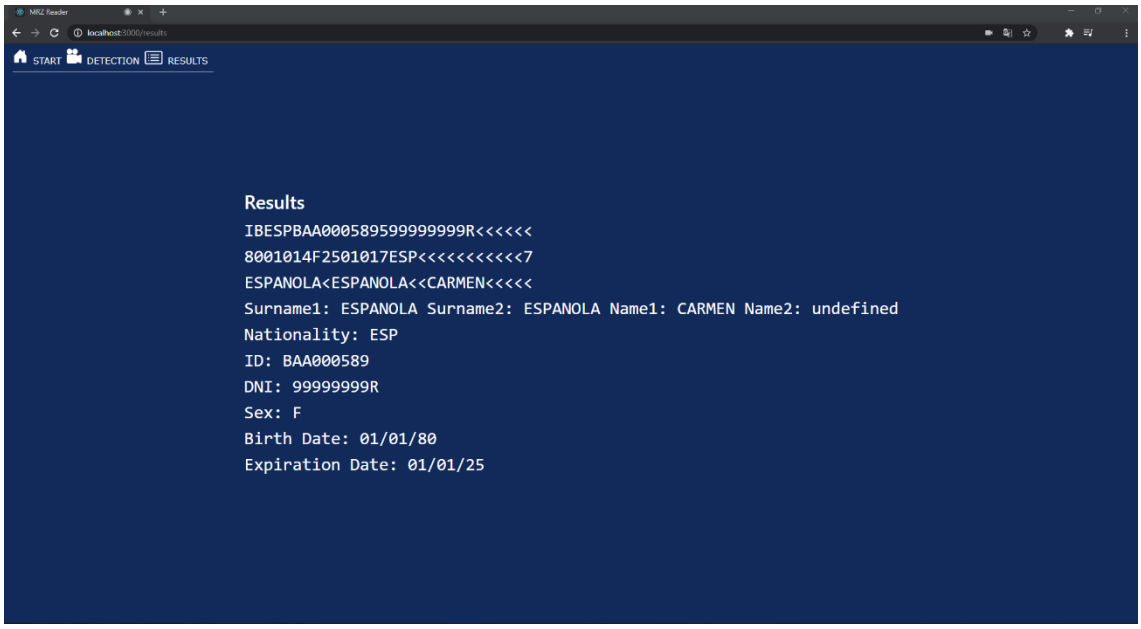

Figura 29. Captura de la pantalla de resultados obtenidos.

En la imagen de la Figura 29 se observan todos los resultados obtenidos en la pantalla de detección, concretamente aparecen el MRZ completo, los apellidos y nombres del usuario, su nacionalidad, el ID de su documento de identidad, el DNI de este, el sexo, la fecha de nacimiento y la fecha de expiración del documento.

Debajo de estos resultados aparecen los 3 links de las diferentes pantallas por las que ha ido navegando el usuario en caso de que quiera volver al comienzo o quiera volver a la fase de detección en caso de que algún resultado no haya sido bien detectado o quiera repetir el proceso.

10 Pruebas

Las pruebas realizadas para verificar el correcto funcionamiento de la librería se han empleado documentos de identidad y pasaportes obtenidos de google imágenes (Figura 30).

Figura 30. Muestra de dos documentos de identidad utilizados para realizar pruebas en la librería.

También se devolvía la imagen resultado de cada transformación o modificación que se realizaban sobre ellos, tal como ilustra la Figura 31.

Figura 31. Una de las imágenes resultado del procesamiento del documento de identidad.

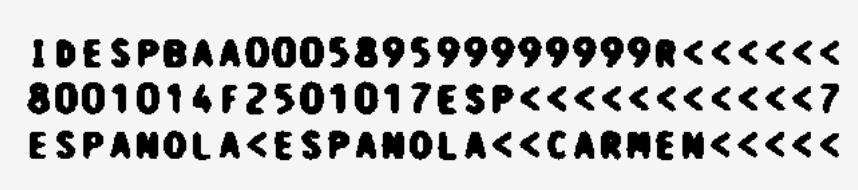

Figura 32. Otra de las imágenes resultado del procesamiento del documento de identidad.

Respecto las pruebas realizadas sobre Android, se hacen directamente con mi dispositivo Android, empleando mi documento de identidad y otros documentos de identidad obtenidos de Google imágenes (Figura 33)

Figura 33. Imagen de un documento de identidad empleado para la detección del MRZ en mi dispositivo móvil y web.

Para las pruebas realizadas sobre la aplicación web se hacen directamente con la webcam que tengo y empleando también mi documento de identidad y la imagen de la Figura 33.

11 Conclusiones y trabajos futuros

11.1 Conclusiones

Se han podido alcanzar los objetivos planteados al principio del proyecto, la librería funciona correctamente y las aplicaciones consiguen leer el MRZ del documento de identidad del usuario haciendo uso de la librería desarrollada y enseñar el resultado al usuario, todo empleando OpenCV y Tesseract.

Nunca había trabajado con aplicaciones como Android Studio o Visual Studio Code ni empleado antes los lenguajes de Kotlin, Typescript o Webassembly no obstante el aprendizaje de los mismos no ha supuesto un problema ya que conforme trabajaba con ellos más rápidamente me iba adaptando a cómo utilizarlos y su forma de funcionar, han sido de gran ayuda la información encontrada en internet sobre la documentación del uso de los mismos lenguajes y la forma de adaptarnos a entornos así adquirida en mis estudios en la universidad.

Ha sido de gran utilidad todo lo estudiado en la universidad sobre todo lo relacionado con programación orientado a desarrollar programas con clases con funciones distintas cada una, procesamiento de imágenes, matrices, como recorrerlas utilizando bucles de la forma más eficiente y extraer información de la forma menos costosa.

De este proyecto he aprendido a montar una librería con CMake y como adaptar su uso a aplicaciones Android y web lo cual ha resultado muy satisfactorio y a mi parecer de bastante utilidad, también he aprendido el funcionamiento de los programas realizados en Android Studio y como se gestiona por actividades clasificadas en fragmentos y una actividad main principal que las gestiona. He aprendido también a como realizar aplicaciones Android y web.

El desarrollo de ambos proyectos se ha desarrollado de forma satisfactoria, el mayor reto de este proyecto no ha sido aprender a programar en los lenguajes que se han empleado y que nunca había tenido posibilidad de utilizar en proyectos de este tamaño, el mayor reto ha sido aprender cómo funcionan las estructuras de los proyectos y aprender a utilizar CMake para la conversión de librerías para su uso correspondiente en las aplicación y luego en las aplicaciones conseguir que el proyecto las utilice sin errores de por medio.

11.2 Relación del trabajo desarrollado con los estudios cursados

A lo largo del curso universitario se han adquirido conocimientos que me han ayudado en el desarrollo del trabajo.

De una forma más general cabe destacar la metodología empleada a la hora de desarrollar programa, buscar los errores del mismo, establecer su estructura, la forma de escribir y desarrollar el código para facilitar su comprensión, encontrar la forma óptima de escribir el código para que no se usen gran cantidad de recursos ni pierda tiempo, es decir, todo un conjunto de habilidades y conocimientos que se adquieren a

lo largo del curso y que son independientes al entorno en el que se haya programado y son aplicables a todos los ámbitos de la programación.

De una forma más específica me ha ayudado todo lo que aprendimos en el curso relacionado con el procesamiento de imágenes, es decir, que las imágenes son matrices de información que se pueden recorrer y modificar con bucles. Tambien ha sido de gran utilidad el desarrollo de clases dentro de un programa y los métodos y funciones necesarias para que sea fácilmente accesible, así como la forma de organizarlas según su trabajo o función dentro del programa.

11.3 Trabajos futuros

Para este trabajo me hubiese gustado desarrollar la aplicación para dispositivos Apple, era un apartado opcional y por falta de tiempo no lo he podido realizar, me hubiese gustado conocer el entorno de programación, como funciona y realizarlo tal y como hice con las aplicaciones Android y web. También me hubiese gustado utilizar el vector de confianzas para cambiar el color del texto sacado por pantalla para informar al usuario del nivel de confianza y acierto que está teniendo el programa a la hora de leer el MRZ de su documento de identidad.

La realización de este trabajo abre puertas para utilizarlo y comprobar su funcionamiento en una base de datos, es decir, se podría ampliar y mejorar utilizándolo de forma conjunta con una base de datos de usuarios de la empresa en cuestión y permitiría al usuario acceder y verificarse utilizando esta aplicación. Tengo mucha curiosidad de cómo se realizaría este proceso de utilizar esta aplicación con una base de datos y lo que ello conllevaría.

Para mejorar la eficiencia de este trabajo y ampliarlo me hubiese gustado desarrollar por mi cuenta un archivo de configuración de Tesseract más eficiente, es decir el tessdata, ya que el que hice solo utilizaba 13 muestras y para una mayor eficacia hubiese resultado más beneficioso utilizar 50 muestras teniendo en cuenta más factores que intervienen en una mejor detección, investigando así en un entorno de usuario todos los elementos que pueden alterar la lectura y ponerles remedio, pero por desgracia no disponía de tiempo suficiente para dedicar a esa parte.

Con el trabajo terminado si hubiese que añadir mejoras o ampliar el mismo no tendría sentido enfocarse en optimizarlo centrándose en los aspectos de los dispositivos que se van a utilizar y sus configuraciones ya que el dispositivo de cada usuario es diferente y acabaríamos perdiendo mucho tiempo en conseguir que funcione con las mejores configuraciones en absolutamente todos los dispositivos disponibles en el mercado, tendría más sentido enfocarse en cómo reducir el tamaño de las librerías importando y utilizando solo lo estrictamente necesario en el código y agilizar la carga de las mismas en el programa, además de crear un fichero de configuración de una forma más minuciosa y detallada investigando todas las posibilidades.

Referencias

[1] ReadID.

https://readid.com/

[2] Eclipse.

https://www.eclipse.org/

[3] C++.

https://www.cplusplus.com/info/description/

[4] CMake.

https://cmake.org/

[5] OpenCV.

https://opencv.org/about/

[6] Tesseract.

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

[7] GaussianBlur.

https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html

[8] Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150. Prentice Hall, 2001.

[9] Mark S. Nixon and Alberto S. Aguado. Feature Extraction and Image Processing. Academic Press, p. 88, 2008.

[10] R.A. Haddad and A.N. Akansu, "A Class of Fast Gaussian Binomial Filters for Speech and Image Processing," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 39, pp 723-727, March 1991.

[11] BlackHat.

https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

[12] Image Analysis and Mathematical Morphology by Jean Serra, ISBN 0-12-637240-3 (1982).

[13] Digital Image Processing (Third Edition) by Rafael C. Gonzalez and Richard E. Woods, ISBN 978-93-325-7032-0 (2008)

[14] Sobel.

https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

[15] Soft Computing Techniques in Vision Science 395. Springer. Patnik, S. and Yang Y.M. (2012)

[16] Close.

https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

[17] Threshold y Otsu.

https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html

[18] Image segmentation based on 2D Otsu method with histogram analysis. Zhang, Jun & Hu, Jinglu, ISBN 978-0-7695-3336-0 (2008)

[19] A comparative performance study of several global thresholding techniques for segmentation. Computer Vision, Graphics, and Image Processing. Lee, Sang Uk and Chung, Seok Yoon and Park, Rae Hong (1990)

[20] Erode.

https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

[21] Morphological Image Analysis; Principles and Applications by Pierre Soille, ISBN 3-540-65671-5 (1999)

[22] Documentación ICAO.

https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf

[23] Android.

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

[24] Android Studio.

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

[25] Kotlin.

https://www.qualitydevs.com/2019/10/02/que-es-kotlin/

[26] Procesos Android.

https://developer.android.com/guide/components/processes-and-threads?hl=es

[27] Ciclos de vida de una actividad Android.

https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es

[28] Draw.io.

https://app.diagrams.net/

[29] Diseño Android Studio.

https://developer.android.com/studio/write/layout-editor

[30] React.

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

[31] Visual Studio Code.

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

[32] Typescript.

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

[33] HTML.

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

[34] https://codigofacilito.com/articulos/que-es-html

[35] Emscripten.

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

[36] WebAssembly.

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

[37] https://webassembly.org/

[38] https://pablomagaz.com/blog/empezando-con-webassembly