App móvil para el escaneo y verbalización de documentos

Ivan Cano Sánchez

Resumen— Este proyecto trata sobre el desarrollo de una aplicación móvil destinada a ayudar a las personas con problemas graves de visión para que sean capaces de "leer" el texto que se encuentra en su entorno durante el día a día, tales como las instrucciones de cualquier aparato, etiquetas o el menú de un restaurante. La aplicación estará disponible para la plataforma Android con posibilidad de ampliar a otras plataformas como iOS o Windows Phone, para eso se hará uso de Xamarin. Xamarin es un framework escrito en C# obtenido recientemente por Microsoft para el desarrollo aplicaciones Cross-Platform, lo cual significa poder desarrollar aplicaciones para múltiples plataformas mediante un código común. La aplicación permitirá interpretar el texto a partir de una imagen realizada a través de la cámara del dispositivo móvil.

Palabras clave— Aplicación móvil, Xamarin, framework, aplicaciones Cross-Platform, interpretar texto, cámara del dispositivo móvil.

Abstract— This project is about developing a mobile application to help people with severe vision problems so they can "read" the text that is in their environment during the day, such as instructions for any device, labels or menu of the restaurant. The application will be available for the Android platform with the possibility of expanding to other platforms such as iOS or Windows Phone, for that will be made use of Xamarin. Xamarin is a framework in C# recently obtained by Microsoft to develop Cross-Platform applications, which means being able to develop applications for multiple platforms through a common code. The application allows to interpret the text of an image made through the camera of the mobile device.

Index Terms— Mobile application, Xamarin, framework, Cross-Platform applications, interpret the text, camera of the mobile device.

1 Introducción

Uno de los principales problemas a los que se enfrenta cualquier persona ciega o con problemas graves de visión es el no poder leer los textos que se encuentran a su alrededor en su día a día. Lo que se tratará de lograr con esta aplicación es dar independencia a la gente que padece este tipo de problemas, ya que solo necesitará realizar una fotografía al contenedor del texto el cual se quiere leer y la aplicación le dirá si se encuentra texto o no, y en caso afirmativo leerá el texto encontrado, además de permitir almacenar los resultados obtenidos. Los textos antes de ser "mostrados" al usuario serán traducidos al idioma en el que esté configurado su dispositivo. Debido al público objetivo al que está destinada la aplicación, ésta deberá de ser fácil e intuitiva de utilizar, sin interfaces sobrecargadas. Es por ello que se implementará una interfaz cámara propia sin hacer uso de la aplicación cámara que todos los dispositivos incorporan. Otro de los elementos a tener en cuenta, es que estos usuarios muy probablemente utilicen alguna aplicación de accesibilidad como TalkBack para poder manejar su dispositivo móvil, por lo que la aplicación deberá de ser compatible con este tipo de aplicaciones y se apoyará en ésta en la medida de lo posible para facilitar o mejorar el rendimiento de la aplicación.

- *E-mail de contacto: IvanMiguel*.Cano@e-campus.uab.cat
- *Mención realizada: Ingeniería de Computación.*
- *Trabajo tutorizado por: Jordi Roig de Zárate (MiSE)*
- *Curso 2016/17*

2 Objetivos

El objetivo es desarrollar una aplicación para dispositivos móviles, para la plataforma Android **[5]** con escalabilidad a otras plataformas como iOS o Windows Phone, la cual deberá de ser capaz de verbalizar textos a partir del texto en una imagen. Debido al público objetivo se deberán de tener en cuenta todas las opciones de accesibilidad posibles y ser fácil e intuitiva para facilitar su uso.

Cómo se dará la opción de estar disponible en diferentes plataformas, se hará uso de Xamarin **[1]**, un framework para el lenguaje de programación de Microsoft C# **[2]**, el cual permite programar para diferentes plataformas con un código común con un posterior compilado a nativo. Lo que facilita muchísimo el desarrollo y mantenimiento de las aplicaciones.

También se plantea hacer uso de las librerías para visión por computador OpenCV. OpenCV implementa multitud de funcionalidades para el tratamiento de imágenes y matrices y es muy utilizado en la visión por computador. OpenCV está desarrollado en C y C++ bajo una licencia BSD (Berkeley Software Distribution) lo cual lo hace gratuito y multiplataforma.

El inconveniente que presenta esta librería, es que no está disponible para el lenguaje C#. Por lo que para poder utilizar las funciones de OpenCV en C# será necesario utilizar EmguCV **[6]**, una librería Cross-Platform desarrollada en .NET la cual permite llamar a los métodos de OpenCV desde los lenguajes de programación de Microsoft, entre ellos C#.

Mediante la librería EmguCV se espera realizar todas las operaciones necesarias para el tratamiento de las imágenes y la posterior extracción del texto haciendo uso de funciones OCR. También se quiere introducir una funcionalidad a la aplicación para que ella sola realizando un análisis previo de lo que capte la cámara tome fotografías automáticamente. Esta funcionalidad puede ser muy beneficiosa para la aplicación, ya que el target de usuarios al que está destinada difícilmente podrá tomar buenas fotografías.

2.1 Funcionalidades de la aplicación

Toma de fotografías automática: La idea es analizar las imágenes que capta la cámara mediante la transformada de Hough, si se detecta un rectángulo, el cual podría ser una hoja de papel con texto, tomar una fotografía. A continuación, se acotará el área de la fotografía en busca del texto, tratando de encontrar líneas horizontales la cuales deben de representar este texto.

Normalización de las imágenes de entrada: Una vez tenemos una imagen con texto, ya sea obtenida de forma manual por el usuario, o de forma automática por el dispositivo. A continuación, esta imagen será procesada para poder obtener el texto que contiene de la forma más eficientemente posible. Estas operaciones consistirán en enderezar el texto de la imagen mediante rotaciones desde los ejes Y, X o Z. Resaltar el contraste del texto con respecto al resto de la imagen mediante el análisis de histogramas, eliminación de colores, etc… Y finalmente la eliminación del ruido restante.

Extracción y análisis del texto: Una vez la imagen de entrada haya sido normalizada, se procederá a la extracción del texto mediante funciones OCR. Este texto será posteriormente analizado para verificar si se trata de un texto con sentido o no. Ya que en caso de no tener sentido es probable que la imagen no hubiese sido suficientemente buena y se deberá de pedir al usuario que tome otra fotografía.

Operaciones sobre el texto: Una vez obtenido un texto con sentido, podremos obtener toda la información de él que el usuario quiera, como por ejemplo la lectura del texto, ya sea toda seguida o párrafo a párrafo, obtención de precios, por si la imagen se trataba de la etiqueta de precio de un producto, etc…
Traducción: Se tratará de que el texto obtenido de una imagen, pueda ser traducido al idioma de preferencia del usuario para aumentar así su comprensión. Por defecto se traducirá al idioma en el que esté el dispositivo.

Almacenamiento de los resultados: Se proveerá de una opción con la cual el usuario pueda almacenar el resultado obtenido. Este resultado podrá ser consultado a posteriori cuando el usuario desee. Los resultados podrán ser almacenados indicando un nombre a preferencia del usuario, o bien, en caso de no indicar ningún nombre, por defecto se indicará la fecha y hora actual como nombre.

Disponibilidad en varios idiomas: La aplicación estará disponible tanto en inglés como en español. Los idiomas se configurarán automáticamente en función del idioma del dispositivo.

Compatibilidad con TalkBack: TalkBack es una herramienta que incorpora accesibilidad en el teléfono para ayudar a las personas ciegas. Cambia la experiencia de usuario del dispositivo de forma que sea posible usarlo sin necesidad de ver. La aplicación deberá de poder trabajar en conjunto con esta herramienta.

3 Estado del Arte

Actualmente en el mercado existen diferentes tipos de aplicaciones con objetivos similares al nuestro, entre ellas destacan las siguientes:

KNFB Reader:
- Precio: $99
- Desarrollado por: K-NFB Reading Technology & Sensotec NV
- Plataforma: Android y iOS
- Vídeo demostración: https://www.youtube.com/watch?v=cS-i9rn9nao
- Link: http://www.knfbreader.com/
- Descripción: La aplicación permite ajustar o inclinar la cámara y lee el texto en voz alta a partir de una imagen. La gente con dispositivos en braille puede tomar fotos o documentos de texto y pasarlos a ese sistema de lectura casi instantáneamente.

Speech Engine:
- App general gratuita, +2.99€ por cada idioma.
- Desarrollado por: SVOX Mobile Voices
- Plataforma: Android

- Link: https://play.google.com/store/apps/details?id=com.svox.classic&hl=es
- Descripción: Combinación de más de 40 voces masculinas y femeninas que apoya a las personas mediante la lectura en voz alta de sus textos, e-libros, traducciones, e incluso de navegación. La aplicación cuenta con soporte de voz en áreas clave como la navegación, ya que lo mantendrá guiado cuando esté conduciendo. También lee en voz alta documentos como libros electrónicos o en formato PDF, por lo que es una aplicación gratuita de ojos.

Text Fairy (OCR Text Scanner):
- Gratuita
- Desarrollada por: Renard Wellnitz
- Plataforma: Android
- Link: https://play.google.com/store/apps/details?id=com.renard.ocr&hl=es
- Descripción: Esta aplicación no está pensada para personas con discapacidad visual, por lo que no contempla opciones de accesibilidad. Permite extraer un texto a partir de una imagen, aunque no permite traducción. Tampoco lee el texto en voz alta, por lo que únicamente se trata un OCR puro. Permite añadir múltiples idiomas de forma gratuita.

Reader (OCR Speaker):
- Gratuita
- Desarrollada por: Suman Sucharit Das
- Plataforma: Android
- Link: https://play.google.com/store/apps/details?id=com.suman.news_hound&hl=es_419
- Descripción: Esta aplicación, aunque no está pensada para gente con problemas de visión, realiza algunas de las funcionalidades que queremos implementar en nuestra App. Como la normalización de imágenes para aplicar después un OCR y así extraer el texto, y detectar el idioma del texto para posteriormente leerlo. Se trata de una aplicación bastante rápida, aunque con algunos problemas de diseño, ya que su uso es realmente incómodo.

Se muestran únicamente 4 de los cientos de aplicaciones disponibles en el mercado, debido a que la mayoría de ellas realizan las mismas funciones, solamente se diferencian en el diseño de la interfaz y en su velocidad de procesamiento de imágenes, aunque en la mayoría de ellas, ésta tampoco varía mucho.

De las aplicaciones mostradas anteriormente, la más similar a nuestro proyecto es KNFB Reader, ya que implementa prácticamente todas las funcionalidades que se quieren incorporar a nuestra aplicación y algunas más. También hay que añadir que se trata del fruto de seis años de trabajo y no de unos pocos meses.

En el mercado existen infinidad de OCRs y Scanners de texto a partir de imágenes, pero pocos de ellos contemplan opciones de accesibilidad al estar pensadas para un público diferente al nuestro. Muchas de éstas aplicaciones además se muestran lentas e ineficientes, requieren de imágenes lo más perfectas posible.

4 Metodología

La metodología de desarrollo de software que se ha seguido ha sido puramente "Incremental", la cual consiste en una estrategia para controlar la complejidad y los riesgos, desarrollando una parte del software reservando el resto de aspectos para el futuro.

Los principios básicos del desarrollo incremental son:

- Una serie de mini-Cascadas se llevan a cabo, donde todas las fases de la cascada modelo de desarrollo se han completado para una pequeña parte de los sistemas, antes de proceder a la próxima.
- Se definen los requisitos antes de proceder con lo evolutivo, se realiza un mini-Cascada de desarrollo de cada uno de los incrementos del sistema.
- El concepto inicial de software, análisis de las necesidades, y el diseño de la arquitectura y colectiva básicas se definen utilizando el enfoque de cascada, seguida por iteraciones de prototipos, que culminan en la implementación del prototipo final.

De forma que se ha avanzado paso a paso, empezando por las funcionalidades más prioritarias y siguiendo con los aspectos más relevantes en función del tiempo y las herramientas disponibles.

Esta metodología ha resultado perfecta, ya que ha permitido ir probando las distintas funcionalidades de la aplicación y en caso de ser necesario modificarlas o añadir mejoras sobre la marcha, adaptando así ideas que en un principio no se habían contemplado.

5 Planificación

En la planificación del proyecto no se contempla la redacción de la documentación como una tarea propiamente dicha, ya que se realizará a lo largo del desarrollo del proyecto.

Es posible consultar la tabla con la planificación de las tareas del proyecto en el **Apéndice 1**.

6 Cambios con respecto a los objetivos iniciales

Durante el desarrollo de la aplicación, se han encontrado múltiples herramientas que han facilitado mucho el trabajo a realizar y que no se tenían en consideración durante el inicio del proyecto.

6.1 Versión para iOS

En un principio se pensó en desarrollar una aplicación que estuviese disponible tanto para las plataformas Android como iOS, y por lo tanto se decidió utilizar el framework adquirido recientemente por Microsoft, Xamarin. Xamarin permite el desarrollo de aplicaciones Cross-Platform por lo que facilita mucho la reutilización de código y la escalabilidad entre aplicaciones de distintas plataformas.

Pero debido a problemas de tiempo, a la falta de conocimientos sobre el desarrollo de aplicaciones para iOS y a la falta de recursos, ya que para desarrollar en iOS es necesario un ordenador MAC y en Xamarin no es una excepción, ya que para realizar la compilación se debe de conectar a un equipo MAC remotamente, se decidió priorizar en la optimización de las funcionalidades de la aplicación para una única plataforma y dejar de lado la versión para iOS.

Una de las funcionalidades extra que se decidió incorporar y que no se planteó en un principio, es la toma de fotografías de forma automática.

6.2 Toma de fotografías automática

Una de las ideas que se quiso incorporar en la aplicación y que mejoraría sustancialmente la experiencia de usuario, es hacer que el propio dispositivo tomase las fotografías automáticamente en cuando detectase que lo que está captando es una hoja de papel con texto. Aunque no necesariamente deba ser un folio de papel, cualquier otro contenedor de texto con forma cuadrada o rectangular que resaltase con respecto al fondo sería válido.

Esta funcionalidad resultaría tremendamente cómoda para nuestros usuarios, ya que les permitiría tomar imágenes sin preocuparse de colocar el teléfono en la posición correcta. Es un hecho que, para nuestro público objetivo, realizar una buena fotografía para poder captar el texto que contenga no es una tarea trivial y esta función les facilitaría mucho las cosas.

La idea es mediante las funciones de la librería EmguCV aplicar la transformada de Hough sobre las imágenes que está captando la cámara, analizar si se encuentra un cuadrado o rectángulo que resalte sobre el fondo y en caso afirmativo capturar esa imagen para extraer el texto.

Uno de los problemas que nos podemos encontrar con este procedimiento, es que, si tratamos de capturar un folio al completo, es muy posible que el texto se vea tan pequeño que no sea legible. Por eso mismo antes de pasar a la extracción del texto, deberemos de acotar el área donde el texto se encuentra, mediante la misma transformada de Hough podríamos observar el área donde se acumulan más líneas horizontales, las cuales podemos suponer que indicarían la existencia de texto y recortar esa área de la fotografía para utilizarla en la extracción del texto.

Esta parecía una funcionalidad muy interesante para incorporar en la aplicación, pero debido a múltiples motivos ha sido imposible de incorporar en nuestro proyecto.

Para poder entender un poco mejor en qué han consistido los problemas encontrados, se debe entender como está formado un proyecto en Xamarin. Los proyectos en Xamarin están compuestos de otros proyectos, cada uno de ellos destinado a una plataforma diferente a excepción del proyecto portable, donde se programa todo el Core común entre las aplicaciones. Por lo tanto, nuestro proyecto a su vez está compuesto de 3 proyectos diferentes, uno destinado a la compilación de aplicaciones para Android, otro para iOS y otro común para todas las plataformas y que es donde se han programado todas las funcionalidades de la aplicación.

El primer problema encontrado ha sido que la librería EmguCV no es compatible con el proyecto portable que es común para todas las plataformas, debido a que utiliza el framework .NetPortable el cual no incorpora ninguna de las librerías gráficas de .NET, para poder ser compatible con el resto de proyectos. EmguCV se alimenta de DLLs gráficas que .NetPortable no incorpora y que tampoco es posible utilizar aunque las importemos manualmente. Debido a este problema de incompatibilidad pasamos a implementar esta funcionalidad en cada una de las plataformas por separado, lo cual tampoco suponía un gran problema debido a que finalmente solo se ha desarrollado para Android.

En el proyecto Android nos encontramos que este trabaja con Mono, el cual incorpora las bibliotecas de clase que proveen un conjunto de facilidades para desarrollar aplicaciones Android en C#. Gracias al framework Mono podemos desarrollar aplicaciones para Android como si de Java se tratase. El problema lo encontramos cuando al trabajar con la librería EmguCV, para realizar casi cualquier tipo de llamada, necesitamos tener a disposición el objeto Point que está incluido dentro de la librería System.Drawing.dll de .Net y que Mono también incorpora mediante su propia librería System.Drawing, con la mala suerte de que el que incorpora Mono no es compatible con el que necesita EmguCV, por lo que EmguCV nos pide incluir la librería System.Drawing de .Net para po-

der usarla, pero al incluir esta librería se produce una colisión entre paquetes que no es posible solucionar.

Se ha intentado excluir de alguna forma la librería System.Drawing de Mono para poder evitar la colisión, aunque sin éxito.

También se ha intentado crear una biblioteca de clases para Android de forma que podamos externalizar las llamadas conflictivas a otro proyecto, pero el problema persiste.

Otra de las medidas que se ha probado, es crear un proyecto externo que utilice el framework .Net convencional de forma que podamos trabajar con EmguCV, pero ha sido imposible incluir un proyecto dentro de otro.

En cada caso, se ha probado a utilizar versiones diferentes de EmguCV, desde la más actual la versión 3.2 a la más antigua que se ha encontrado la 2.4. La versión 2.4 se consiguió compilar con éxito desde el proyecto portable, que es el único que no daba conflicto de librerías, pero al ejecutar la aplicación en el teléfono, ésta se cerraba inmediatamente sin indicar error alguno. Al final se dedujo que esta versión está demasiado obsoleta para poder usarse.

6.3 Extracción del texto

En un principio se implementó un OCR mediante la librería EmguCV, donde se aplicaba la normalización y la extracción de texto en un tiempo aceptable, pero se mostraba ineficiente al tratar de leer fuentes que no fuesen las más comunes y al intentar reconocer algunos caracteres especiales, por lo que se optó por externalizar esta funcionalidad mediante un servicio web mucho más eficiente y rápido.

Este servicio se trata de las APIs de visión por computador desarrolladas por Microsoft, las Microsoft Cognitive Services, que están disponibles de forma gratuita para todos los desarrolladores de software que estén interesados.

Entre estas APIs se encuentra un OCR **[3]** (**O**ptical **C**haracter **R**ecognition de sus siglas en inglés) la cual nos permite la extracción del texto de una imagen de forma muy sencilla, ya que la propia API aplica normalización, por lo que además de múltiples filtros para hacer resaltar el texto, también lo ajusta horizontalmente, de forma que podemos tomar una foto de un texto del revés o en vertical y ella sola la rota sin ningún tipo de problema. Además, es capaz de reconocer caracteres de entre 21 idiomas diferentes.

Por lo tanto, gracias a esta API es posible evitar hacer uso de las librerías de EmguCV ya que ahora ya no será necesario aplicar un tratamiento previo a las imágenes.

Si bien es cierto que esta API facilita mucho el trabajo a realizar, en su versión gratuita incorpora pequeñas limitaciones. Ya que el número de peticiones al OCR está limitado a cuatro mil fotos al mes, y además éstas no pueden tener un tamaño superior a 4 MB.

Además, al tratarse de una API online es indispensable tener acceso a internet, por lo que si no se dispone de una buena conexión el proceso de captación de texto puede volverse un poco lento.

7 Problemas encontrados

Además de los problemas comentados en el punto anterior, tanto para la incorporación de la versión para iOS como para la implementación de la toma de fotografías automática, los cuales han afectado a los objetivos iniciales marcados para el proyecto, también se han encontrado otros problemas, menos graves y que se si se han podido solucionar.

7.1 Traductor

Inicialmente se probó a utilizar la API de traducción de Microsoft, Microsoft Translator Text API, para realizar las traducciones de texto, ya que dispone de auto-detección de idioma de entre unos 45 idiomas diferentes y es muy sencilla de obtener y utilizar en Xamarin debido a que todo pertenece a Microsoft. El problema de usar esta API es que no es gratuita, si bien es cierto se puede obtener una versión de prueba gratuita, pero esta está limitada a dos millones de caracteres, cantidad que en un principio pareció más que suficiente pero que se consumió rápidamente al realizar varias pruebas.

Debido a esto, se buscó una alternativa, y se pensó en utilizar la API de traducción de Google incorporada en su Google Cloud Platform **[7]**. Tener disponibles las APIs de esta plataforma no es gratuito, pero al crear una cuenta, Google nos regala 300 dólares para realizar pruebas. Puesto que los precios para el uso de esta API son de 20 dólares por cada millón de caracteres traducidos y otros 20 dólares por la detección de idioma de cada millón de caracteres, tenemos que nuestro gasto sería de 40 dólares por cada millón de caracteres, ya que será necesario tanto detectar el idioma de origen como traducir el texto. Realizando un sencillo cálculo sabemos que con los 300 dólares de los que disponemos, podríamos traducir unos 7,5 millones de caracteres, una cifra que no se esperaba alcanzar hasta la finalización del proyecto, por lo que se decidió utilizar esta API para la traducción.

Sin embargo, para poder utilizar la API hay que autenticarse cada hora y debido a la complejidad que supone la

autenticación de las credenciales en dispositivos móviles, ya que la verificación se realiza mediante un fichero JSON que deberían de tener todos los dispositivos móviles en los que se instale la aplicación, se decidió que era inviable.

Así que finalmente se optó por lo menos sencillo de implementar, pero que nos proporciona un mejor servicio ya que es gratuito y su única limitación depende de la red a la que estemos conectados, que consiste en crear un cliente web capaz de realizar peticiones a un servicio en la nube mediante una URL y a continuación descargar su contenido HTML en formato texto.

De esta forma, únicamente realizamos peticiones a la web del Google Translate indicando el texto a traducir y el par de idiomas de entrada y salida y nos descargamos el contenido para obtener el texto traducido. Un ejemplo de URL que construiríamos para traducir la palabra "hello" al español con auto detección de idioma sería el siguiente: http://www.google.com/translate_t?hl=en&ie=UTF8&text=hello&langpair=auto|es

Gracias al uso de la web de Google, disponemos de un traductor gratuito, sin limitación de caracteres y que además es capaz de reconocer y traducir textos de entre 97 idiomas diferentes.

7.2 Interfaz de la aplicación (Cámara)

En cuanto a la interfaz de la aplicación, encontramos que al necesitar hacer uso del hardware para poder mostrar lo que captura la cámara del dispositivo en tiempo real, siendo que en Android cada dispositivo maneja la cámara de una forma distinta y que en iOS la cámara no tiene nada que ver, se dificulta muchísimo escribir un código que sea común entre ambas plataformas.

Para poder generar un código que fuese común entre plataformas se intentó usar un plugin para Xamarin llamado Custom Camera Plugin **[4]**, el cual es muy sencillo de implementar y parecía que funcionaba muy bien, pero consta de algunas pegas, la mayor de ellas es que no implementa el auto-focus, y la otra es que tampoco implementa el flash de la cámara por lo que difícilmente podríamos tomar buenas fotografías. Debido a estas limitaciones se descartó su uso.

Finalmente viendo que no sería posible implementar un código común para gestionar la cámara, se optó por implementar una solución propia para cada plataforma independientemente, así que la aplicación acabó teniendo una interfaz completamente "nativa" en cada plataformas y compartiría todas las funcionalidades, como la extracción del texto e incluso el speaker para leerlo.

8 Presentación de Resultados

Para verificar el correcto funcionamiento de la captación de textos en imágenes, se han seleccionado diferentes objetos que contienen texto para realizar algunas pruebas.

8.1 Obtención de texto

Prueba 1: Texto abundante.

Se puede consultar el input utilizado en el **Apéndice 2**.

Esta es una fotografía tomada a un libro sobre patrones de diseño de software, donde podemos observar que hay bastante texto (para ser captado en una única foto), donde encontramos distintas fuentes y tamaños de texto.
El resultado obtenido es el que se puede observar en el **Apéndice 3**.

Como se puede observar el resultado obtenido es bastante satisfactorio, ya que salvo en algunas palabras, el texto ha sido reconocido completamente. Los fallos en algunas palabras podemos comprobar que son debido al exceso de iluminación en la imagen.

El tiempo transcurrido desde la toma de la foto a la obtención del texto, ha sido de 06.537 segundos, y aunque el dispositivo está conectado a una red de fibra óptica de 300 Mb de conexión, teniendo en cuenta la cantidad de texto, podemos decir que se trata de un tiempo de espera más que aceptable.

Prueba 2: Distintas fuentes y varios colores de fondo

Se puede consultar el input utilizado en el **Apéndice 4**.

Esta es una fotografía a la portada del libro anterior, la cual dispone de tamaños de fuente variados y un color de fondo un poco más confuso que en el caso de un folio normal.

El resultado obtenido es el que se puede observar en el **Apéndice 5**.

En este caso el resultado es perfecto, ya que ha captado tanto el título y subtítulo del libro, como el nombre del autor sin ningún problema.

Esta vez el tiempo transcurrido durante la operación es de 05.609 segundos, más rápido que en el caso anterior seguramente debido a que la imagen contiene menos texto.

Prueba 3: Texto torcido y caracteres especiales.

Se puede consultar el input utilizado en el **Apéndice 6**.

En este caso se ha realizado una fotografía con el texto torcido de forma deliberada. El texto pertenece a las especificaciones de un adaptador de corriente, el cual está en alemán y contiene algunos caracteres especiales.

El resultado obtenido es el que se puede observar en el **Apéndice 7**.

Como se puede observar, en este caso el resultado exceptuando la interpretación de "100V" por "1 oov" ha sido perfecto. El algoritmo ha sabido interpretar los "@" correctamente y los signos de puntuación propios del alemán.

El resultado ha sido obtenido en 04.697, probablemente en un tiempo anterior a la prueba anterior aun conteniendo más texto, debido a que en este caso el contraste de las letras sobre el fondo está mucho más acentuado.

Prueba 4: Fuentes especiales y poco contraste

Se puede consultar el input utilizado en el **Apéndice 8**.

Esta vez se ha optado un por un folleto informativo, donde aparecen varias fuentes especiales sobre un fondo muy sobrecargado de colores y formas.

El resultado obtenido es el que se puede observar en el **Apéndice 9**.

Como era de esperar, el resultado no ha sido perfecto debido a que parte del texto de la imagen ha pasado desapercibido, probablemente debido al tipo de fuente, ya que puede resultar confusa incluso para el ojo humano. Aunque si se ha detectado el texto que resulta más legible de imagen.

El resultado ha sido obtenido en 03.488 segundos.

Prueba 5: Caracteres orientales.

Esta vez se ha tomado una foto a una caja con indicaciones en chino para realizar la prueba.

Se puede consultar el input utilizado en el **Apéndice 10**.

Aun siendo un texto completamente con caracteres especiales, están situados sobre fondo blanco por lo que resultan fácilmente legibles.

El resultado obtenido es el que se puede observar en el **Apéndice 11**.

Como se puede apreciar, el resultado obtenido es perfecto. El algoritmo ha sido capaz de reconocer cada uno de los logogramas sin aparente problema.

El resultado ha sido obtenido en 02.921 segundos, todo un récord.

Prueba 6: Lectura de viñetas

Se puede consultar el input utilizado en el **Apéndice 12**.

Para esta prueba se optado por realizar una fotografía a la viñeta de un manga en español. En este caso los caracteres son fácilmente reconocibles, aunque en una fuente estilo "comic sans".

El resultado en esta ocasión es el que se puede observar en el **Apéndice 13**.

Como se puede ver, y en contra de los resultados esperados, esta vez el algoritmo no ha sido capaz de reconocer el texto de la imagen. Aunque ha sido capaz de reconocer la existencia de un texto, no ha sabido interpretar la fuente, dando como resultado una amalgama de caracteres aparentemente aleatorios.

El tiempo transcurrido durante la obtención de este resultado ha sido de 04.565 segundos, superior a los dos casos anteriores.

Comparativa de resultados

A continuación, se muestra una tabla comparativa con los resultados obtenidos en cada prueba, se tiene en cuenta el tiempo transcurrido hasta obtener un resultado, y el porcentaje de texto correctamente captado. Finalmente se ha puntuado el resultado de cada prueba en función de los dos datos anteriores.

	Tiempo	% Texto Obtenido	Puntuación
Prueba 1	06.537s	95%	9 / 10
Prueba 2	05.609s	100%	9.7 / 10
Prueba 3	04.697s	98%	9.6 / 10
Prueba 4	03.488s	45%	4.5 / 10
Prueba 5	02.921s	99%	10 / 10
Prueba 6	04.565s	0.2%	0 / 10

8.2 Traducción de Textos

Para estas pruebas utilizaremos los mismos elementos que en las pruebas anteriores, incluyendo las que tienen el texto en castellano, ya que el traductor también realiza un poco de corrección se espera verificar si ésta mejora los resultados obtenidos anteriormente o si por el contrario los empeora

Traducción prueba 1

Para esta prueba se ha utilizado el resultado obtenido en la primera prueba de obtención de texto que se puede consultar en el **Apéndice 3**.

En este caso siendo un texto en castellano, aplicar traducción ha sido indiferente, ya que como se puede consultar en la imagen del **Apéndice 14** el resultado es exactamente el mismo antes y después.

Traducción prueba 2

Para esta prueba se ha utilizado el resultado obtenido en la segunda prueba de obtención de texto que se puede consultar en el **Apéndice 5**.

En este caso al igual que en el caso anterior, se trata de un texto en castellano, y como era de esperar aplicar la traducción también ha sido indiferente, ya que como se puede consultar en la imagen del **Apéndice 15** el resultado es exactamente el mismo antes y después.

Traducción prueba 3

Para esta prueba se ha utilizado el resultado obtenido en la tercera prueba de obtención de texto que se puede consultar en el **Apéndice 7**.

En este caso se trata de un resultado con el texto en alemán y por lo que se puede observar en la imagen del **Apéndice 16**, el texto traducido es bastante legible en general a excepción de la palabra "Linder" que al parecer el traductor no ha sabido interpretar.

Traducción prueba 4

Para esta prueba se ha utilizado el resultado obtenido en la cuarta prueba de obtención de texto que se puede consultar en el **Apéndice 9**.

En este caso se trata de un texto en catalán, que, aunque pueda ser fácilmente interpretable por una persona, debido a los errores gramaticales que contiene, no lo es tanto para una máquina por lo que la traducción obtenida que se puede consultar en el **Apéndice 17**, está lejos de ser un resultado óptimo.

Esta prueba a su vez ha sido de gran utilidad, ya que podemos darnos cuenta de que pasando un auto corrector a los textos antes de traducirlos, podríamos obtener mejores resultados.

Traducción prueba 5

Para esta prueba se ha utilizado el resultado obtenido en la quinta prueba de obtención de texto que se puede consultar en el **Apéndice 11**.

En este caso se trata de un texto en chino, y el resultado obtenido es el que se puede observar en el **Apéndice 18**. El traductor en este caso ha hecho un trabajo regular, ya que si alguien lee el texto obtenido fácilmente comprobará como este en muchos casos carece de sentido. También hemos de recordar que los pictogramas en la escritura china son algo parecido a ideas o conceptos por lo que no debe de ser nada sencillo realizar una traducción fiel.

Aun así, pasar de no entender nada de un texto a entender aunque sea un 15% de lo que se indica, es un cambio significativo.

Traducción prueba 6

Para finalizar, en esta prueba se utilizado el resultado obtenido en la sexta prueba de obtención de texto que se puede consultar en el **Apéndice 13**.

En este último caso se trata de un texto sin ningún significado, únicamente es una amalgama de caracteres sin aparente sentido. Y como era de esperar, la traducción de este texto tampoco tiene sentido, tal como se puede comprobar a través de la imagen del **Apéndice 19**.

9 CONCLUSIONES

9.1 Sobre Xamarin

Como se ha comentado anteriormente, Xamarin es un framework de Microsoft destinado al desarrollo de aplicaciones Cross-Platform bajo lenguaje C#. Este sistema resulta muy útil para aplicaciones destinadas a ser multiplataforma, donde únicamente será necesario desarrollar un solo código compatible para varias plataformas.

Pero no todo es tan simple, en muchos casos, si no se trata de aplicaciones sencillas, será necesario implementar gran parte de la gestión de la interfaz de usuario en cada una de las plataformas por separado, ya que, el tratamiento del hardware de los distintos dispositivos suele ser completamente diferente. En el caso de la aplicación que hemos desarrollado, al hacer uso de la cámara, nos hemos visto obligados a implementar la interfaz al completo como si la aplicación fuese solamente para Android. Añadiendo además el hándicap de estar desarrollando mediante un IDE que no está diseñado exclusivamente para ese entorno y que por lo tanto aporta menos soporte al desarrollo en esa plataforma.

Por lo tanto, aun siendo un entorno de desarrollo Cross-Platform, en muchos casos, nos veremos obligados a saber desarrollar para cada una de las plataformas. Aunque si es verdad que se trata de un entorno muy fácilmente escalable y se nos permite programar en C# como si de Java, Objective-C o Swift se tratase.

Por otro lado, comentar que, aunque en la mayoría de casos la sintaxis sea prácticamente la misma, también incorpora algunos cambios, en muchos casos sutiles, aunque en otros no tanto, llegando a cambiar sustancialmente un código que escribiríamos en Java a como se ha de escribir en C#, dificultando el desarrollo al obligar al programador a conocer todos estos cambios en el lenguaje.

También se trata de un sistema relativamente nuevo donde falta depurar mucho el desarrollo y tiene grandes carencias de información oficial, aunque por otro lado dispone de una comunidad de desarrolladores bastante grande y gracias a esto se puede encontrar mucha información útil en foros y otras webs no oficiales.

Podemos concluir finalmente que Xamarin no facilita tanto el desarrollo de aplicaciones, aunque en casos donde no se deban de utilizar APIs internas para gestionar el hardware del dispositivo en cada plataforma si permite reducir mucho el tiempo invertido en el desarrollo. Así que para aplicaciones sencillas Xamarin es una muy buena opción a tener en cuenta para desarrollar aplicaciones para distintas plataformas, pero no tanto para aplicaciones más complejas donde sea necesario el uso intensivo del hardware del dispositivo. Aunque sí que facilita muchísimo en ambos casos las tareas de actualización y mantenimiento de las aplicaciones.

9.2 Sobre EmguCV

Sobre EmguCV no hay mucho que decir que no se haya dicho ya. EmguCV está pensado para poder trabajar con las librerías de OpenCV en entornos de desarrollo en .Net, pero no parece tener en cuenta su uso para aplicaciones en Xamarin, o por lo menos desde nuestra experiencia.

Sabiendo que Xamarin fue obtenido recientemente por Microsoft podemos deducir que aún le queda mucho recorrido y es probable que en un futuro próximo sea posible utilizar EmguCV sin padecer tantos problemas como los que nos hemos encontrado.

9.3 Sobre la obtención de textos

A raíz de todas las pruebas realizadas, podemos concluir que en la mayoría de casos obtenemos resultados bastante satisfactorios en tiempos más que razonables.

Claro que la aplicación consta de una dependencia muy fuerte en cuanto a la calidad de las imágenes. Para imágenes desenfocadas, con mala iluminación o con textos con fuentes complejas, en la mayoría de los casos los resultados que se obtienen dejan mucho que desear.

Teniendo una dependencia tan fuerte de la calidad de las imágenes que tome el usuario, y que, a nuestros usuarios, debido a sus problemas de visión, se les dificulta esta tarea, deberíamos añadir algún tipo de mecanismo para intentar garantizar que las fotografías que tomen, tengan la mejor calidad posible.

9.4 Sobre la traducción de texto

Las traducciones de textos en general para idiomas parecidos como pueden ser todos los idiomas de origen latino, donde la gramática es similar suelen ser bastante aceptables, en cambio para idiomas que tiene muy poco o nada que ver, la traducción obtenida se puede volver difícil de entender, aunque claro pasar de entender un 0% del texto a entender un 20 o un 30% es un paso significativo.

La traducción de un texto tiene una fuerte dependencia de la ortografía del mismo. Si tratamos de traducir un texto el cual tiene errores orográficos severos como cambios de una “I” por un “1” o una “O” por un “0” como nos puede suceder debido a que el OCR se basa en las formas de los caracteres que interpreta, difícilmente la traducción podrá ser buena.

Así pues, deberíamos introducir alguna herramienta para corregir las palabras con errores ortográficos en la medida de lo posible, y de esta forma mejorar en gran medida los resultados que se obtienen tanto en la interpretación del texto mediante el OCR como en su posterior traducción. Obteniendo así textos lo más comprensibles posible.

AGRADECIMIENTOS

Quiero agradecer a mi tutor del TFG Jordi Roig por su asesoramiento en el ámbito de la accesibilidad en los dispositivos electrónicos, así como el haber hecho de Beta Tester de la aplicación. También quiero agradecer a Jorge Bernal sus consejos sobre visión por computador, que, aunque no se han podido aplicar debido a problemas técnicos, si me han servido para entender mucho mejor estos conceptos.

BIBLIOGRAFÍA

[1] Can Bilgin. (Marzo 2016). Mastering Cross-Platform Development with Xamarin (1ª Ed). Liverly Place, Birmingham, UK. Packt Publishing Ltd.

[2] Xamarin Inc. Microsoft. (2017). Developer-Center Xamarin. Recuperado de: https://developer.xamarin.com/

[3] Microsoft (2016). Microsoft Cognitive Services - Documentation. Recuperado de: https://www.microsoft.com/cognitive-services/en-us/computer-vision-api/documentation

[4] Niels Cup (Abril 2016). Custom Camera Plugin for Xamarin (Android + iOS). Recuperado de: https://github.com/nielscup/CustomCamera

[5] Alphabet.inc. Google (2012). Android Developers. Recuperado de: https://developer.android.com/index.html

[6] MediaWiki.org (Mayo 2017). Emgu CV. OpenCV in .NET (C#, VB, C++ and more). Recuperado de: http://www.emgu.com/wiki/index.php/Main_Page

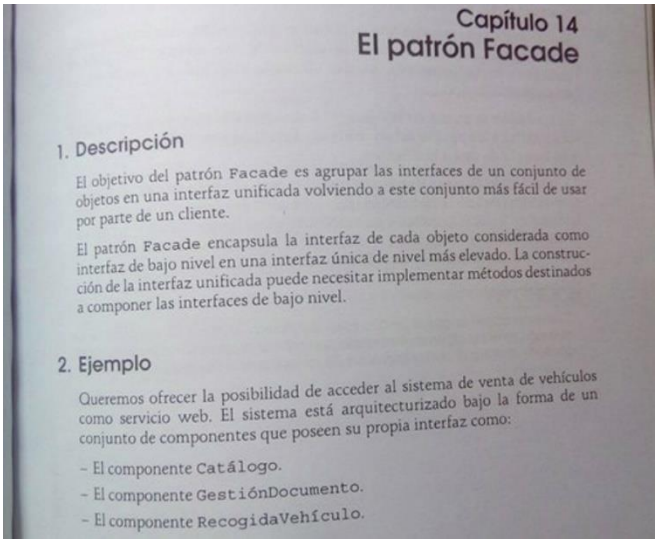
[7] Oracle and Affiliates. Google (Junio 2017). Google Cloud Platform Documentation. Recuperado de: https://cloud.google.com/docs/
.

APÉNDICES

A1. Planificación de Tareas

Fecha Inicio	Fecha Final	Tarea Realizada
12/02/2016	18/02/2016	Recolección de información sobre las herramientas a utilizar y los productos existentes en el mercado.
18/02/2016	19/02/2016	Instalación y preparación de todas las herramientas y librerías
20/02/2016	21/02/2016	Creación de una primera App simple para obtener las imágenes desde el dispositivo móvil
21/02/2016	28/02/2016	Desarrollo del OCR, para obtener el texto a partir de una imagen sencilla.
28/02/2016	07/03/2016	Implementación del traductor de texto
07/03/2016	11/04/2016	Creación de las interfaces definitivas para la App
11/04/2016	02/05/2016	Implementación de las diferentes funcionalidades de la App. Lectura texto completo, lectura por párrafos y obtención de precios.
02/05/2016	23/06/2016	Testeo, correcciones y modificaciones que se deban realizar para dejar la App lo más optimizada posible.

A2. Input prueba 1



Capítulo 14
El patrón Facade

1. Descripción

El objetivo del patrón Facade es agrupar las interfaces de un conjunto de objetos en una interfaz unificada volviendo a este conjunto más fácil de usar por parte de un cliente.

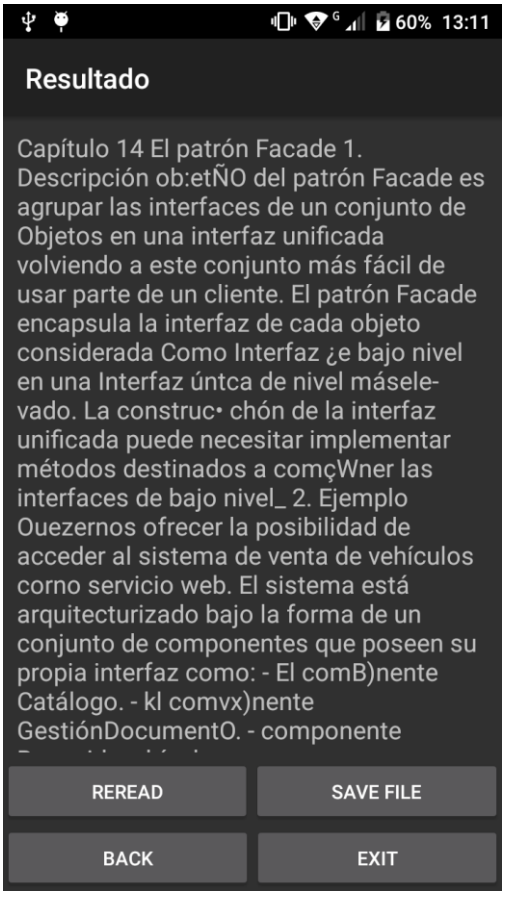
El patrón Facade encapsula la interfaz de cada objeto considerada como interfaz de bajo nivel en una interfaz única de nivel más elevado. La construcción de la interfaz unificada puede necesitar implementar métodos destinados a componer las interfaces de bajo nivel.

2. Ejemplo

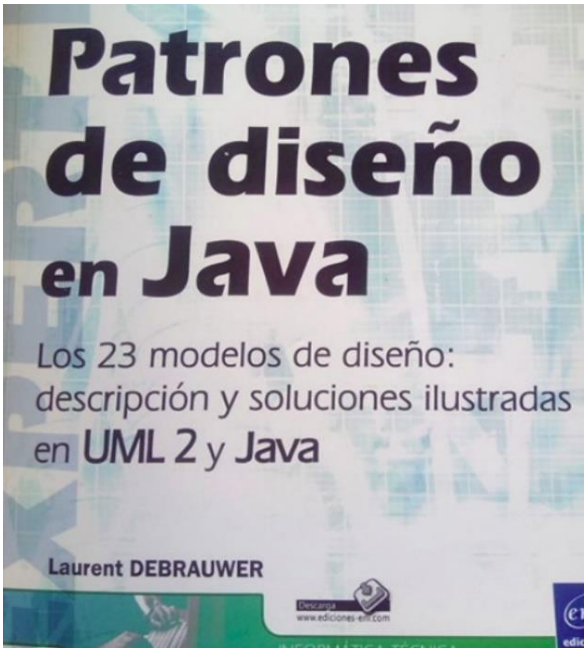
Queremos ofrecer la posibilidad de acceder al sistema de venta de vehículos como servicio web. El sistema está arquitecturizado bajo la forma de un conjunto de componentes que poseen su propia interfaz como:

- El componente Catálogo.
- El componente GestiónDocumento.
- El componente RecogidaVehículo.

A3. Output prueba 1



A4. Input prueba 2



A5. Output prueba 2

Resultado

Patrones de diseño en Java Los 23 modelos de diseño: descripción y soluciones ilustradas en UML 2 y Java Laurent DEBRAUWER

A6. Input prueba 3

A7. Output prueba 3

Resultado

Universal Reiseadapter Dieser Reiseadapter ist kompatibel mit nahezu allen Ländern weltweit und besitzt zugleich ein sehr kompaktes Design. Spezifikation • Spannung: 100 — 240V • Maximale Leistung 660w @ 1 oov 1350W @ 230V • Stecker-Arten EU, Aus. Kompatible Linder (Beispiele) • Australien, Bulgarien, China. Dänemark. Japan. Spanien, USA und viele mehr

A8. Input prueba 4

A9. Output prueba 4

Resultado

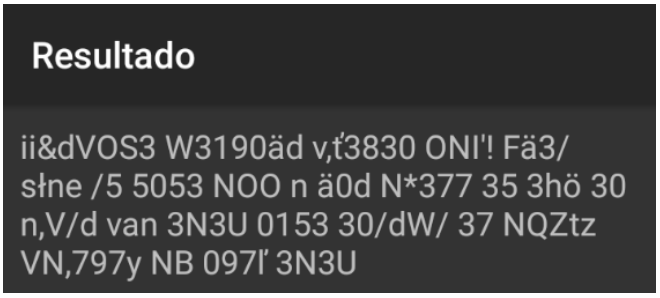
VALDAR ELTEU BITL'LET TE PREMI T-jove•

A10. Input prueba 5

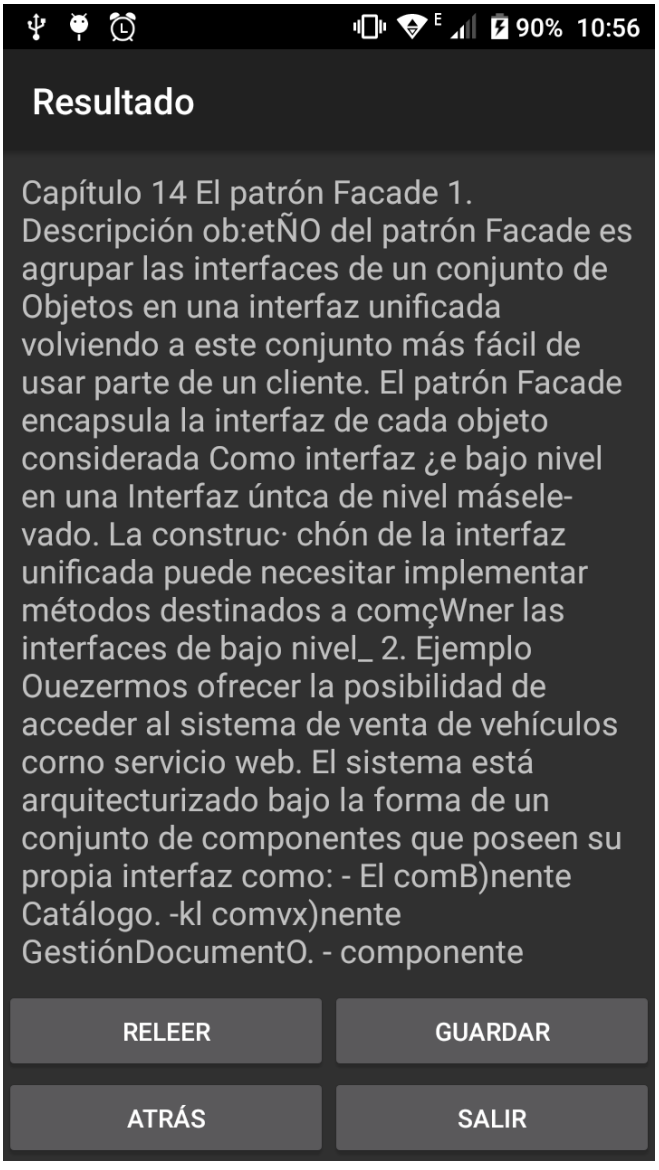
A11. Output prueba 5

A12. Input prueba 6

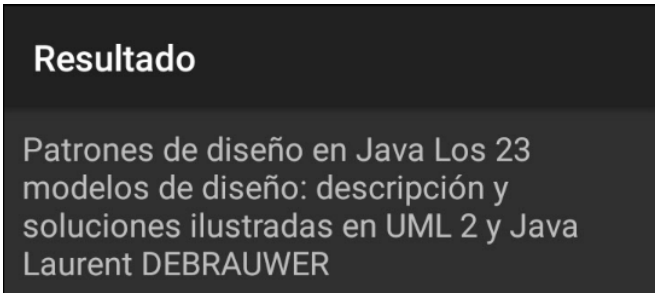
A13. Output prueba 6



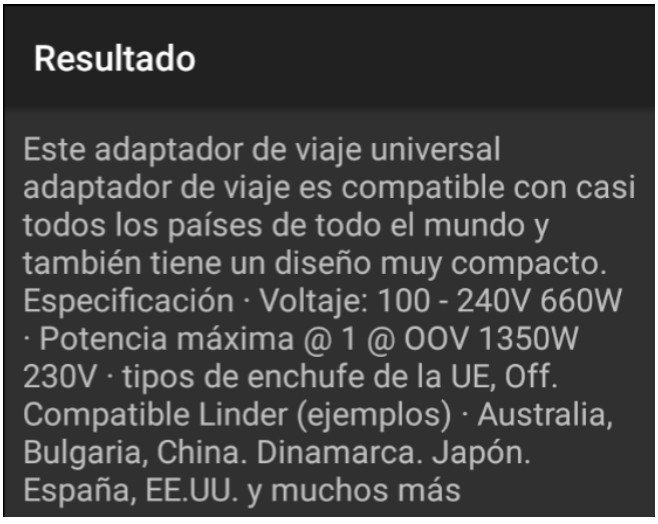
A14. Output traducción 1



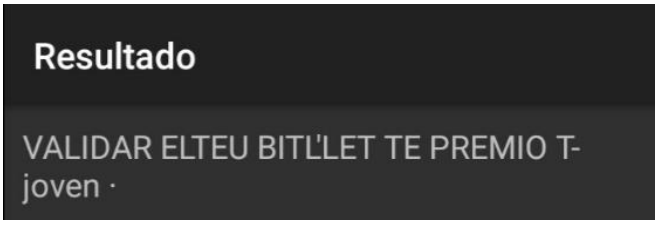
A15. Output traducción 2



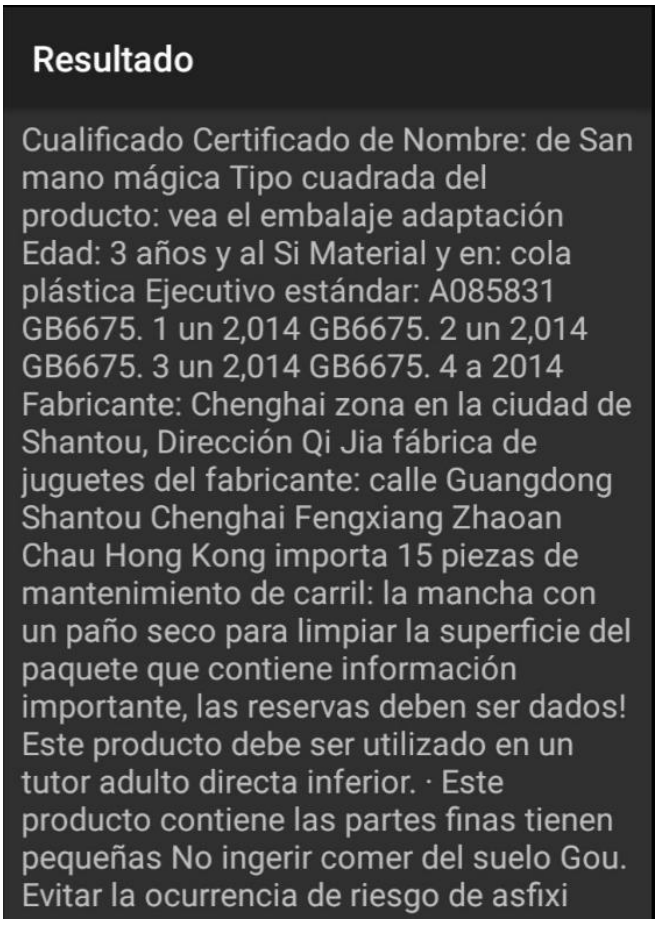
A16. Output traducción 3



A17. Output traducción 4



A18. Output traducción 5



A19. Output traducción 6