UNIVERSITAT OBERTA DE CATALUNYA

MÁSTER UNIVERSITARIO EN SOFTWARE LIBRE

ESPECIALIDAD: DESARROLLO DE APLICACIONES

"SOFTWARE DE ADQUISICIÓN DE IMÁGENES Y RECONOCIMIENTO ÓPTICO DE CARACTERES PARA ANDROID"

TRABAJO FINAL DE MÁSTER

ENERO – 2013

AUTOR: Jaime Navarro Santapau

CONSULTORES: Gregorio Robles Martínez

Roberto Gil Casas

COPYRIGHT

Autor

Jaime Navarro Santapau

Para cualquier comentario o sugerencia contactar con jaimenavarrosantapau@edu.uoc

Publicación

Contenido actualizado en Enero de 2013.

Documento creado con OpenOffice.org 3.2.0.

AGRADECIMIENTOS

A la Universidad Oberta de Catalunya que conjuntamente con la empresa Tempos21 me han brindado la oportunidad de realizar este trabajo final de máster. Especialmente a los consultores del mismo Gregorio Robles Martínez y Roberto Gil Casas, por su inestimable ayuda y atención.

A todos los compañeros del máster por hacerme más llevadero y ameno el trabajo y la estancia virtual, y a los amigos que he conocido durante mis estudios en esta universidad.

No puedo finalizar este apartado de agradecimientos, sin hacer una mención especial a mi novia por el apoyo incondicional recibido durante todo este máster.

RESUMEN DEL PROYECTO

El presente proyecto final de máster parte de la propuesta realizada por la empresa Tempos21, donde se recoge la necesidad de crear una librería de reconocimiento óptico de caracteres para la plataforma Android, por lo que esta librería podrá ser utilizada por diferentes aplicaciones ejecutadas en terminales móviles que cuenten con este sistema operativo.

La utilidad de esta librería se centra en que sea capaz de llevar a cabo el reconocimiento óptico de caracteres sobre imágenes que hayan sido capturadas por la cámara principal del teléfono móvil.

Como punto de partida en la creación de nuestra librería, se han utilizado algunas de las librerías de reconocimiento óptico de caracteres existentes en diferentes proyectos de software libre.

También se ha creado una aplicación basada en software libre, donde se implementa el proceso de adquisición de imágenes y las llamadas necesarias a nuestra librería para realizar el proceso de reconocimiento óptico de caracteres sobre las imágenes capturadas.

Finalmente se ha creado un manual paso a paso para poder generar un patrón de referencia de forma manual, que sea capaz de adaptarse a unas características concretas de estilos de fuente, tipografías, caracteres, etc. Partiendo de este manual se ha generado un patrón de referencia adaptado para algunos tipos de *Tickets* de gastos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUCCIÓN

1. INTRODUCCIÓN

A través de la empresa Tempos21[1] se define la necesidad de crear una librería de OCR[2] (Reconocimiento Óptico de Caracteres) que pueda ser utilizada por diferentes aplicaciones en teléfonos inteligentes Android.

La utilidad de esta librería se centra en que sea capaz de llevar a cabo el reconocimiento óptico de caracteres sobre imágenes que hayan sido capturadas a través de la cámara principal del teléfono móvil.

Como punto de partida en la creación de la librería, se recomienda utilizar alguna de las librerías OCR existentes en diferentes proyectos de SL (Software Libre) como por ejemplo GOCR, The Tesseract OCR o Java OCR, entre otras.

Las aplicaciones que se podrían desarrollar haciendo uso de esta librería de reconocimiento son múltiples. En este proyecto nos vamos a centrar en generar una aplicación OCR que sea capaz de adaptarse y configurarse para realizar el reconocimiento de caracteres sobre un tipo concreto de **tickets** de gastos generados por TPV[3]**.**

Lo que se pretende es poder extraer la información del ticket de una forma automática para posteriormente utilizarla para diferentes propósitos.

1 Más información acerca de empresa Tempos21: http://www.tempos21.com/web/
2 OCR:Es el acrónico a partir del inglés de Optical Character Recognition
3 Más información acerca de terminales TPV: http://es.wikipedia.org/wiki/Terminal_punto_de_venta

1.1 OBJETIVOS

A continuación se identifican algunos de los objetivos más relevantes que se esperan alcanzar o desarrollar a lo largo del proyecto:

- Investigación de diferentes técnicas de OCR.
- Estudio del estado del arte de las librerías OCR en el SL.
- Investigación del funcionamiento de la cámara en teléfonos móviles con Android.
- Estudio de la viabilidad de adaptación y capacitación de las librerías analizadas para nuevos patrones de referencia.
- Desarrollo e implementación de la librería para ser utilizada en teléfonos Android.
- Desarrollo e implementación de una aplicación que haga uso de esta librería.

1.2 DEFINICIÓN Y PLANIFICACIÓN DE TAREAS

En este apartado se enumeran las tareas y labores definidas al inicio de este TFM mediante un plan de trabajo. En la *Tabla 1* se muestran las diferentes fases de las que consta el proyecto y el subconjunto de tareas que se han abordado para poder realizar cada una de estas fases.

Fases	Tareas	Descripción
Planificación		
	Plan de Trabajo	Definición de objetivos, tareas y planificación.
Análisis		
	Estudio inicial	Estudio del estado del arte de las librerías de SL.
	Instalación y configuración librerías	Instalación y configuración de las librerías de SL identificadas mediante el estudio anterior.
	Testing librerías	Pruebas de uso y funcionalidad de las librerías instaladas.

Diseño		
	Estudio cámara teléfono	Estudio de los diferentes modos de captura e identificación de las condiciones óptimas.
	Generación imágenes objetivo	Generación de diferentes imágenes de muestra de tickets mediante la cámara del teléfono.
	Adaptación de las librerías a nuevos patrones	Adaptar y capacitar a las librerías en el reconocimiento de las imágenes objetivo anteriores mediante nuevos patrones de referencia.
	Elección librería	Elección de la librería más apropiada mediante datos objetivos (adaptabilidad, eficacia, eficiencia, licencias, etc).
Implementación		
	Migración/Adaptación librería	Migración y adaptación de la librería elegida a la plataforma Android.
	Creación aplicación	Desarrollo de una aplicación simple que haga uso de la librería desarrollada.
	Capacitación para nuevos patrones	Proceso de creación de un patrón de referencia para capacitar a nuestra solución software en la detección y reconocimiento de tipografías de tickets.
Pruebas y documentación		
	Pruebas y documentación de la aplicación	Identificación de errores, solución de errores y documentación del código mediante la herramienta Doxygen.
	Pruebas y documentación de la librería	Identificación de errores, solución de errores y documentación del código mediante la herramienta Doxygen.
Validación		
	Objetivos alcanzados	Documentación de los objetivos alcanzados
	Evolución/mejora	Identificación de líneas de evolución/mejora

Tabla 1: Tareas definidas para la realización del TFM

En la siguiente *Tabla 2* se muestran los hitos que se establecieron al inicio del proyecto y las correspondientes fechas de finalización de cada uno de estos hitos.

ID	Hito	Fecha
H1	Final Análisis	09/10/12
H2	Final Diseño	31/10/12
H3	Final Desarrollo	03/12/12
H4	Final Pruebas	31/12/12
H5	Entrega Proyecto	06/01/13

Tabla 2: Hitos definidos

Finalmente en la *Figura 1* se muestra la evolución que ha llevado el proyecto a través de su diagrama de Gantt, donde podemos observar que no se han encontrado contratiempos importantes, por lo que la planificación inicial no ha sufrido desviaciones relevantes.

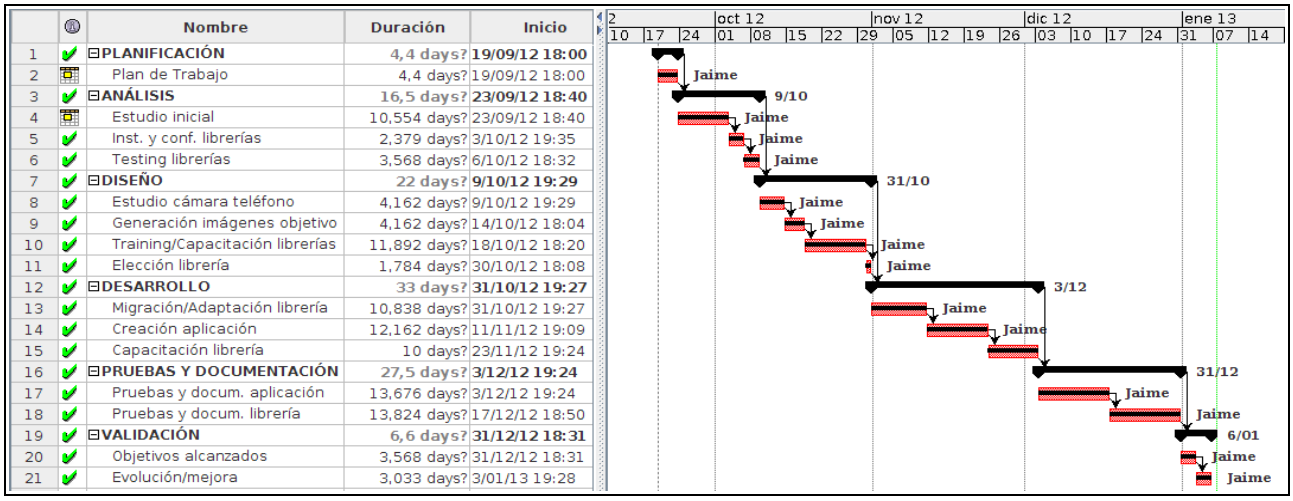

Figura 1: Diagrama de Gantt

1.3 ESTRUCTURA DE LA MEMORIA

La memoria del presente Trabajo Fin de Máster consta de un total de 6 capítulos y 4 anexos, distribuidos como se muestra a continuación, en función de su contenido:

- **CAPÍTULO 1: INTRODUCCIÓN.**

  Este capítulo presenta el proyecto describiendo las necesidades que lo fundamentan, los

objetivos que se esperan alcanzar, y las tareas que se han abordado para alcanzarlos. Por último se explica la estructura de la memoria realizada, en el presente documento.

- **CAPÍTULO 2: ANÁLISIS.**

  En esta primera fase del proyecto hemos tratado de investigar y analizar la mayor cantidad de información relacionada con los algoritmos de OCR, y también hemos recopilado información de aquellos proyectos de SL que pudieran tener relación directa e indirecta con el procesamiento digital de imágenes para conseguir extraer e identificar los caracteres existentes dentro de imágenes digitalizadas.

- **CAPÍTULO 3: DISEÑO.**

  En esta fase del TFM se han realizado las labores necesarias para definir una solución viable que cumpla con los objetivos planteados al inicio de este proyecto. Para ello debemos tener en cuenta el análisis realizado e incorporar los objetivos, características y requisitos particulares que deberá afrontar nuestra solución software.

- **CAPÍTULO 4: IMPLEMENTACIÓN**.

  En esta etapa del proyecto hemos abordado las tareas para desarrollar una solución software que dé respuesta a las necesidades explicadas y definidas durante las etapas de análisis y diseño. Por lo tanto, a partir de los resultados y conclusiones obtenidas en las fases de análisis y diseño intentaremos construir una solución software que de respuesta a los requerimientos definidos.

- **CAPÍTULO 5: CONCLUSIONES Y LÍNEAS FUTURAS.**

  Esta parte de la memoria consta de un capítulo en el que se enumeran las conclusiones que se han obtenido del trabajo realizado y se proponen posibles líneas de trabajo o de mejora.

- **ANEXO A – INFORMACIÓN ALGORITMOS OCR**

  En este anexo se presenta información general acerca del funcionamiento general de un algoritmo de reconocimiento óptico de caracteres.

- **ANEXO B – MANUAL DE USUARIO**

  En este anexo se puede ver el entorno gráfico de la aplicación, sus características y un

ejemplo de uso.

- **ANEXO C – PRESUPUESTO DEL PROYECTO**

  En este anexo se presenta un presupuesto detallado del coste empresarial que supondría el desarrollo de este proyecto.

- **ANEXO D – HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS**

  En este anexo se muestran los conocimientos utilizados o adquiridos, y las herramientas hardware y software que han sido necesarias para llevar a cabo el desarrollo de este proyecto.

- **ANEXO E – LICENCIAS DEL SOFTWARE UTILIZADO**

  En este anexo se enumeran las diferentes licencias de software libre de las que consta la solución software creada en este proyecto.

CAPÍTULO 2

ANÁLISIS

2. ANÁLISIS

En esta fase del proyecto hemos tratado de investigar y analizar la mayor cantidad de información relacionada con los algoritmos de OCR, y también hemos recopilado información de aquellos proyectos de SL que pudieran tener relación directa e indirecta con el procesamiento digital de imágenes para conseguir extraer e identificar los caracteres existentes dentro de imágenes digitalizadas.

2.1 ANÁLISIS MOTORES OCR EN SL

En este primer punto de la fase de análisis se comenta el estado del arte de los proyectos de SL más relevantes encargados de la creación, mantenimiento y evolución de software destinado a realizar labores de OCR.

Proyecto	WebSite	Ultima Versión	Fecha publicación	Ultima actualización repositorio	Escrito en	Licencia
GOCR	http://jocr.sourceforge.net/index.html	0.49	24/09/10	31/05/12	C y C++	GPL v2
JavaOCR	http://sourceforge.net/projects/javaocr/	1.101	06/06/10	06/06/10	Java	BSD License
Ocrad	http://www.gnu.org/software/ocrad/	0.21	10/01/11	22/01/12	C y C++	GPL v3
Tesseract	https://code.google.com/p/tesseract-ocr/	3.02	01/11/12	03/11/12	C y C++	Apache License 2.0
Cuneiform	https://launchpad.net/cuneiform-linux/	1.1	19/04/11	19/04/11	C y C++	Simplified BSD Lic.

Tabla 3: Motores OCR de SL

A continuación se realiza una pequeña descripción de cada uno de los proyectos de SL enumerados en la tabla anterior:

1. **GOCR** [1] [2]. Última versión publicada 0.49. Fecha última liberación 24/09/2010. Última actualización del repositorio 31/05/2012 **Licencia Robusta**, GNU Public License v2. Escrito en C y C++.

   GOCR es un programa de OCR desarrollado bajo la Licencia Pública GNU. Convierte las imágenes escaneadas que contienen texto a archivos de texto. Joerg Schulenburg inició el programa y ahora dirige un equipo de desarrolladores. GOCR se puede usar con diferentes *front-end*, lo que hace que sea muy fácil de portar a diferentes sistemas operativos y arquitecturas. Se puede utilizar con multitud de formatos de imagen diferentes, su calidad ha ido mejorando día a día en los últimos años.

2. **JavaOCR** [3] [4]. Última versión publicada 1.101. Fecha última liberación 06/06/2010. Fecha última actualización del repositorio 06/06/2012. **Licencia Permisiva** , BSD License. Escrito en Java.

   Este es un motor genérico OCR entrenable. Por defecto no es capaz de realizar la detección y extracción de caracteres. Sin embargo es capaz de filtrar y limpiar la imagen, convertir a escala de grises, dividir el documento en líneas, dividir las líneas en caracteres, y finalmente comparar cada carácter con los patrones conocidos de las imágenes de capacitación proporcionados por el usuario, y obtener como salida las opciones más cercanas como texto.

   El motor fue escrito originalmente para digitalizar documentos o secciones específicas de los documentos, que fueron impresos a partir de un conjunto de fuentes conocidas.

3. **Ocrad** [5] [6]. Última versión publicada 0.21. Fecha de la última publicación 10/01/2011. Fecha última actualización del repositorio 22/01/2012. **Licencia Robusta**, GNU Public License v3. Escrito en C y C++.

GNU Ocrad es un programa de OCR basado en un método de extracción de características (*feature extraction*). Ocrad es capaz de leer imágenes en diferentes formatos como pbm (*mapa de bits*), pgm (*escala de grises*) o ppm (*color*), y produce texto en formato UTF-8.

También incluye un analizador de composición (*layout*) capaz de separar las columnas o bloques de texto que forman normalmente las páginas impresas.

Ocrad puede ser usado como aplicación autónoma desde la línea de comandos o como complemento (*backend*) de otros programas.

4. **Tesseract** [7] [8]. Última versión publicada 3.02. Fecha última publicación 01/11/2012. Fecha última actualización repositorio 03/11/2012. **Licencia Permisiva**, Apache License 2.0. Escrito en C y C++.

   El motor de OCR Tesseract fue uno de los 3 mejores motores en la prueba de Precisión 1995 UNLV. Entre 1995 y 2006 tuvo poca evolución, pero desde entonces se ha mejorado notablemente con la colaboración de Google y es probablemente uno de los motores más precisos de código abierto. En combinación con la librería de procesamiento de imágenes Leptonica es capaz de leer una amplia variedad de formatos de imagen y convertirlos a texto en más de 40 idiomas.

5. **Cuneiform** [9] [10]. Última versión publicada 1.1. Fecha última publicación 19/04/2011. Fecha última actualización repositorio 19/04/2011. **Licencia Permisiva**, Simplified BSD Licence. Escrito en C y C++.

   Cuneiforme es un sistema OCR originalmente desarrollado como código abierto y basado en tecnologías cognitivas. Este proyecto tiene como objetivo crear una versión totalmente portátil de la escritura cuneiforme[4].

4 Más información acerca de escritura cuneiforme: http://es.wikipedia.org/wiki/Escritura_cuneiforme

2.2 ANÁLISIS DE PROYECTOS DE SL RELACIONADOS CON MOTORES OCR

A continuación se describen dos proyectos SL que están estrechamente relacionados con los motores de OCR comentados en el apartado anterior. Las aplicaciones que se comentan van más allá del simple reconocimiento óptico de caracteres, por lo que integran diferentes módulos para ofrecer funcionalidades añadidas. Estas aplicaciones integran alguno de los motores comentados anteriormente, sobre los que delegan la labor propia del reconocimiento óptico de caracteres.

Proyecto	WebSite	Ultima Versión	Fecha publicación	Ultima actualización repositorio	Escrito en	Licencia
OCRopus	http://code.google.com/p/ocropus/	0.5	01/06/12	01/06/12	C/C++ /Python	Apache License 2.0
OCRFeeeder	https://live.gnome.org/OCRFeeder	0.7.9	11/04/12	22/04/12	C y C++	GPL v2

Tabla 4: Proyectos de SL relacionados con OCR.

A continuación se realiza una pequeña descripción de cada uno de los proyectos de SL enumerados en la tabla anterior:

- **OCRopus** [11] [12]. Última versión publicada 0.5. Fecha de la última publicación 01/06/2012. Fecha última actualización del repositorio 01/06/2012. **Licencia Permisiva**, Apache License 2.0.

  OCRopus es un sistema OCR que combina un módulo de análisis de la distribución[5], un módulo de reconocimiento de caracteres[6] (*utiliza la librería tesseract*), y un módulo para el modelado del lenguaje[7]. Su objetivo principal se centra en la conversión de documentos de gran volumen y en la mejora de escritorios que ayuden a personas con discapacidad visual.

- **OCRFeeeder** [13] [14]. Última versión publicada 0.7.9. Fecha de la última

5 Más información acerca de análisis de la distribución: http://en.wikipedia.org/wiki/Document_Layout_Analysis
6 Más información acerca de OCR: http://en.wikipedia.org/wiki/Optical_character_recognition
7 Más información acerca del modelado del lenguaje: http://en.wikipedia.org/wiki/Language_modeling

publicación 11/04/2012. Fecha última actualización del repositorio 22/04/2012. **Licencia Robusta**, GNU Public License v2.

OCRFeeder es un software libre de escritorio de OCR para la suite GNOME. Convierte documentos en papel a archivos de documentos digitales o los hace accesibles a los usuarios con discapacidad visual. El programa está escrito en *Python* y utiliza la *GTK+* biblioteca (*usando PyGTK*) que actúa como *front-end* para otras herramientas existentes. Por ejemplo, no realiza el reconocimiento de caracteres en sí, sino que utiliza programas externos, por lo que hace uso del motor OCR que este instalado en el sistema. Se puede adaptar para utilizar cualquiera de los motores OCR comentados, es decir, Cuneiform , GOCR , Ocrad y Tesseract como módulo para el OCR.

Este último proyecto que se comenta en este apartado no trata sobre un software para el reconocimiento óptico de caracteres, sino de una librería general para el tratamiento digital de imágenes, que nos puede permitir la creación de un algoritmo propio de OCR o también nos puede ayudar a realizar un preprocesado de las imágenes previo al tratamiento para OCR.

Proyecto	WebSite	Ultima Versión	Fecha publicación	Ultima actualización repositorio	Escrito en	Licencia
OpenCV	http://opencv.org/	2.4.2	04/07/12	04/07/12	C/C++	BSD License

Tabla 5: Otro Proyecto de SL.

- **OpenCV** [15] [16]. Última versión publicada 2.4.2. Fecha de la última publicación 04/07/2012. Fecha de la última modificación 04/07/2012. **Licencia permisiva**, BSD License. Escrito en C y C++.

  OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999 se ha utilizado en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia

BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, MacOS X, Windows y Android. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel (IPP).

En el siguiente enlace *http://blog.damiles.com/2008/11/basic-ocr-in-opencv/* se muestra como se puede crear una aplicación rudimentaria que ofrece las opciones más básicas para realizar el reconocimiento óptico de caracteres sobre imágenes digitalizadas, a partir de esta librería. Esta aplicación ejecuta un conjunto de algoritmos de procesamiento sobre las imágenes, de forma similar a lo expuesto en el *ANEXO A – INFORMACIÓN ALGORITMOS OCR*.

2.3 PRIMERAS PRUEBAS Y TESTS

A lo largo de las siguientes etapas del proyecto se abordará el desarrollo y la creación de una librería para la plataforma Android adaptada y optimizada para teléfonos móviles inteligentes. Esta librería debe realizar las tareas de reconocimiento óptico de caracteres utilizando la cámara del teléfono. Para poder llevar a cabo esta tarea, seleccionaremos alguna de las librerías que se han comentado en el apartado *2.1 ANÁLISIS MOTORES OCR EN SL* para utilizarla como base para el desarrollo de nuestra solución.

Para ello hemos instalado, configurado y probado las diferentes librerías y soluciones comentadas. Y hemos realizado un conjunto de validaciones mediante imágenes de prueba como la que se

muestra a continuación.

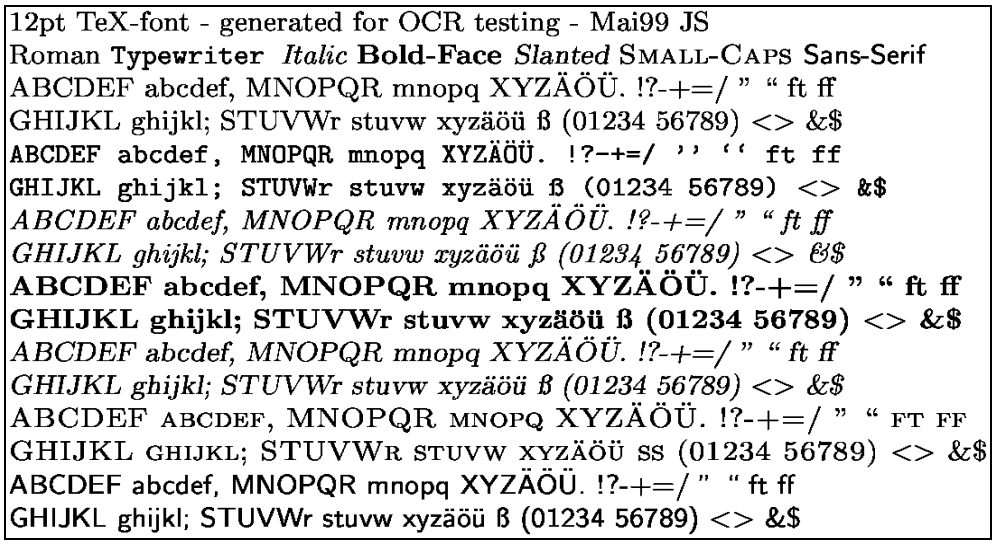

Figura 2: Ejemplo de imagen de Test

Estas imágenes tienen un conjunto variado de caracteres y fuentes que nos servirán para valorar de forma general las capacidades de las librerías comentadas. Por lo que intentaremos obtener una comparativa de las siguientes características:

1. **Eficacia**: Lógicamente debemos seleccionar aquella librería que consiga realizar correctamente o con menor margen de error el proceso de OCR para el conjunto de imágenes que tenemos como objetivo (*tickets TPV*). Para ello se realizarán diferentes pruebas para validar la eficacia de las librerías sobre un conjunto de imágenes de Test. Estas pruebas las realizaremos sobre un sistema GNU/Linux (*Debian/Ubuntu*) donde instalaremos estas librerías.

2. **Adaptabilidad**: También es interesante tener una librería que sea fácilmente adaptable para otras finalidades u otros objetivos concretos, por ejemplo sería interesante poder adaptarse con facilidad a otros idiomas u otras fuentes de caracteres. Por lo que también debemos considerar en nuestra elección que la librería sea adaptable ante diferentes entornos.

3. **Eficiencia**: También debemos tener en cuenta la eficiencia de las librerías, es decir, el coste en recursos (CPU, RAM, etc) que necesita para realizar la tarea de OCR. Este aspecto es importante ya que los terminales móviles disponen de unas limitaciones obvias en cuanto a recursos hardware y autonomía. Para ello realizaremos un estudio del consumo de recursos

sobre el conjunto de imágenes de Test, descrito anteriormente.

Otros aspectos que no debemos descuidar y que deben ser tenidos en cuenta antes de elegir la librería o el proyecto sobre el que apoyaremos nuestra solución son:

4. **Licencia:** Este aspecto puede definir la finalidad del software que se desea construir, es decir, si podrá ser utilizada dentro de aplicaciones comerciales o únicamente podremos realizar aplicaciones de SL con ella. Si nuestra librería se basa en software que está licenciado bajo una *licencia robusta* (por ejemplo GPL) cualquier aplicación que haga uso de ella estará obligada a ser distribuida bajo esta misma licencia. Si por el contrario deseamos crear un librería que pueda ser utilizada por aplicaciones comerciales debemos basarnos en alguna de las librerías que estén licenciadas bajo alguna *licencia permisiva*, por ejemplo la licencia BSD o Apache Licence 2.0.

5. **Evolución**: Otro aspecto importante a tener en cuenta en el proceso de selección de la librería es la evolución y el mantenimiento que hacen las comunidades de SL que existen en torno a estos proyectos. Por lo que es importante elegir aquellas librerías que tienen una comunidad activa alrededor de ellas, ya que esto propicia un buen mantenimiento del software y continuas evoluciones y correcciones que podremos ir incorporando.

Más adelante en el apartado *3.4 ELECCIÓN LIBRERÍA* se comentan los resultados obtenidos y las conclusiones extraídas durante el proceso de selección realizado entre las diferentes librerías, teniendo en cuenta los diferentes aspectos enumerados en este punto.

CAPÍTULO 3

DISEÑO

3. DISEÑO

En esta fase del TFM se realizan las labores necesarias para definir una solución viable que cumpla con los objetivos planteados al inicio de este proyecto. Para ello debemos tener en cuenta el análisis realizado anteriormente e incorporar las características y requisitos particulares que deberá afrontar nuestra solución software.

A lo largo de los siguientes apartados se comentan algunos de los requisitos que deberá afrontar y algunas de las características con las que trabajará nuestra solución software, entre ellas destacaremos las características hardware para las cuales se diseña la solución, la capacidad de adaptación de nuestro software a diferentes tipografías, tipos de caracteres, etc. También comentaremos las condiciones en que se debe realizar el proceso de adquisición de las imágenes.

Para finalizar con esta fase del proyecto se expondrán los resultados obtenidos en las pruebas realizadas sobre las diferentes librerías y se elegirá la que ofrezca las mejores prestaciones, y a partir del software elegido construiremos nuestra librería para la plataforma Android.

3.1 CAPACITACIÓN Y ADAPTACIÓN DE LAS LIBRERÍAS

Uno de los puntos claves en el desarrollo de este proyecto lógicamente es la construcción de una solución software haciendo uso de alguna de las librerías de SL, comentadas en el punto *2.1 ANÁLISIS MOTORES OCR EN SL*, que nos permita realizar las labores propias del reconocimiento óptico de caracteres y que además sea capaz de adaptarse a las características particulares que poseen las imágenes digitalizadas de *tickets*.

Por lo tanto la solución software que diseñemos debe ser capaz no sólo de realizar las labores y tareas generales del procesado OCR sobre textos generales, sino que también debe tener la capacidad o la posibilidad de poder adaptarse y realizar la extracción de caracteres sobre imágenes más particulares, como lo son los textos y la tipografía en *tickets*.

En consecuencia la capacidad de adaptación o capacitación a nuevos patrones de referencia por parte de las librerías de SL comentadas en *2.1 ANÁLISIS MOTORES OCR EN SL*, pasa a ser un parámetro crítico que a su vez influye directamente en otros aspectos como su eficacia y eficiencia. Esto es debido a que el proceso de detección y reconocimiento de caracteres sobre imágenes que tengan características diferentes de un texto general, puede producir resultados de muy baja calidad o con múltiples errores de detección y reconocimiento.

En la siguiente fase, es decir en la etapa de implementación, intentaremos documentar todos los pasos necesarios para realizar estas labores de capacitación y adaptación. De este modo, con independencia de la finalidad de este TFM, este software tendrá la posibilidad de ser adaptado y capacitado para cualquier otra finalidad, utilizando para ello los patrones de referencia específicos que se hayan creado.

De entre la gran variedad de *ticktes* disponibles en el mercado nosotros nos vamos centrar en un conjunto de *tickets* específico. Por lo tanto adaptaremos y capacitaremos nuestro software para realizar el proceso de OCR sobre imágenes de *tickets*. A continuación se muestran algunos ejemplos de imágenes de *tickets* para los cuales se han realizado las tareas de adaptación de nuestra solución software.

Figura 3: Imágenes de tickets

Figura 4: Imágenes de tickets

Este conjunto de imágenes es una muestra de diferentes tipos de *tickets* generados por diferentes terminales TPV proporcionados por diversos bancos.

Lo primero que se puede observar es que cada uno de estos *tickets* está generado con diferentes tipografías y que en algunos casos difieren bastante unas de otras. Por lo tanto es muy probable que para cada tipo de *ticket* se necesite realizar una labor especifica de adaptación y capacitación acorde a su tipografía. Aunque si observamos con detalle también podemos ver que todos estos *ticktes* comparten ciertas características comunes que a continuación pasamos a detallar:

1. En todos ellos aparecen un conjunto casi idéntico de palabras:
    - *“COMERCIO:”*
    - *“TPV:”*
    - *“APLIC.:”*
    - *“MASTERCAD” o “VISA”*
    - *“Tran:”*
    - *“Sec:”*
    - *“Aut:”*
    - *“Op:”*
    - *“Resp:”*
    - *“Fecha:”*
    - *“Hora:”*

2. También existe un conjunto limitado de caracteres sobre el que debemos realizar el proceso de OCR:
    - Entre el conjunto de posibles palabras que pueden aparecer en mayúscula se encuentra el nombre del establecimiento y el nombre del propietario de la tarjeta. Por lo tanto el conjunto de caracteres en mayúsculas que debemos ser capaces de detectar es el abecedario español completo *“A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P, Q, R, S, T, U, V, W, X, Y y Z”*.
    - Entre el conjunto de palabras que aparecen con letras minúsculas sólo se encuentran las comentadas en los apartados anteriores, por lo tanto el conjunto de caracteres en minúsculas que deberemos ser capaces de detectar es *“a, c, e, h, n, o, p, r, s, t, u”*.
    - Entre el conjunto de posibles caracteres numéricos que se pueden encontrar en un *ticket* aparecen todos los caracteres numéricos, por lo que deberemos de ser capaces de detectar *“*0,1,2,3,4,5,6,7,8,9”.

- Finalmente identificamos el conjunto de caracteres especiales que se pueden encontrar en un ticket *"* : / . ,"*.

3.2 TELÉFONO MÓVIL

En este apartado se identifica el terminal móvil [10] sobre el que nos apoyaremos para realizar las diferentes etapas de este TFM como son el análisis, diseño, desarrollo y pruebas. El dispositivo móvil que utilizaremos para realizar el proyecto es un *Samsumg Galaxy SII.*

A continuación se identifican las características técnicas más relevantes que posee este terminal, resaltando en negrita las más importantes como son: Versión de S.O, Procesador, memoria RAM, espacio en disco, cámaras, batería.

GENERAL	**Red**	GSM 850 / 900 / 1800 / 1900 - HSDPA 850 / 900 / 1900 / 2100
TAMAÑO	**Dimensiones**	125.3 x 66.1 x 8.5 mm
	Peso	116 g
DISPLAY	**Tipo**	Super AMOLED Plus touchscreen capacitivo, 16M colores
	Tamaño	480 x 800 píxeles, 4.3 pulgadas
		- Pantalla Gorilla Glass - Interfaz de usuario TouchWiz v4.0 - Soporte multi-touch - Sensor acelerómetro para auto rotación - Controles sensibles al tacto - Sensor de proximidad para auto apagado - Sensor giroscópico - Teclado Swype
RINGTONES	**Tipo**	Polifónico, MP3, WAV
	Vibración	Si
		- Conector de audio 3.5 mm
MEMORIA	**Agenda telefónica**	Entradas y campos prácticamente ilimitados, Foto de llamada
	Registro de llamadas	Prácticamente ilimitado
	Slot de tarjeta	**microSD hasta 32GB, 8GB incluidos**
		- 16GB/32GB memoria interna, 1GB RAM - Procesador Exynos Cortex A9 dual-core 1.2 GHz, GPU Mali-400MP
CARACTERÍSTICAS	**GPRS**	Class 12 (4+1/3+2/2+3/1+4 slots)
	Velocidad de datos	32 - 48 kbps
	OS	**Android OS, v4.0.3 - Ice Cream Sandwich** [8]

	Mensajería	SMS, MMS, Email, Push Mail, IM, RSS
	Navegador	HTML
	Reloj	Si
	Alarma	Si
	Puerto infrarrojo	No
	Juegos	Si
	Colores	Negro, Blanco, Rosa
		8 MP, 3264x2448 píxeles, autofocus, flash LED, geo-tagging, detección de rostro y sonrisa, foco táctil, estabilizador de imagen, video 1080p@30fps, cámara frontal 2MP
		- GPS con soporte A-GPS - Brújula digital - EDGE Clase 12 - 3G HSDPA 21Mbps / HSUPA 5.76Mbps - Wi-Fi 802.11 a/b/g/n; DLNA; Wi-Fi Direct - Bluetooth v3.0+HS - microUSB 2.0 - Integración con redes sociales - Soporte NFC (opcional) - Salida TV - Cancelación activa de ruido con micrófono dedicado - Reproductor de video MP4/DivX/XviD/WMV/H.264/H.263 - Reproductor de audio MP3/WAV/eAAC+/AC3/FLAC - Radio FM Stereo con RDS - Organizador - Editor de imagen/video - Editor de documentos (Word, Excel, PowerPoint, PDF) - Integración Google Search, Maps, Gmail, YouTube, Calendar, Google Talk, Picasa - Soporte Adobe Flash 10.1 - Memo/comandos/discado de voz - Manoslibres incorporado - Ingreso predictivo de texto
BATERÍA		**Standard, Li-Ion 1650 mAh**
	Stand-by	**Hasta 710 h (2G) / Hasta 610 h (3G)**
	Tiempo de conversación	**Hasta 18 h 20 min (2G) / Hasta 8 h 40 min (3G)**

Tabla 6: Características Técnicas Samsung Galaxy SII.

3.3 ADQUISICIÓN DE IMÁGENES OBJETIVO

En este apartado se comenta uno de los aspectos clave y previo al procesamiento de cualquier imagen digitalizada. Este aspecto es el método o proceso de adquisición de estas imágenes. Por lo tanto en este apartado vamos a identificar las características de la cámara de nuestro terminal móvil y posteriormente identificaremos cuales de estas características deberían ser incorporadas o usadas por nuestra aplicación para que las imágenes capturadas sean obtenidas en las mejores condiciones

8 Esta versión del sistema operativo Android permite diferentes opciones para la depuración de aplicaciones, por lo que es más apropiado que versiones anteriores como 2.3 Gingerbread.

posibles, y facilitar de este modo el proceso posterior a la adquisición, que es el procesamiento digital de la imagen.

A continuación se describen las características de la cámara principal del dispositivo móvil. Con esta cámara van a ser tomadas las imágenes sobre las que posteriormente se realizará el reconocimiento óptico de caracteres. Hay que tener en cuenta que este terminal móvil posee también una cámara frontal de 2MP, aunque nosotros nos vamos a centrar en su cámara principal, ya que es la que se utilizará para adquirir las imágenes. Las características más importantes de la cámara principal son:

- 8 MP
- 3264x2448 píxeles
- autofocus
- flash LED
- geo-tagging
- detección de rostro y sonrisa
- foco táctil
- estabilizador de imagen
- vídeo 1080p@30fps

Entre las características comentadas vamos a centrarnos en aquellas que consideramos imprescindibles para poder realizar la adquisición de las imágenes con un una calidad óptima, y por tanto que deberían ser gestionadas desde nuestra aplicación móvil:

- 8 MP: La aplicación deberá ser capaz de adquirir la imagen con la mejor calidad y resolución posible.

- 3264x2448 píxeles: La aplicación deberá ser capaz de adquirir la imagen con la mejor calidad y resolución posible.

- Autofocus: La aplicación debería ser capaz de tener este parámetro configurable para permitir su activación o desactivación, ya que dependiendo del ángulo de enfoque o de la

cantidad de luz este método de auto-enfoque puede funcionar incorrectamente, como se puede ver a la izquierda en la *Figura 5*.

Debido al tamaño de los tickets y a que el modo de funcionamiento de autoenfoque puede introducir errores en la adquisición, nosotros vamos a optar por mantener esta opción desactivada desde el inicio.

Figura 5: Imágenes de tickets TPV. A la izquierda capturada en modo Autoenfoque. A la derecha capturada en modo Macro.

- Flash LED: Nuestra aplicación también debería ser capaz de permitir la activación o desactivación del LED de la cámara, ya que existen múltiples situaciones en las que nos podríamos encontrar en circunstancias de una iluminación insuficiente para la adquisición de las imágenes.

3.4 ELECCIÓN LIBRERÍA

En este apartado se comenta el proceso seguido en la elección de la librería de SL, que finalmente será utilizada como base para la construcción de nuestra librería adaptada al sistema operativo Android.

En el aparatado *2.3 PRIMERAS PRUEBAS Y TESTS* se comentaban algunas de las características

que deberían ser evaluadas de estas librerías antes de tomar una decisión para determinar cuál es la que mejor se adapta a nuestras necesidades. Es importante comentar que todas estas pruebas se realizan sobre un sistema Ubuntu 12.04 [9] [10].

1. **Eficacia**: Para realizar una comparativa de la eficacia de las diferentes librerías se ha utilizado un conjunto de 4 imágenes de Test que contienen textos similares al mostrador en la *Figura 2*. A continuación se muestra una comparativa donde se recoge el porcentaje de aciertos de las diferentes librerías en el proceso de detección y extracción de caracteres:

	Imagen Test 1	**Imagen Test 2**	**Imagen Test 3**	**Imagen Test 4**
Ocrad	97,51%	98,81%	97,23%	98,90%
GOCR	95,78%	96,23%	95,97%	96,78%
Tesseract	98,34%	99,21%	98,86%	99,04%
JavaOCR	86,38%	85,48%	87,48%	85,96%

Tabla 7: Comparativa de Eficacia.

2. **Adaptabilidad**: Para realiza una comparativa de la capacidad de adaptación de las diferentes librerías a nuevos patrones de referencia, se han identificando las posibilidades y opciones que ofrecen cada una de ellas para esta característica:

	Adaptación nuevos caracteres	**Adaptación nuevas tipografías**	**Adaptación palabras frecuentes**	**Lista de caracteres permitidos**	**Lista de caracteres prohibidos**
Ocrad	NO	NO	NO	NO	NO
GOCR	NO	SI	NO	NO	NO
Tesseract	SI	SI	SI	SI	SI
JavaOCR	SI	SI	NO	NO	NO

Tabla 8: Comparativa de Adaptación.

3. **Eficiencia**: Para realizar una comparativa de eficiencia de las diferentes librerías, se han utilizado el conjunto de imágenes de Test que contienen textos similares al mostrador en la

9 **Núcleo:** 3.2.0-32-generic-pae **Arquitectura:**i686 i386 GNU/Linux
10 **CPU:** Intel(R) Core(TM)2 Duo CPU P8700 @ 2.53GHz **RAM:** 4 GB

Figura 2. A continuación se presenta una comparativa donde se muestra el tiempo (seg.) de procesado[11] que han consumido las diferentes librerías durante su ejecución:

	Imagen Test 1	Imagen Test 2	Imagen Test 3	Imagen Test 4
Ocrad	0,75	0,82	0,79	0,75
GOCR	0,89	0,86	0,92	0,87
Tesseract	2,37	1,98	2,23	2,11
JavaOCR	2,12	1,82	2,25	2,17

Tabla 9: Comparativa de Eficiencia.

4. **Licencia:** Este punto es de suma importancia ya que define la finalidad de la librería que se desea construir, es decir, si podrá ser utilizada por aplicaciones comerciales o únicamente podremos realizar aplicaciones de SL con ella.

	Licencia	Tipo
Ocrad	GPL v3	Robusta
GOCR	GPL v2	Robusta
Tesseract	Apache License 2.0	Permisiva
JavaOCR	BSD License	Permisiva

Tabla 10: Comparativa de Tipo Licencia.

5. **Evolución**: Otro aspecto importante a tener en cuenta en el proceso de selección de la librería, es la evolución y el mantenimiento que hacen las comunidades de SL que existe alrededor de estos proyectos.

	Fecha Ultima Release	Ultima Actualización Repositorio
Ocrad	10/01/11	22/01/12
GOCR	24/09/10	31/05/12
Tesseract	01/11/12	03/11/12
JavaOCR	06/06/10	06/06/10

Tabla 11: Comparativa de Evolución.

11 Para el calculo de este tiempo se ha utlizado el comando **time**.

Después de valorar todos estos aspectos observamos que la librería que mejor se adapta a nuestras necesidad es la **librería Tesseract**. A continuación se expresan los motivos que nos han llevado a esta decisión:

- Mayor eficacia: Las primeras pruebas sobre textos generales ofrecen resultados con una gran capacidad de acierto y con un pequeño margen de error.

- Mayor adaptación: Esta librería cumple con una de las necesidades fundamentales comentadas en el punto *3.1 CAPACITACIÓN Y ADAPTACIÓN DE LAS LIBRERÍAS*, ya que es la que ofrece las mayores opciones de adaptación y aprendizaje. Esta capacidad de utilizar nuevos patrones adaptados a otros tipos de textos, que no sean textos generales, repercute directamente en una mejora de su eficacia y su eficiencia.

- Una licencia permisiva que ofrece más margen de maniobra para adaptaciones o futuros usos comerciales.

- Un proyecto de SL que está en continua evolución y adaptación, incorporando nuevas funcionalidades y solucionando posibles deficiencias y errores, por lo que nos permitirá incorporar estas evoluciones y correcciones a nuestra solución software.

- La menor eficiencia detectada en esta librería puede reducirse creando patrones de referencia que estén adaptados al texto que se desea procesar, de este modo se puede minimizar el tiempo de procesado.

CAPÍTULO 4

IMPLEMENTACIÓN

4. IMPLEMENTACIÓN

En esta etapa del proyecto se han abordado las tareas para desarrollar una solución software dando respuesta a las necesidades explicadas y definidas durante las etapas de análisis y diseño. Como ya se comentó al final de la fase anterior, la librería de software libre que mejor parecía adaptarse a nuestros requerimientos es la **librería Tesseract**. Por lo tanto a partir de esta librería intentaremos construir una solución software que dé respuesta a los requerimientos definidos. A lo largo de los siguientes apartados explicaremos los puntos más relevantes de este proceso de adaptación e implementación.

4.1 IMPLEMENTACIÓN DE LA LIBRERÍA

4.1.1 Introducción

Para poder utilizar la librería Tesseract en el sistema operativo Android es necesario la adaptación de esta librería desarrollada en C/C++ a un entorno comprensible y utilizable desde Android. Para alcanzar este objetivo existen dos posibilidades:

1. La transformación de todo el código de la librería Tesseract de C/C++ a Java, con la consiguiente complejidad y coste que ello supone, y además con la probable pérdida de eficiencia que podría producirse por la utilización de un lenguaje como Java en lugar de C o C++.

2. La utilización del código nativo C/C++ de la librería Tesseract directamente desde una aplicación Android escrita en Java[12], sin necesidad de convertir toda la librería al lenguaje

12 Las aplicaciones para Android son desarrolladas mediante el SDK de Android basado en Java, el cual incluye un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales.

Java.

Se ha optado por utilizar la solución comentada en segundo lugar, ya que nos permite el acceso desde Java al código nativo de la librería Tesseract escrita en C/C++, sin necesidad de reescribir sus funcionalidades en Java. Por lo que nuestra propuesta de solución va a crear un interfaz JNI[13](Java Native Interface) que permitirá a cualquier aplicación Android escrita en Java y ejecutada sobre una máquina virtual Dalvik[14] poder interactuar con la librería Tesseract escrita en lenguaje C/C++. Para este propósito Android ofrece un conjunto de herramientas llamadas NDK[15], que nos permiten utilizar bibliotecas escritas en C, C++ u otros lenguajes, siempre que hayan sido compiladas para su arquitectura hardware concreta (ARM o x86). Por lo tanto, los programas en Java que corren en la máquina virtual Dalvik pueden hacer uso de clases nativas por medio de la función *System.loadLibrary,* que forma parte de las clases estándar de Java en el SDK de Android.

Para la creación de esta interfaz JNI para la librería Tesseract se han tenido en cuenta diferentes proyectos de software libre que actualmente ya implementan parte de esta interfaz, por lo que a continuación se enumeran algunos de los proyectos revisados:

Proyecto	WebSite	Versión	Fecha	Licencia
Tess4J	http://tess4j.sourceforge.net/	1.0	31/08/2012	Apache License, v2.0
Tesseract-android-tools	http://code.google.com/p/tesseract-android-tools/	1.0	22/09/2012	Apache License, v2.0
Tess-two	https://github.com/rmtheis/tess-two	1.0	09/07/2012	Apache License, v2.0

Tabla 12: Interfaces JNI para Tesseract

De las diferentes interfaces mostradas en la tabla anterior se ha optado por basar nuestra solución en las dos últimas *Tesseract-android-tools* y *Tess-two*, ya que son interfaces JNI creadas especialmente para Android mediante el SDK de Android, por lo que están especialmente diseñadas para ser usadas por aplicaciones Android, al contrario que la solución expuesta en primer lugar, *Tess4J*, que trata de una interfaz JNI más general para sistemas Windows o GNU/Linux.

13 Para más información sobre JNI: http://es.wikipedia.org/wiki/Java_Native_Interface
14 Para más información sobre máquina virtual Dalvik: http://es.wikipedia.org/wiki/Dalvik
15 Para más información sobre NDK de Android: http://developer.android.com/tools/sdk/ndk/index.html

Además de lo comentado en el punto anterior, las interfaces para Android *Tesseract-android-tools* y *Tess-two* cuentan con una extensa interfaz JNI que nos da acceso a un amplio abanico de funcionalidades nativas de la librería Tesseract.

4.1.2 Clases principales del interfaz JNI

En este apartado comentaremos la interfaz JNI que hemos adoptado para nuestra solución software, basándonos en las interfaces *Tesseract-android-tools* y *Tess-two* comentadas en el punto anterior.

Gran parte de la información que se muestra en este apartado también ha sido documentada en el propio código fuente mediante comentarios que han sido procesados por la herramienta Doxygen[16].

Para un funcionamiento óptimo de la librería Tesseract es necesario que ésta se apoye en algunas funcionalidades ofrecidas por la librería Leptonica, que se encarga del procesamiento digital de las imágenes. Por lo tanto es necesario incorporar esta librería y crear una interfaz JNI para aquellos métodos que deban ser usados desde la interfaz JNI de Tesseract.

En la siguiente tabla se muestran las clases que conforman la interfaz JNI para la librería Leptonica:

Clases	Descripción
AdaptiveMap	Esta clase contiene métodos de mapeo para la adaptación de imágenes
Binarize	Esta clase contiene métodos para la binarización de imágenes
Box	Esta clase es una envoltura de la clase BOX nativa de Leptonica
Constants	Esta clase contiene constantes de la librería Leptonica
Convert	Esta clase contiene métodos de conversión de la profundidad de bits de la imagen
Enhance	Esta clase contiene métodos para la nitidez de la imagen.
JpegIO	Esta clase contiene métodos de entrada y salida para imágenes

16 Para más información de la herramienta Doxygen: http://www.stack.nl/~dimitri/doxygen/index.html

	JPEG.
Pix	Esta clase es una representación en Java de un objeto nativo PIX en Leptonica
Pixa	Esta clase es una representación en Java de un objeto nativo Pixa. Este objeto contiene varios objetos PIX y sus correspondientes objetos delimitadores
ReadFile	Esta clase contiene métodos para el tratamiento de imágenes de entrada
Rotate	Esta clase contiene métodos para la rotación de las imágenes
Scale	Esta clase contiene métodos para el escalado de las imágenes
Skew	Esta clase contiene métodos para la detección de la rotación e inclinación de la imagen
WriteFile	Esta clase contiene métodos para el tratamiento de la imagen de salida

Tabla 13: Clases interfaz JNI para Leptonica

En la siguiente tabla se muestran las clases que conforman la interfaz JNI para la librería Tesseract:

Clases	Descripción
TessBaseAPI	Esta clase es la interfaz Java del motor de OCR Tesseract. No implementa todos los métodos JNI disponibles pero pone en práctica los suficientes para ser útil
TessBaseAPI.PageSegMode	Esta clase contiene los modos de ejecución del motor Tesseract

Tabla 14: Clases interfaz JNI para Tesseract

Para poder entender un poco mejor las funcionalidades que ofrecen las clases comentadas en la *Tabla 13* y *Tabla 14,* a continuación se muestran extractos de código fuente de nuestra aplicación Android, desde donde se hace uso de los métodos más importantes que implementan las clases de las interfaces JNI mostradas en las tablas anteriores.

```
import com.googlecode.leptonica.android.ReadFile;
import com.googlecode.tesseract.android.TessBaseAPI;

private TessBaseAPI baseApi; // Java interface for the Tesseract OCR
private int pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_AUTO;

if (baseApi != null) {
        baseApi.setPageSegMode(pageSegmentationMode);
        baseApi.setVariable(TessBaseAPI.VAR_CHAR_BLACKLIST, characterBlacklist);
        baseApi.setVariable(TessBaseAPI.VAR_CHAR_WHITELIST, characterWhitelist);
        }
```

En el extracto de código mostrado, se realiza en primer lugar un *import* de aquellas clases que nuestra aplicación va a necesitar de la interfaz JNI. Posteriormente se define la referencia a la clase *TessBaseAPI* desde la que se llama a los métodos que se muestran a continuación para establecer las opciones de ejecución de la librería Tesseract:

- **TessBaseAPI.setPageSegMode()**: Establece el modo de ejecución de la librería de entre los 11 modos disponibles. El modo que se establece de forma predeterminada al arranque de la aplicación será *TessBaseAPI.PageSegMode.PSM_AUTO.*

- **TessBaseAPI.setVariable()**: Con este método se pueden establecer diferentes opciones de configuración de la librería Tesseract. Entre ellas se encuentra la lista de caracteres permitidos (*characterWhitelist*), que son aquellos que la librería debe buscar, y la lista de caracteres prohibidos (*characterBlacklist*), que son aquellos que debe descartar.

```
baseApi.init(destinationDirBase + File.separator, languageCode);
baseApi.clear();
baseApi.end();
```

En el extracto de código anterior, se muestran las llamadas a algunos de los métodos principales de la clase *TessBaseAPI,* que son usados para:

- **TessBaseAPI.init()**: Este método inicializa la librería Tesseract estableciendo el directorio de referencia donde se encuentran los datos de la aplicación, en nuestro caso se trata del directorio */mnt/sdcard/Android/data/edu.uoc.ocr/files/mounted,* y el nombre del fichero que contiene los patrones de referencia que debe utilizar la librería Tesseract, en nuestro caso

serán los ficheros *spa.traineddata, eng.traineddata, cat.traineddata* o *man.traineddata*.

- **TessBaseAPI.clear()**: Libera los resultados del reconocimiento y los datos de imagen almacenados, sin llegar a la liberación de todos los datos de configuración.

- **TessBaseAPI.end()**: Cierra Tesseract y libera toda la memoria. Es equivalente a la auto-destrucción y la reconstrucción del objeto *TessBaseAPI*.

```
try {
        baseApi.setImage(ReadFile.readBitmap(bitmap));
        textResult = baseApi.getUTF8Text();
        timeRequired = System.currentTimeMillis() - start;
        // Check for failure to recognize text
        if (textResult == null || textResult.equals("")) {
                return false;
        }
        ocrResult = new OcrResult();
        ocrResult.setWordConfidences(baseApi.wordConfidences());
        ocrResult.setMeanConfidence(baseApi.meanConfidence());
        ocrResult.setRegionBoundingBoxes(baseApi.getRegions().getBoxRects());
        ocrResult.setWordBoundingBoxes(baseApi.getWords().getBoxRects());
} catch (RuntimeException e) {
        Log.e("OcrRecognizeAsyncTask",
        "Caught RuntimeException in request to Tesseract. Setting state to CONTINUOUS_STOPPED.");
        e.printStackTrace();
                                        ……..........
```

En el extracto de código mostrado se realiza el proceso de detección de caracteres sobre una imagen, para ello se hace uso de las siguientes funciones:

- **ReadFile.readBitmap()**: Crea un objeto *Pix* de la imagen obtenida por Android a través de la cámara del teléfono, que es el elemento principal que utiliza la librería Leptonica para el procesamiento de las imágenes. Actualmente sólo es compatible con formato *ARGB_8888* mapas de bits.

- **TessBaseAPI.setImage()**: Proporciona la imagen que procesará la librería Tesseract para la extracción y detección de caracteres.

- **TessBaseAPI.getUTF8Text()**: Procesa la imagen detectando y extrayendo el texto reconocido, y este texto es devuelto mediante un *String* codificado en formato UTF-8.

- **TessBaseAPI.wordConfidences()**: Devuelve el porcentaje de coincidencia de todas las palabras mediante una matriz de enteros. El número de porcentaje de coincidencia debe corresponder con el número de palabras delimitadas por espacios en g*etUTF8Text()*.

- **TessBaseAPI.meanConfidence()**: Devuelve el promedio de confianza que ofrece el reconocimiento de todo el texto procesado.

- **TessBaseAPI.getRegions()** *y* **TessBaseAPI.getWords()**: Devuelven un objeto *Pixa*, que es un objeto que contiene varios objetos *Pix* y sus correspondientes objetos delimitadores.

- **Pixa.getBoxRects()**: Devuelve una *ArrayList* de los delimitadores de recta de cada objeto *Box*.

```
if (pageSegmentationModeName.equals(pageSegmentationModes[0])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_AUTO_OSD;
} else if (pageSegmentationModeName.equals(pageSegmentationModes[1])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_AUTO;
} else if (pageSegmentationModeName.equals(pageSegmentationModes[2])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_SINGLE_BLOCK;
} else if (pageSegmentationModeName.equals(pageSegmentationModes[3])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_SINGLE_CHAR;
} else if (pageSegmentationModeName.equals(pageSegmentationModes[4])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_SINGLE_COLUMN;
} else if (pageSegmentationModeName.equals(pageSegmentationModes[5])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_SINGLE_LINE;
} else if (pageSegmentationModeName.equals(pageSegmentationModes[6])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_SINGLE_WORD;
} else if (pageSegmentationModeName.equals(pageSegmentationModes[7])) {
        pageSegmentationMode = TessBaseAPI.PageSegMode.PSM_SINGLE_BLOCK_VERT_TEXT;
}
```

En el extracto de código anterior se muestran los diferentes modos de ejecución que nuestra aplicación utiliza para configurar el modo de ejecución de la librería Tesseract. De los 11 modos existentes solamente 8 serán útiles para nuestra aplicación. Todos los modos que ofrece la librería se pueden encontrar en *TessBaseAPI.PageSegMode*.

4.2.3 Estructura de directorios y archivos

En este apartado se describe la estructura de directorios y se mencionan los archivos más relevantes del proyecto Android creado para la construcción y creación de la librería. En la siguiente imagen se muestra esta estructura de directorios tal como se puede ver desde el IDE de desarrollo Eclipse.

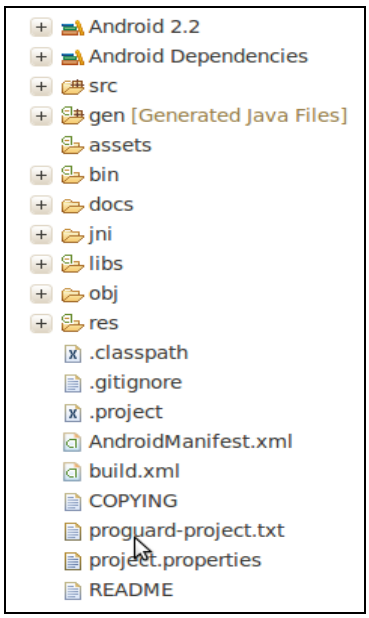

Figura 6: Estructura de directorios

- **Directorio SRC:** Contiene la totalidad del código fuente de la interfaz JNI de las librerías Leptonica y Tesseract.

- **Directorio GEN:** Esta carpeta guarda un conjunto de archivos de código Java creados automáticamente cuando se compila el proyecto. En este caso se crea el archivo *BuildConfig.java* donde se define la variable *DEBUG* a *true* o *false*, dependiendo del tipo de compilación.

- **Directorio ASSETS:** Nuestra librería no contiene ningún fichero adicional, por lo tanto este directorio estará vacío.

- **Directorio BIN:** Este directorio se genera automáticamente y es utilizado por el compilador para preparar los archivos *Class* y realizar el empaquetado final en formato *JAR*. Cabe considerar excluir esta carpeta y el directorio *gen* si se utiliza un sistema de control de versiones.

- **Directorio DOCS:** Este directorio ha sido creado por nosotros para depositar el archivo de configuración de la herramienta *Doxygen,* y la documentación generada en formato HTML a

partir del código fuente de la interfaz JNI. La página principal se puede consultar en */docs/html/index.html.*

- **Directorio JNI:** Este directorio contendrá todo el código fuente nativo perteneciente a las librerías Leptonica y Tesseract, por lo que colocaremos aquí el código fuente C/C++ de estas librerías.

- **Directorio LIBS:** En este directorio se ubicarán las librerías dinámicas Leptonica y Tesseract, una vez hayan sido compiladas para las arquitecturas *armeabi*, *armeabi-v7a*, *x86*.

- **Directorio OBJ:** En este directorio se situarán todos los objetos que se creen durante el proceso de compilación de las la librerías Leptonica y Tesseract para las arquitecturas *armeabi*, *armeabi-v7a*, *x86.* También contiene las librerías compiladas en modo estático (*lib.a*) y dinámico (*lib.so*) para cada una de las arquitecturas.

- **Directorio RES:** Es el directorio que contiene los recursos relacionados con la parte gráfica de las aplicaciones como imágenes, idiomas, estilos, sonidos, etc. En el caso de nuestra librería se encuentra vacío.

- **Archivo AndroidManifest.xml:** Podemos decir que este es el archivo principal para todos los proyectos Android, por lo que debe aparecer en el directorio raíz de todos los proyectos, ya sean aplicaciones o librerías. En este archivo se definen las características del proyecto como el nombre, paquete, versión del SDK de Android para el que se ha diseñado la librería.

- **Archivo build.xml:** Contiene la información necesaria para compilar el proyecto desde línea de comandos.

4.2.4 Compilación y generación

En este apartado vamos a describir los pasos que se deben seguir para la generación de la interfaz JNI que dará acceso a los métodos nativos de las librerías Leptonica y Tesseract. Para ello se explicará el proceso seguido a través de la línea de comandos.

En primer lugar debemos ubicar el código fuente original de las librerías Tesseract y Leptonica en los directorios correspondientes:

- Código fuente de la librería Leptonica en *./jni/com_googlecode_leptonica_android/src/*.

- Código fuente de librería Tesseract en *./jni/com_googlecode_tesseract_android/src/*.

Una vez desplegado el código fuente de las librerías en los directorios comentados, procederemos a la compilación del código C/C++ de ambas librerías para las arquitecturas *armeabi, armeabi-v7a* y *x86*. Esta tarea la llevaremos acabo mediante la herramienta *ndk-build* contenida en el paquete NDK de Android, tal como se muestra a continuación.

```
jaime@jaime-laptop:~/workspace/tesseract-android-tools$ ndk-build -j4 V=1
rm -f ./libs/armeabi/lib*.so ./libs/armeabi-v7a/lib*.so ./libs/mips/lib*.so ./libs/x86/lib*.so
Compile thumb  : lept <= adaptmap.c
rm -f ./libs/armeabi/gdbserver ./libs/armeabi-v7a/gdbserver ./libs/mips/gdbserver ./libs/x86/gdbserver
Compile thumb  : lept <= affine.c
/DATOS/Android/Codigo/android-ndk/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86/bin/arm-linux-androideabi-
gcc -MMD -MP -MF ./obj/local/armeabi/objs/lept/src/src/adaptmap.o.d -fpic -ffunction-sections -funwind-tables -fstack-
protector -D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__
-march=armv5te -mtune=xscale -msoft-float -mthumb -Os -fomit-frame-pointer -fno-strict-aliasing -finline-limit=64 -Istdio
-Ijni/com_googlecode_leptonica_android -Ijni/com_googlecode_leptonica_android/src/src
-I/DATOS/Android/Codigo/android-ndk/sources/cxx-stl/gnu-libstdc++/4.6/include -I/DATOS/Android/Codigo/android-
ndk/sources/cxx-stl/gnu-libstdc++/4.6/libs/armeabi/include -Ijni/com_googlecode_leptonica_android -DANDROID
-DHAVE_CONFIG_H -Wa,--noexecstack -O2 -DNDEBUG -g -I/DATOS/Android/Codigo/android-ndk/platforms/android-
8/arch-arm/usr/include -c  jni/com_googlecode_leptonica_android/src/src/adaptmap.c -o
./obj/local/armeabi/objs/lept/src/src/adaptmap.o
Compile thumb  : lept <= affinecompose.c

                                        …….....
```

Las opción -j del comando *ndk-build* permite lanzar en paralelo diferentes tareas para optimizar el proceso de compilación en aquellas máquinas que dispongan de más de un procesador. Esto agilizará la tarea de compilación, ya que la generación de estas librerías para las tres arquitecturas comentadas consume un tiempo considerable. La opción V=1 muestra todas las trazas de la compilación y es útil para aquellos casos en que pueda producirse un error de compilación y necesitemos depurarlo.

Una vez finalizada la compilación de las librerías se habrá generado el siguiente contenido en los

directorios *./obj* y *./libs*, tal como se puede ver en la siguiente imagen:

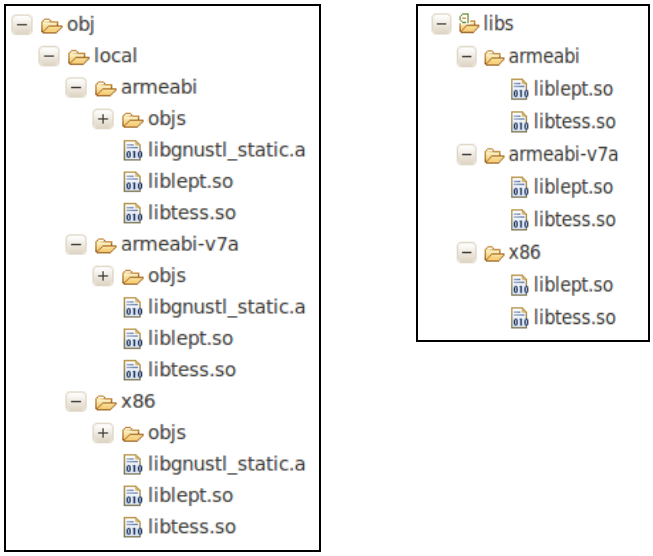

Figura 7: Contenido de directorios

Una vez compilada la parte nativa de las librerías Leptonica y Tesseract, procederemos a crear la interfaz JNI mediante las herramientas que ofrece el SDK de Android. Para ello, en primer lugar definiremos el directorio de nuestro proyecto.

```
jaime@jaime-laptop:~/workspace/tesseract-android-tools$ android update project --path .
Updated local.properties
Updated file ./proguard-project.txt
It seems that there are sub-projects. If you want to update them
please use the --subprojects parameter.
```

Después procederemos a la compilación de la parte Java mediante *ant debug* o *ant release*, dependiendo de si deseamos realizar una versión de prueba o una versión definitiva.

```
jaime@jaime-laptop:~/workspace/tesseract-android-tools$  ant release
Buildfile: /DATOS/Android/Codigo/workspace/tesseract-android-tools/build.xml

-set-mode-check:

-set-release-mode:

-release-obfuscation-check:
    [echo] proguard.config is ${proguard.config}

-check-env:
 [checkenv] Android SDK Tools Revision 21
 [checkenv] Installed at /DATOS/Android/Codigo/android-sdk-linux

-setup:
    [echo] Project Name: tesseract-android-tools
  [gettype] Project Type: Android Library

-build-setup:
    [echo] Resolving Build Target for tesseract-android-tools...
[gettarget] Project Target:   Android 2.2
[gettarget] API level:        8
    [echo] ----------
    [echo] Creating output directories if needed...
   [mkdir] Created dir: /DATOS/Android/Codigo/workspace/tesseract-android-tools/bin
   [mkdir] Created dir: /DATOS/Android/Codigo/workspace/tesseract-android-tools/bin/res
   [mkdir] Created dir: /DATOS/Android/Codigo/workspace/tesseract-android-tools/gen
   [mkdir] Created dir: /DATOS/Android/Codigo/workspace/tesseract-android-tools/bin/classes
   [mkdir] Created dir: /DATOS/Android/Codigo/workspace/tesseract-android-tools/bin/dexedLibs
    [echo] ----------
                                                    ….....
```

4.2 IMPLEMENTACIÓN DE LA APLICACIÓN

4.2.1 Introducción

Para la construcción de nuestra aplicación vamos a prestar especial atención a dos aspectos que ya fueron comentados durante las etapas de análisis y diseño:

1. La aplicación debe hacer uso de la cámara del teléfono para la adquisición de las imágenes objetivo y estas imágenes deben ser capturadas en las condiciones más óptimas posibles.

2. La aplicación debe hacer uso de la librería comentada en el apartado anterior para el procesamiento digital de las imágenes y su consiguiente detección y extracción de caracteres.

Con el fin de proveer a nuestra aplicación con las herramientas más óptimas para abordar los requisitos comentados, se ha estudiado el código fuente de diferentes aplicaciones de software libre.

También se han visitado diferentes foros donde se ofrece código fuente y donde se tratan estos aspectos. Entre ellos destacamos los siguientes:

Proyecto	WebSite	Versión	Fecha	Licencia
---	http://labs.makemachine.net/2010/03/simple-android-photo-capture/	1.0	03/10/2010	----
androidMDW	https://github.com/androidMDW/	1.0	31/03/2011	----
ZXing	http://code.google.com/p/zxing/	4.0	22/10/2012	Apache License, v2.0

Tabla 15: Aplicaciones para la gestión de la cámara

Proyecto	WebSite	Versión	Fecha	Licencia
eyes-free	https://code.google.com/p/eyes-free/	0.6.1	31/08/2012	Apache License, v2.0
android-ocr	https://github.com/rmtheis/android-ocr	0.5.13	22/09/2012	Apache License, v2.0
Simple-Android-OCR	https://github.com/GautamGupta/Simple-Android-OCR	1.0	09/0/7/2012	-----

Tabla 16: Aplicaciones para la gestión de la interfaz JNI Tesseract-android-tools

Después de hacer un pequeño estudio del funcionamiento de las aplicaciones comentadas, y viendo las funcionalidades que podrían ofrecer cada una, se ha decidido incorporar y adoptar el código fuente de las siguientes aplicaciones:

- ***Zxing:*** Para cubrir la funcionalidad encargada de la gestión de la cámara y de la adquisición de las imágenes.
- ***Android-ocr*** y ***Eyes-free:*** Para incluir aquella parte del código fuente donde se hace uso de una interfaz JNI similar a la creada en el apartado *4.1 IMPLEMENTACIÓN DE LA LIBRERÍA*.

En todo momento hemos optado por incluir código fuente de aplicaciones que estén sujetas a algún tipo de licencia de software libre, en este caso bajo la licencia *Apache License, v2.0.* De este modo se evitan futuros problemas con los derechos de autor en caso de incluir código fuente en nuestra

aplicación que no esté sujeto a ninguna licencia.

4.2.2 Clases principales de la aplicación

En este apartado comentaremos las clases de las que finalmente constará nuestra aplicación software. Una parte importante de estas clases ha sido adaptada partiendo de las aplicaciones ***Zxing, Android-ocr*** y ***Eyes-free*** comentadas en el punto anterior.

Gran parte de la información que se muestra en este apartado también ha sido documentada en el propio código fuente mediante comentarios que han sido procesados por la herramienta Doxygen.

En la siguiente tabla se muestran las clases que forman parte de nuestra aplicación software para Android:

Clases	Descripción
BeepManager	Maneja sonidos y vibraciones para **CaptureActivityForOcr**
CaptureActivityForOcr	**Actividad** que accede a la cámara e inicia el procesado mediante un subproceso de fondo. Se dibuja un visor para ayudar al usuario a definir los límites de la imagen a procesar. Finalmente se superponen los resultados cuando finaliza el procesado de forma exitosa
CaptureActivityHandler	Esta clase se encarga de gestionar los diferentes estados y situaciones que comprende el proceso de captura de imágenes
DecodeHandler	Clase que envía los datos de la imagen para realizar el procesado OCR
DecodeThread	Este hilo hace todo el trabajo pesado de decodificación de las imágenes
FinishListener	**Listener** que se utiliza para salir de la aplicación en algunos casos
HelpActivity	**Actividad** que muestra las páginas de ayuda o información al usuario a través de un **WebView**
LanguageCodeHelper	Clase para el manejo de las funciones relacionadas con la conversión entre los códigos de idioma estándar y la conversión de códigos de idioma para los nombres de idiomas
LuminanceSource	El propósito de esta clase abstracta es extraer diferentes

	valores de escala de grises de luminancia. La clase proporciona métodos inmutables, por lo que cualquier modificación creará una copia de la imagen. Esto es para asegurar que un lector no modifica la fuente de luminancia original
OcrCharacterHelper	Clase auxiliar para habilitar las listas negras de caracteres y las listas blancas definidas para cada fichero de patrones
OcrInitAsyncTask	Instala los datos lingüísticos necesarios para el motor OCR e inicializa el motor de OCR utilizando un subproceso en segundo plano
OcrRecognizeAsyncTask	Clase para enviar solicitudes al motor de OCR en un hilo separado. Envía un mensaje de éxito/fracaso y cierra el cuadro de diálogo de progreso
OcrResult	Encapsula el resultado del OCR
OcrResultFailure	Clase para mantener los metadatos de aquellos resultados fallidos del procesado OCR
OcrResultText	Encapsula el texto, caracteres, palabras y coordenadas resultantes del reconocimiento OCR
PlanarYUVLuminanceSource	Esta clase es la implementación de **LuminanceSource**. Puede utilizarse para excluir píxeles superfluos alrededor del perímetro. Funciona para cualquier formato de píxel donde el canal Y es plano y aparece por primera vez
PreferencesActivity	**Actividad** para manejar las preferencias que se guardan en cada sesión de la aplicación. Muestra una jerarquía de preferencias para el usuario, organizadas en secciones. Estas preferencias se muestran en el menú de opciones que aparece cuando el usuario pulsa el botón de menú
ViewfinderView	Esta vista se superpone en la parte superior de la vista previa de la cámara. Se añade el rectángulo visor y permite transparencia parcial fuera de él, así como el texto resultante
camera.AutoFocusManager	Implementa los métodos de la interfaz **Camera.AutoFocusCallback**
camera.CameraConfigurationManager	Esta clase se ocupa de la lectura, del análisis y los ajustes de aquellos parámetros de la cámara que se utilizan para configurar su hardware
camera.CameraManager	Esta clase contiene los servicios de la cámara y espera ser la única que interactúe con ella. La aplicación encapsula los pasos necesarios para tomar imágenes de tamaño de vista previa que se utilizan tanto para la vista previa como para la decodificación
camera.PreviewCallback	Se le llama cuando se recibe el siguiente cuadro de vista

	previa
camera.ShutterButton	Es un botón diseñado para ser utilizado como botón de disparo en la pantalla principal. En la actualidad es un **ImageView** que puede llamar a un delegado cuando se pulsa o se producen cambios de estado
camera.ShutterButton.OnShutterButtonListener	Es el interfaz donde se definen las llamadas que gestionan los cambios de estado cuando se presiona el botón **camera.ShutterButton**

Tabla 17: Clases de nuestra aplicación

Para facilitar el proceso de comprensión de las funcionalidades generales de la aplicación y de las funcionalidades de las clases expuestas en la tabla anterior, se ha creado en el *ANEXO B – MANUAL DE USUARIO*, un manual de usuario donde podemos hacernos una idea de las funcionalidades generales que ofrece la aplicación y las opciones de configuración de las que consta. A través de este manual también se ofrece la posibilidad de entender de forma más visual el cometido de alguna de las clases aquí comentadas.

4.2.3 Estructura de directorios y archivos

En este apartado se describe la estructura de directorios y se mencionan los archivos más relevantes del proyecto Android creado para nuestra aplicación. En la siguiente imagen se muestra esta estructura de directorios tal como se puede ver desde el IDE de desarrollo Eclipse.

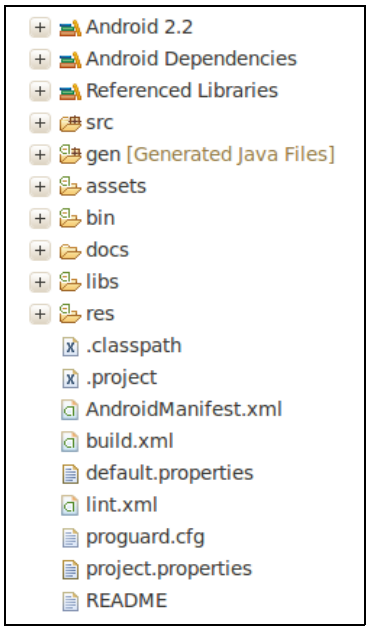

Figura 8: Contenido de directorios

- **Directorio SRC:** Contiene la totalidad del código fuente Java de nuestra aplicación.

- **Directorio GEN:** Esta carpeta guarda un conjunto de archivos de código Java creados automáticamente cuando se compila el proyecto. El archivo *R.java* ajusta automáticamente todas las referencias a archivos y valores de la aplicación guardados en la carpeta *res*. También se crea el archivo *BuildConfig.java* donde se define la variable *DEBUG* a *true* o *false*, dependiendo del tipo de compilación.

- **Directorio ASSETS:** En esta carpeta depositaremos todos los archivos que vayamos a querer que acompañen a la aplicación, en nuestro caso estarán organizados en tres directorios, uno para cada uno de los idiomas para los que se ha desarrollado la interfaz gráfica de la aplicación (español, inglés y catalán). Estos ficheros y directorios son empaquetados en el *APK* final. Pueden ser considerados como recursos al igual que el contenido de la carpeta *res*, pero no están indexados en el fichero *R.java* de modo que el acceso es menos eficiente y algo más laborioso. En este directorio ubicaremos los ficheros *html* que se mostrarán cuando se inicie la actividad *HelpActivity.* Estos archivos mostrarán información acerca de nuestra aplicación.

- **Directorio BIN:** Esta carpeta al igual que el directorio *gen,* se genera automáticamente y la utiliza el compilador para preparar los archivos para el empaquetado final en forma de *APK*.

- **Directorio DOCS:** Este directorio ha sido creado por nosotros para depositar el archivo de configuración de la herramienta Doxygen y la documentación generada en formato HTML a partir del código fuente contenido en el directorio *src*. La página principal se puede encontrar en */docs/html/index.html*.

- **Directorio LIBS:** En este directorio se incluirán aquellas librerías externas o de terceros que vayan a enlazar con nuestra aplicación. En este caso nuestra aplicación hace uso de la librería externa *jtar-1.0.4,* utilizada desde *OcrInitAsyncTask* para el desempaquetado y descompresión de los ficheros de patrones de los idiomas español, inglés y catalán, descargados desde http://tesseract-ocr.googlecode.com/files/.

- **Directorio RES:** Contiene toda la información relacionada con los recursos gráficos que se hayan definido para construir la parte gráfica de nuestra aplicación Android. Este directorio esta formado por los siguientes subdirectorios:
    - res/drawable/: Este subdirectorio contiene las imágenes que necesita la aplicación. Está divido en tres subdirectorios; *drawable-ldpi*, *drawable-mdpi* y *drawable-hdpi*, que contienen imágenes de diferentes tamaños en función de la resolución del dispositivo.
    - res/raw/: Este subdirectorio contiene archivos de propósito general. En nuestro caso contiene un archivo de sonido en formato .ogg.
    - res/layout/: Incluye los archivos que definen el diseño de la interfaz gráfica de nuestra aplicación. Para esta aplicación se han diseñado dos vistas:
        1. La interfaz principal gestionada por la actividad *CaptureActivityForOcr*, que se encarga de presentar la vista principal de la aplicación.
        2. Una segunda interfaz gráfica gestionada por la actividad *HelpActivity*, que se encarga de presentar la vista donde se recoge información adicional acerca de la aplicación.
    - res/values/: Guarda información acerca de los datos que utiliza la aplicación, tales como colores, cadenas de texto, estilos, dimensiones, etc. En este directorio se guardarán las cadenas de texto para el idioma principal de la aplicación que en este caso será el español.
    - res/values-ca/: En este directorio se guardarán las cadenas de texto para traducir el GUI al idioma Catalán.
    - res/values-en/: En este directorio se guardarán las cadenas de texto para traducir el GUI al idioma Inglés.
    - res/preferences/: Este directorio contiene el archivo *preferences.xml* donde se definen todas las opciones de configuración que ofrece la aplicación. Esta vista es gestionada por la actividad *PreferencesActivity***.**

- **Archivo AndroidManifest.xml:** Podemos decir que éste es el archivo principal de todos los proyectos Android y se encuentra en el directorio raíz del proyecto. En este archivo se definen las características del proyecto como el nombre, paquete o los permisos que va a requerir la aplicación. También es en este archivo donde se describen los componentes de la aplicación como las actividades que la componen.

- **Archivo build.xml:** Contiene la información necesaria para compilar el proyecto desde la línea de comandos.

- **Archivo project.properties:** Contiene información de dependencias con librerías y la versión del SDK necesaria para la generación y compilación de la aplicación.

4.2.4 Compilación y generación

Para la compilación del proyecto y la generación del fichero APK, explicaremos el proceso seguido a través de la línea de comandos. Para ello en primer lugar definiremos el directorio de trabajo de nuestro proyecto.

```
jaime@jaime-laptop:~/workspace/edu.uoc.ocr.CaptureActivity$ android update project --path .
Updated local.properties
It seems that there are sub-projects. If you want to update them
please use the --subprojects parameter.
```

Después procederemos a la compilación del proyecto mediante *ant debug* o *ant release*, dependiendo de si deseamos realizar una versión de prueba o una versión definitiva.

```
jaime@jaime-laptop:~/workspace/edu.uoc.ocr.CaptureActivity$ ant release
Buildfile:/DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/build.xml

-set-mode-check:

-set-release-mode:

-release-obfuscation-check:
    [echo] proguard.config is proguard.cfg
    [echo] Proguard.config is enabled

-check-env:
 [checkenv] Android SDK Tools Revision 21
 [checkenv] Installed at /DATOS/Android/Codigo/android-sdk-linux

-setup:
    [echo] Project Name: OCR Configurable
  [gettype] Project Type: Application

-build-setup:
    [echo] Switching between debug and non debug build: Deleting previous compilation output...
   [delete] Deleting directory /DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin/classes
    [echo] Resolving Build Target for OCR Configurable...
[gettarget] Project Target:   Android 2.2
[gettarget] API level:        8
```

```
    [echo] ----------
    [echo] Creating output directories if needed...
   [mkdir] Created dir: /DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin/classes
    [echo] ----------
    [echo] Resolving Dependencies for OCR Configurable...
[dependency] Library dependencies:
[dependency]
[dependency] ------------------
[dependency] Ordered libraries:
[dependency]
[dependency] ------------------
[dependency] API<=15: Adding annotations.jar to the classpath.
    [echo] ----------
    [echo] Building Libraries with 'release'...
                                                    ….....................................
                                                    ….....................................
                                                    ….....................................

-release-nosign:
    [echo] No key.store and key.alias properties found in build.properties.
    [echo] Please sign /DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin/OCR Configurable-release-
unsigned.apk manually
    [echo] and run zipalign from the Android SDK tools.
[propertyfile] Updating property file: /DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin/build.prop
[propertyfile] Updating property file: /DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin/build.prop
[propertyfile] Updating property file: /DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin/build.prop
[propertyfile] Updating property file: /DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin/build.prop

-release-sign:

-post-build:

release:

BUILD SUCCESSFUL
Total time: 17 seconds
```

Una vez finalizado este proceso de compilación y generación podremos encontrar en el directorio *bin/* el fichero *APK* generado. Llegados a este punto podremos instalar la aplicación directamente en nuestro terminal móvil utilizando la herramienta *adb* proporcionada en el SDK de Android. Para ello debemos tener conectado el terminal móvil al ordenador y ejecutar el siguiente comando:

```
jaime@jaime-laptop:~/DATOS/Android/Codigo/workspace/edu.uoc.ocr.CaptureActivity/bin$ adb install OCR\
Configurable-debug.apk
4383 KB/s (5832261 bytes in 1.299s)
        pkg: /data/local/tmp/OCR Configurable-debug.apk
Success
```

4.3 MANUAL PARA LA CREACIÓN DE UN PATRÓN DE REFERENCIA

Antes de describir el procedimiento que debemos seguir para la creación de nuestro propio archivo de patrones [11] es importante comentar que nuestra aplicación móvil utilizará de forma predeterminada tres ficheros de patrones descargados desde http://tesseract-ocr.googlecode.com/files/, entre los cuales se encuentran los ficheros de patrones para los idiomas español (*spa.traineddata)*, inglés (*eng.traineddata*) y catalán (*cat.traineddata*). Por este motivo también se ha decidido adaptar y traducir la interfaz gráfica de la aplicación para poder trabajar en estos tres idiomas, facilitando de este modo la utilización de la aplicación por parte de usuarios que tengan como idioma materno alguno de los comentados.

Los ficheros descargados desde http://tesseract-ocr.googlecode.com/files/ son patrones generales que han sido creados para la detección y extracción de aquellos caracteres que aparecen en cada uno de los idiomas, y están preparados para un conjunto limitado de estilos de fuente y tipografías. Por lo tanto no están especialmente adaptados para la detección y extracción de caracteres sobre tickets visa, ya que los estilos de fuente y tipografías utilizados para imprimir los tickets son muy específicos y varían en gran medida de unos tickets a otros.

Por lo tanto, en este apartado describiremos los pasos a seguir para crear un nuevo archivo de patrones que pueda ser utilizado desde la librería Tesseract versión 3.00 o superior, centrándonos especialmente en la adaptación para tickets visa. El proceso que se describe a continuación se ha realizado sobre un sistema GNU/linux (Debian/Ubuntu) con la librería Tesseract v3.02 instalada.

4.3.1 Requisitos de las imágenes de entrenamiento

Las características que deben cumplir las imágenes que vayan a ser utilizadas como imágenes de entrenamiento son las que se enumeran a continuación:

1. Hay que asegurarse que la imagen o el conjunto de imágenes que se utilizan para el entrenamiento contiene el conjunto completo de caracteres que deseamos reconocer. Además debe existir un número mínimo de muestras por cada uno de estos caracteres, por lo general deben existir 10 muestras por cada carácter, aunque podrían aceptarse 5 muestras en el caso de caracteres raros o aquellos que aparezcan con poca frecuencia. Por el contrario

aquellos caracteres que aparezcan con más frecuencia deberían tener unas 20 muestras.

2. Es fundamental que los caracteres dentro de las imágenes estén suficientemente espaciados para poder facilitar su detección de forma inequívoca, de este modo cada una de las muestras existente de cada uno de los caracteres podrá ser detectada de forma individual, evitando posibles solapes entre caracteres.

3. Las imágenes que se utilicen durante el proceso de entrenamiento deben estar agrupadas por tipo de letra, estilo de fuente o tipografía. Es muy importante agrupar todos los caracteres de una mismo tipo de fuente, ya sea en una sola imagen o en un conjunto de imágenes, pero lo que **nunca debe hacerse** es mezclar diferentes tipos de fuentes dentro de una misma imagen o dentro de otro conjunto de imágenes con otro tipo de fuente.

4. El formato de imagen que debe utilizarse durante el proceso de entrenamiento es TIFF, hasta un máximo de 32 páginas dentro del mismo archivo. Por lo general cada una de las imágenes suele identificarse con la siguiente nomenclatura **[lang].[fontname].exp[num].tiff** , donde:
    - **lang**: Identifica el idioma o nombre del fichero de patrones que se desea construir, en nuestro caso será **man** (patrón manual).
    - **fontname:** Identifica la tipografía o el tipo de fuente.
    - **exp[num]**: Identifica el número de la imagen de entrenamiento.

4.3.2 Obtención de imágenes de entrenamiento

Existen dos formas de proporcionar imágenes de entrenamiento a la librería Tesseract para la generación de un patrón de referencia. Por un lado existe la posibilidad de crear el patrón de referencia partiendo de las imágenes reales de tickets visa, y por otro lado tenemos la opción de crear imágenes que simulen la tipografía o tipo de fuente que contiene un ticket visa. Ambas opciones tienen sus ventajas e inconvenientes.

4.3.2.1 Imágenes reales

Este método se basa en utilizar un conjunto suficientemente amplio de imágenes reales de tickets donde podamos encontrar aproximadamente 10 muestras de cada uno de los caracteres que se desea

entrenar. En el apartado *3.1 CAPACITACIÓN Y ADAPTACIÓN DE LAS LIBRERÍAS* se definió el conjunto de caracteres que podían formar parte de un *ticket*. A continuación se muestra este conjunto de caracteres:

- "A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P, Q, R, S, T, U, V, W, X, Y y Z".
- *"a, c, e, h, n, o, p, r, s, t, u"*
- *"0,1,2,3,4,5,6,7,8,9"*
- *"* : / . ,"*

Este método ofrece la ventaja de proporcionar muestras de caracteres que se corresponden completamente con la realidad, por lo que ofrece la máxima precisión en los detalles de los caracteres a entrenar, y por lo tanto disminuye al mínimo el nivel de incertidumbre que existe entre la muestra proporcionada de cada carácter y su correspondencia con la realidad.

Los inconvenientes que nos encontramos con este método son varios, ya que debemos cumplir con las características comentadas en el anterior apartado *4.3.1 Requisitos de las imágenes de entrenamiento*:

- Es necesario tener un conjunto suficientemente amplio de imágenes reales de tickets donde podamos encontrar aproximadamente 10 muestras de cada uno de los caracteres a entrenar.
- Es fundamental la calidad y la nitidez con la que son tomadas estas imágenes, ya que los caracteres dentro de las imágenes deben estar suficientemente espaciados para poder facilitar su detección de forma inequívoca.
- Las imágenes que se utilicen durante el proceso de entrenamiento deben estar agrupadas por tipo de letra, estilo de fuente o tipografía. **Nunca deben mezclarse** caracteres con tipos de letra, estilo de fuente o tipografías diferentes.
- El formato de imagen que debe utilizarse durante el proceso de entrenamiento es TIFF, por lo que una vez adquirida la imagen deberemos convertirla a este formato.

A continuación se ilustra el proceso a seguir para generar un fichero de patrones partiendo de imágenes reales de tickets:

Figura 9: A la izquierda Ticket tipo 1 - Tipo fuente 1. A la derecha Ticket tipo 2 – Tipo fuente 2

Partiendo de un conjunto de imágenes de tickets lo suficientemente grande para cumplir con los requisitos comentados, primero deberemos separarlos por tipo de fuente y seguidamente renombrarlos con la nomenclatura definida en el apartado *4.3.1 Requisitos de las imágenes de entrenamiento*, por lo que quedarán de forma similar a:

Ticket con tipo fuente 1	Ticket con tipo fuente 2
man.tipo-fuente-1.exp0.tiff	man.tipo-fuente-2.exp0.tiff
man.tipo-fuente-1.exp1.tiff	man.tipo-fuente-2.exp1.tiff
man.tipo-fuente-1.exp3.tiff	man.tipo-fuente-2.exp2.tiff
...	...

Siguiendo la pasos que se presentan en los próximos apartados obtendremos al final del proceso un nuevo patrón de referencia que estará especialmente adaptado a la tipografía y al conjunto de caracteres que existen en un ticket visa.

PATRON MANUAL
man.traineddata

4.3.2.2 Imágenes simuladas

Otro método para la creación de estas imágenes de entrenamiento en formato TIFF, es mediante la utilización de un editor de textos como LibreOffice, donde podremos introducir las muestras

(aproximadamente 10) de cada uno de los caracteres a entrenar. En el apartado *3.1 CAPACITACIÓN Y ADAPTACIÓN DE LAS LIBRERÍAS* se definió el conjunto de caracteres que podían componer un ticket. A continuación se vuelven a mostrar:

- *"A, B, C, D, E, F, G, H, I, J, K, L, M, N, Ñ, O, P, Q, R, S, T, U, V, W, X, Y y Z"*.
- *"a, c, e, h, n, o, p, r, s, t, u"*
- *"0,1,2,3,4,5,6,7,8,9"*
- *"* : / . ,"*

Después de insertar el número de muestras de estos caracteres en el editor de textos, procederemos a la elección del tipo de fuente o tipografía. Es muy importante elegir la tipografía que más se aproxime al tipo de fuente con el que se imprimió el ticket. Para esta labor se han utilizado algunas herramientas online, que normalmente son utilizadas por diseñadores para discernir entre las múltiples tipografías existentes. Algunos de los recursos web consultados son:

- WhatTheFont[17]: Esta web reconoce el tipo de fuente partiendo de una imagen que se proporciona al servidor, el sistema mostrará los resultados de las fuentes más próximas.
- Idenfont[18]: Esta web tiene un enfoque diferente y encuentra la tipografía haciendo una serie de preguntas acerca del estilo de fuente que se desea reconocer.
- PrintWorks Bowfin[19]: Este sitio web ofrece varias herramientas en línea para ayudar a encontrar la fuente que se busca. El sistema es similar al de Identifont pero mucho más exacto y más fácil de usar.

Una vez se ha establecido la tipografía más próxima a la que realmente posee el tipo de ticket que se desea entrenar, es necesario utilizar nuestro editor de textos para establecer la tipografía elegida sobre todas las muestras de caracteres. Finalmente procederemos a exportar o imprimir el texto en formato PDF.

Finalmente obtendremos la imagen TIFF a partir del archivo PDF mediante la herramienta *convert*, tal como se muestra a continuación:

17 Página Web WhatTheFont: http://www.myfonts.com/WhatTheFont/, última visita diciembre 2012.
18 Página Web Idenfont: http://www.identifont.com/, última visita diciembre 2012.
19 Página Web PrintWorks Bowfin: http://www.bowfinprintworks.com/SerifGuide/serifsearch.php , última visita diciembre 2012.

```
jaime@jaime-laptop:~$ convert -density 300 -depth 4 ticket_tipo_1.pdf man.tipo-fuente-1.exp0.tiff
```

Para la obtención de la imagen anterior hemos seguido el siguiente proceso de transformación ODT→ PDF → TIFF. Claramente el inconveniente principal de este método es el nivel de incertidumbre que se introduce en la elección del tipo de fuente, por lo que es probable que las muestras de los caracteres proporcionados en la imagen TIFF no se ajusten completamente a las características reales que poseen los caracteres de los tickets.

4.3.3 Primer paso para la creación de un nuevo patrón

Para el proceso de creación del nuevo patrón de referencia partiremos de un conjunto de imágenes reales de tickets, tal como se ilustraba en el apartado 2.3.2.1. Para este primer paso, debemos tener todas las imágenes dentro de un mismo directorio y debemos haberlas renombrado con la nomenclatura apropiada. En nuestro caso los nombres tendrán un formato similar a ***man.tipo-fuente-X.expX.tiff***.

Llegados a este punto podremos ejecutar la librería Tesseract con las opciones que se muestran a continuación:

```
jaime@jaime-laptop:~$ tesseract  man.tipo-fuente-1.exp0.tiff  man.tipo-fuente-1.exp0  batch.nochop  makebox
Tesseract Open Source OCR Engine v3.02 with Leptonica
Page 0
```

Este comando creará un archivo de texto con la extensión BOX. Este archivo enumera de forma ordenada todos los caracteres de la imagen de capacitación, por lo que aparecerá un carácter en cada línea del archivo. También se recogen las coordenadas del rectángulo que determina la posición del carácter dentro de la imagen.

Este proceso debe ser repetido por cada una de las imágenes TIFF que existan en el directorio, por lo que al final de este paso debemos tener un archivo BOX *(man.tipo-fuente-1.expX.box)* por cada imagen TIFF *(man.tipo-fuente-1.expX.tiff)*.

4.3.4 Segundo paso para la creación de un nuevo patrón

En este punto del proceso tenemos que editar los ficheros BOX generados en la etapa anterior para corregir los posibles errores de detección o reconocimiento que haya cometido la librería Tesseract.

Para realizar modificaciones sobre los archivos de textos BOX generados en el paso anterior, podríamos utilizar un editor de textos simple, pero el proceso podría resultar muy laborioso ya que primero debemos identificar las coordenadas del carácter en la imagen y posteriormente observar si se ha producido algún error en la detección o el reconocimiento de ese carácter. Teniendo en cuenta que un ticket puede tener unos 250 caracteres que revisar, la tarea se podría volver demasiado pesada.

Para editar los archivos BOX utilizaremos un herramienta escrita en Phyton llamada *moshpytt* [20] que ha sido especialmente diseñada para editar estos archivos BOX.

```
jaime@jaime-laptop::~/moshpytt-dist$ ./moshpytt.py
```

Una vez descargada y desempaquetada la aplicación podremos ejecutarla, tal como se muestran en el comando anterior. Posteriormente procederemos a editar los archivos BOX mediante esta aplicación. A continuación se muestra la imagen de la aplicación cuando abrimos el archivo ***man.tipo-fuente-1.exp0.tiff***.

20 Para más información a cerca de moshpytt: http://code.google.com/p/moshpytt/

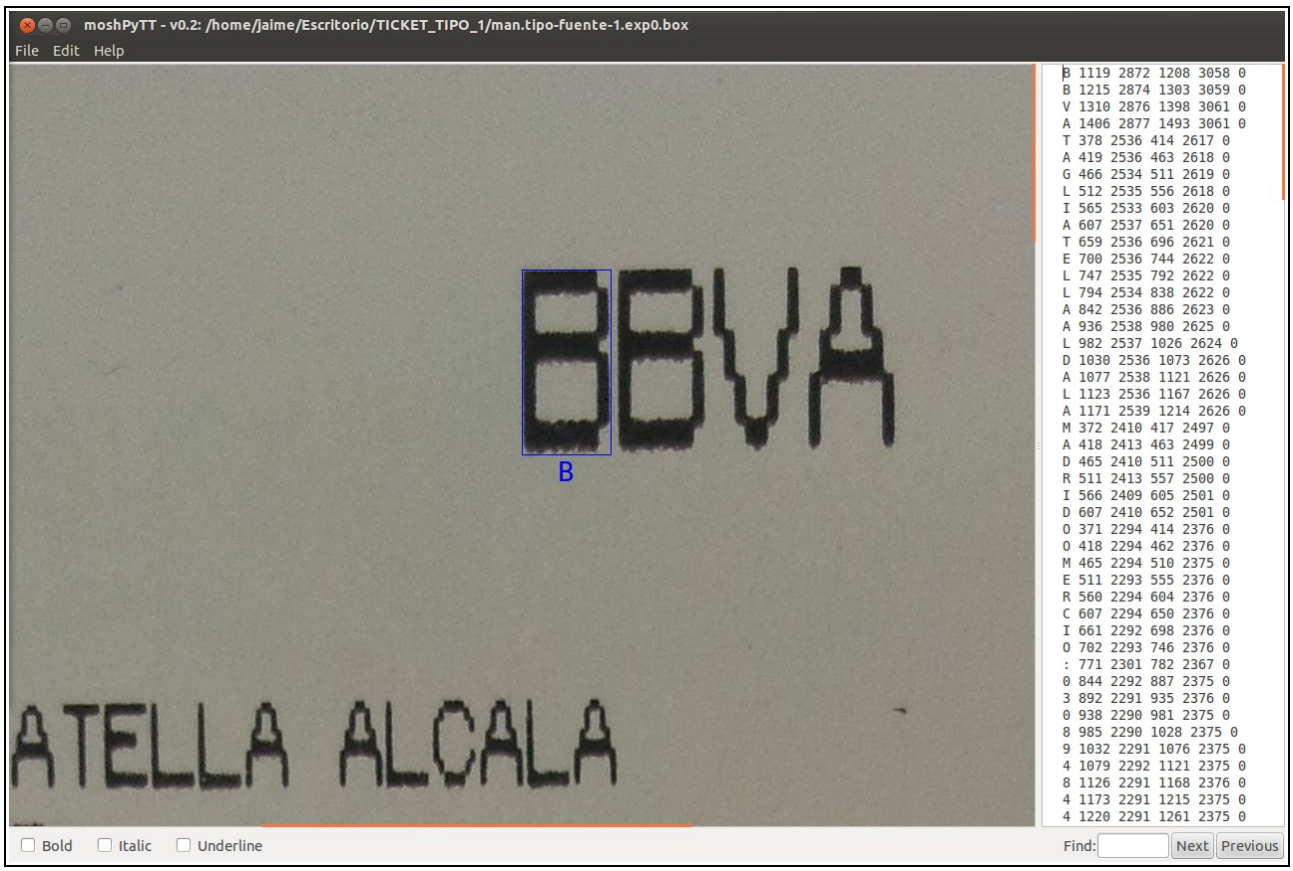

Figura 10: Aplicación moshpytt

Esta aplicación muestra en lado izquierdo la imagen ***man.tipo-fuente-1.exp0.tiff*** y en lado derecho la información generada en el paso anterior a partir de esta imagen, por lo que en la columna de la derecha se puede observar y editar el contenido del archivo ***man.tipo-fuente-1.exp0.box*.**

Esta aplicación tiene algunas características interesantes, por ejemplo muestra las letras mayúsculas en color azul y los números, caracteres especiales y letras minúsculas en color rojo.

Deberemos ir revisando cada uno de los caracteres que se muestran en la columna de la derecha y comprobando que se corresponde con el carácter que aparece resaltado en rojo o azul en la imagen. En caso de encontrar algún error modificaremos el carácter por el correcto, tal como se puede ver en la siguiente imagen donde deberemos sustituir la *D* por una *C*.

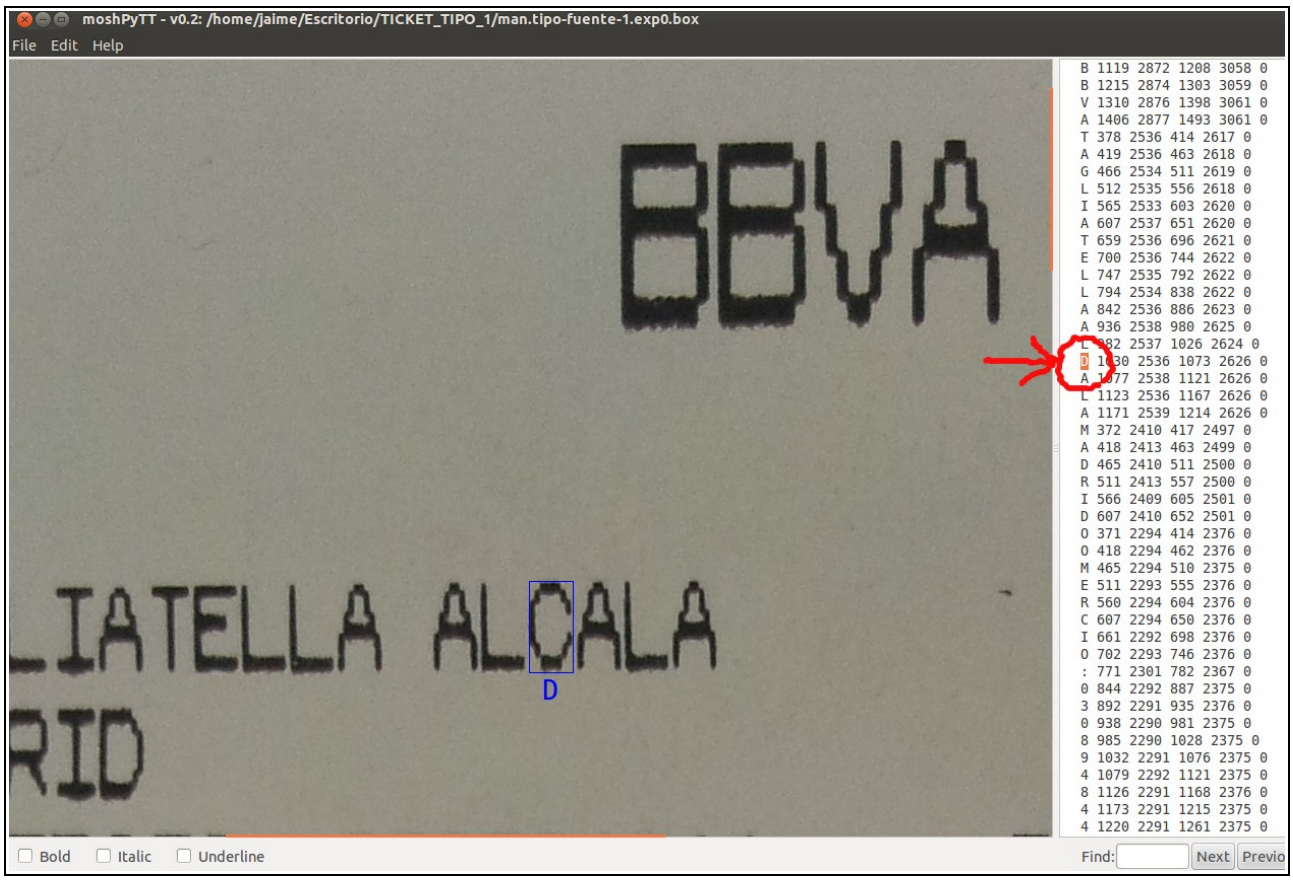

Figura 11: Aplicación moshpytt resaltando en rojo un carácter incorrecto

Este proceso debe ser repetido para revisar cada uno de los archivos BOX generados en el paso anterior.

4.3.5 Tercer paso para la creación de un nuevo patrón

Por cada una de las imágenes de entrenamiento TIFF (*man.tipo-fuente-1.expX.tiff*) proporcionadas, se ha creado un archivo BOX (*man.tipo-fuente-1.expX.box).*

Llegados a este punto y partiendo de la información proporcionada por ambos archivos vamos a proceder a realizar el proceso de entrenamiento, por lo que ejecutaremos el siguiente comando:

```
jaime@jaime-laptop:~$ tesseract  man.tipo-fuente-1.exp0.tiff  man.tipo-fuente-1.exp0  nobatch  box.train.stderr
Tesseract Open Source OCR Engine v3.02 with Leptonica
Page 0
row xheight=123.667, but median xheight = 67.5
row xheight=58.0667, but median xheight = 67.5
row xheight=56.5714, but median xheight = 67.5
row xheight=56.5714, but median xheight = 67.5
row xheight=58.0909, but median xheight = 67.5
```

```
row xheight=58.0909, but median xheight = 67.5
row xheight=124.667, but median xheight = 67.5
row xheight=32, but median xheight = 67.5
APPLY_BOXES:
   Boxes read from boxfile:     251
   Found 251 good blobs.
   Leaving 5 unlabelled blobs in 0 words.
TRAINING ... Font name = tipo-fuente-1
LearnBLob: CharDesc was NULL. Aborting.
Generated training data for 55 words
```

Una vez finalizada la ejecución de este comando se habrá creado un fichero con extensión TR (*man.tipo-fuente-1.exp0.tr*), partiendo de la imagen TIFF (*man.tipo-fuente-1.exp0.tiff*) y del archivo BOX (*man.tipo-fuente-1.exp0.box*)

Este proceso debe repetirse por cada pareja de archivos TIFF y BOX que existan.

Si el comando anterior tuviera algún problema durante el proceso de ejecución mostrará los correspondientes errores por la salida stderr, por lo que es importante revisarlos ya que un error en este paso puede ser fatal para la generación del nuevo archivo de patrones. Es importante comentar que el nombre de los archivos TIFF y BOX debe respetar la nomenclatura descrita y encontrarse sobre el mismo directorio. La salida de este paso es el fichero de texto ***man.tipo-fuente-1.expX.tr,*** que contiene las características que poseen cada una de las muestra de caracteres. También se crea el archivo ***man.tipo-fuente-1.expX.txt***, que únicamente contiene un salto de línea.

4.3.6 Cuarto paso para la creación de un nuevo patrón

Llegados a este punto, la librería Tesseract tiene que reconocer el conjunto de caracteres que finalmente contendrá el nuevo archivo de patrones. Para ello utilizaremos el comando *unicharset_extractor*, que extraerá el conjunto de caracteres existentes para los que debe entrenarse, para lo cual ejecutaremos el comando un *unicharset_extractor* sobre todos los archivos BOX.

```
jaime@jaime-laptop:~$ unicharset_extractor *.box
Extracting unicharset from man.tipo-fuente-1.exp0.box
Extracting unicharset from man.tipo-fuente-1.exp1.box
Extracting unicharset from man.tipo-fuente-1.exp2.box
Wrote unicharset file ./unicharset.
```

Mediante la ejecución de este comando la librería Tesseract crea el archivo ***unicharset***, donde se recopila la información sobre las propiedades de cada uno de los caracteres. Estas propiedades son

isalpha, isdigit, isupper, islower, ispunctuation. Estos datos son codificados en el archivo ***unicharset***, donde cada línea de este archivo corresponde a un carácter.

4.3.7 Quinto paso para la creación de un nuevo patrón

El siguiente paso en el proceso de entrenamiento consistirá en establecer los diferentes estilos de fuente para los que queremos entrenar la librería Tesseract. Para ello crearemos un fichero llamado ***font_properties;*** cada línea del archivo ***font_properties*** tendrá el siguiente formato *<fontname> <italic> <bold> <fixed> <serif> <fraktur>*.

El campo <fontname> es una cadena con el nombre del tipo de fuente, hay que comentar que no se permiten espacios en esta cadena de texto. Los campos <cursiva>, <bold>, <fixed>, <serif> y <fraktur> son simples banderas que indican si la fuente tiene esta propiedad, por lo que los valores permitidos son 0 o 1. La finalidad de este fichero es proporcionar información del estilo de la fuente que ha sido proporcionado durante los pasos anteriores. A continuación se muestra un ejemplo del posible contenido de este fichero:

```
jaime@jaime-laptop:~$jcat font_properties
tipo-fuente-1 0 0 0 0 0
tipo-fuente-2 0 0 0 0 0
……
```

Cuando se ejecuta *mftraining*, los nombres de los archivos TR generados anteriormente (*man.**tipo-fuente-1**.expX.tr*) deben coincidir con una entrada en el archivo ***font_properties***, o *mftraining* abortará. A continuación se muestra la ejecución de este comando y las opciones utilizadas para su puesta en marcha.

```
jaime@jaime-laptop:~$mftraining -F font_properties -U unicharset -O man.unicharset *.tr
Warning: No shape table file present: shapetable
Reading man.tipo-fuente-1.exp0.tr ...
Reading man.tipo-fuente-1.exp1.tr ...
Reading man.tipo-fuente-1.exp2.tr ...
Flat shape table summary: Number of shapes = 47 max unichars = 1 number with multiple unichars = 0
Clustering error: Matrix inverse failed with error 9.05821
Clustering error: Matrix inverse failed with error 4.02737
Clustering error: Matrix inverse failed with error 33.9961
Warning: no protos/configs for 2024 in CreateIntTemplates()
Done!
```

Una vez finalizada la ejecución del comando anterior se habrá creado un archivo de texto

man.unicharset, y tres archivos de datos: ***shapetable*** (que contiene los prototipos y la forma de los caracteres), ***pffmtable*** (que contiene el número de características que se esperan para cada carácter) e ***inttemp***.

Partiendo de la información de los archivos TR usaremos el comando *cntraining* para generar el archivo de texto ***normproto*** que contendrá los prototipos normalizados de cada uno de los caracteres. A continuación se muestra la salida generada con la ejecución de este comando:

```
jaime@jaime-laptop:~$ cntraining *.tr
Reading man.tipo-fuente-1.exp0.tr ...
Reading man.tipo-fuente-1.exp1.tr ...
Reading man.tipo-fuente-1.exp2.tr ...
Clustering ...

Writing normproto ...
```

4.3.8 Sexto paso para la creación de un nuevo patrón

En este paso se creará un diccionario de palabras frecuentes, de este modo informaremos a la librería Tesseract de aquellas palabras que se repiten constantemente en prácticamente todos los tickets analizados. Este conjunto de palabras fue identificado en la fase de análisis y diseño.

Para la creación de este diccionario de palabras frecuentes debemos crear el archivo ***frequent_words_list*** con el siguiente contenido:

```
jaime@jaime-laptop:~$ cat frequent_words_list
COMERCIO:
TPV:
APLIC.:
MASTERCAD
VISA
Tran:
Sec:
Aut:
Op:
Resp:
Fecha:
Hora:
```

Una vez introducido el conjunto de las palabras mostradas en el archivo ***frequent_words_list***, ejecutaremos el comando *wordlist2dawg* con las opciones que se muestran a continuación:

```
jaime@jaime-laptop:~$ wordlist2dawg frequent_words_list man.freq-dawg man.unicharset
Loading unicharset from 'man.unicharset'
Reading word list from 'frequent_words_list'
Reducing Trie to SquishedDawg
Writing squished DAWG to 'man.freq-dawg'
```

La ejecución del comando anterior creará el fichero de datos ***man.freq-dawg***, que contiene la información de nuestro diccionario de palabras frecuentes.

4.3.9 Séptimo paso para la creación de un nuevo patrón

En este último paso se creará el nuevo archivo de patrones, para ello simplemente vamos a proceder a recopilar toda la información creada en los pasos anteriores en un único archivo de datos. Para ello utilizaremos el comando *combine_tessdata*, aunque antes de poder usar este comando debemos renombrar algunos de los archivos creados en pasos anteriores, tal como se muestra a continuación:

```
jaime@jaime-laptop:~$ mv normproto man.normproto
jaime@jaime-laptop:~$ mv pffmtable man.pffmtable
jaime@jaime-laptop:~$ mv shapetable man.shapetable
jaime@jaime-laptop:~$ mv inttemp man.inttemp
jaime@jaime-laptop:~$ mv lang.unicharset man.unicharset
jaime@jaime-laptop:~$ mv lang.freq-dawg man.freq-dawg
```

Una vez tenemos los archivos con el nombre apropiado ejecutaremos el comando *combine_tessdata* con las opciones que se muestran a continuación.

```
jaime@jaime-laptop:~$ combine_tessdata man.
Combining tessdata files
TessdataManager combined tesseract data files.
Offset for type 0 is -1
Offset for type 1 is 140
Offset for type 2 is -1
Offset for type 3 is 3295
Offset for type 4 is 336706
Offset for type 5 is 337058
Offset for type 6 is -1
Offset for type 7 is -1
Offset for type 8 is -1
Offset for type 9 is 345065
Offset for type 10 is -1
Offset for type 11 is -1
Offset for type 12 is -1
Offset for type 13 is 345491
Offset for type 14 is -1
Offset for type 15 is -1
Offset for type 16 is -1
```

La ejecución de este comando ha generado finalmente un nuevo fichero de patrones llamado ***man.traineddata***, que es el nuevo patrón de referencia que posee la información para detectar el conjunto de caracteres que contenía las imágenes de ticktes proporcionadas, y para los estilos de fuente o tipografía definidas como tipo-fuente-1 y tipo-fuente-2.

4.3.10 Resultados ofrecidos por el nuevo patrón de referencia

Finalmente vamos a comprobar las mejoras que ofrece el nuevo patrón de referencia creado. Para ello realizaremos el proceso de detección y reconocimiento de caracteres sobre la imagen mostrada a la izquierda en la *Figura 9*.

Para poder utilizar el nuevo patrón de referencia, primero debemos ubicar el archivo ***man.traineddata*** en */usr/share/tesseract-ocr/tessdata/*.

A continuación ejecutaremos la librería Tesseract para utilizar el patrón de referencia ***man.traineddata*** y guardar la salida generada en el archivo ***salida-man.txt***:

```
jaime@jaime-laptop:~$ tesseract man.tipo-fuente-1.exp0.tiff salida-man -l man
Tesseract Open Source OCR Engine v3.02 with Leptonica
Page 0
```

Volvemos a ejecutar la librería Tesseract utilizando en este caso uno de los patrones de referencia general ***spa.traineddata*** y guardar la salida generada en el archivo ***salida-spa.txt*.**

```
jaime@jaime-laptop:~$ tesseract man.tipo-fuente-1.exp0.tiff salida-spa -l spa
Tesseract Open Source OCR Engine v3.02 with Leptonica
Page 0
```

Finalmente podemos observar las diferencias generadas por utilizar diferentes patrones de referencia. En color rojo se resaltan los errores que se han producido durante proceso de reconocimiento:

salida-man.txt generada con **man.traineddata**	**salida-spa-txt** generada con **spa.traineddata**
BBVA TAGLIATELLA ALCALA MADRID COMERCIO: 030894844 TPV: 00011646797 APLIC.: A0000000041010 MASTERCARD **********7508 NAVARRO SANTAPAU JAIME Tran:00164 Sec:00 VENTA Aut: 403460 Op: 008488 Resp: 00 Fecha:29.09.12 Hora:21:40 31 ,31 EUR	BBVA TAGLIATELLA AL**D**ALA MADRID **O**OMERCIO: 030894844 TPV**!** 00011646797 APLIC.: A000**D**000041010 MASTERCARD ***$*#$#*#$$?508 NAVARRO SANTAPAU JAIME Tran:00164 Sec:00 VENTA Aut: 408460 **09**: 008488 R**059**: 00 Fecha:29.09.12 Hora:21:40 31 **:**31 EUR

Tabla 18: Diferencia de resultados entre diferentes patrones de referencia

CAPÍTULO 5

CONCLUSIONES Y LÍNEAS FUTURAS

5. CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo de la memoria del presente trabajo final de máster, después de haber explicado cada una de las etapas realizadas, pasaremos a presentar el apartado final de este proyecto donde se exponen las conclusiones y las futuras líneas de trabajo. En primer lugar expondremos un resumen de las tareas realizadas, para posteriormente explicar los objetivos iniciales planteados en este proyecto y los resultados obtenidos. Finalmente se expondrán los posibles futuros trabajos a realizar, impulsados a partir de este proyecto.

El objetivo del presente trabajo final de máster era desarrollar una librería de reconocimiento óptico de caracteres para la plataforma Android, por lo que esta librería sería utilizada por diferentes aplicaciones ejecutadas en terminales móviles que cuenten con este sistema operativo.

5.1 TAREAS REALIZADAS

Para la consecución de este objetivo global se han ido abordando y finalizando cada una las tareas que se comentan a continuación:

Fase de Planificación

Plan de Trabajo: Definición de objetivos, tareas y planificación.

Fase de Análisis

Estudio inicial: Estudio del estado del arte de las librerías de SL.

Instalación y configuración librerías: Instalación y configuración de las librerías de SL identificadas mediante el estudio anterior.

Testing librerías: Pruebas de uso y funcionalidad de las librerías instaladas.

Fase de Diseño

Estudio cámara teléfono: Estudio de los diferentes modos de captura e identificación de las condiciones óptimas.

Generación imágenes objetivo: Generación de diferentes imágenes de muestra de *tickets* mediante la cámara del teléfono.

Capacitación librerías: Capacitar o enseñar a las librerías en el reconocimiento de las imágenes objetivo anteriores.

Elección librería: Elección de la librería más apropiada mediante datos objetivos (adaptabilidad, eficacia, eficiencia, licencias, etc)

Fase de Implementación

Migración y adaptación librería: Migración y adaptación de la librería elegida a la plataforma Android.

Creación aplicación: Desarrollo de una aplicación simple que haga uso de la librería desarrollada.

Manual para la capacitación de Tesseract: Manual paso a paso para generar un nuevo patrón de referencia para capacitar a nuestra librería en el proceso OCR para diferentes tipos de fuentes y formatos de texto.

Fase de pruebas y documentación

Pruebas y documentación de la aplicación: Identificación de errores, solución de errores y documentación del código mediante la herramienta Doxygen.

Pruebas y documentación de la librería: Identificación de errores, solución de errores y documentación del código mediante la herramienta Doxygen.

5.2 CONCLUSIONES

En general los resultados obtenidos a la finalización de este proyecto han sido muy satisfactorios en todos los ámbitos. En primer lugar se ha conseguido seguir y cumplir con la planificación propuesta, teniendo en cuenta que contaba con plazos y fechas muy ambiciosos. En segundo lugar se ha conseguido alcanzar prácticamente todos los objetivos propuestos.

A continuación se identifican los objetivos más relevantes que se han alcanzado y desarrollado a lo largo del proyecto:

- Investigación de diferentes técnicas de OCR.
- Estudio del estado del arte de las librerías OCR en el SL.
- Investigación del funcionamiento de la cámara en teléfonos móviles con Android.
- Estudio de la viabilidad de adaptación y capacitación de las librerías analizadas para nuevos patrones de referencia.
- Desarrollo e implementación de la librería para ser utilizada en teléfonos Android.
- Desarrollo e implementación de una aplicación que haga uso de esta librería.

Como punto fuerte de este proyecto hay que destacar que se ha conseguido implementar una solución software estable y funcional, que pretende ser una solución general que ofrezca la posibilidad para realizar el reconocimiento óptico de caracteres para diferentes finalidades, mostrando todas opciones disponibles. A continuación se enumeran algunas de las características más importantes de este software:

- Adaptabilidad en el proceso de adquisición de las imágenes, ya que dispone de algunas opciones de configuración como *Autofocus* o *Flash Led* que podemos activar a nuestro criterio. También ofrece un visor sobre la pantalla principal como el que se muestra en la *Figura 14* para limitar el área sobre la que deseamos realizar el reconocimiento óptico de caracteres.

- Adaptabilidad en el procesado OCR, ya que ofrece múltiples opciones de configuración para optimizar el proceso de reconocimiento óptico de caracteres que debe realizar la librería Tesseract como son: diferentes patrones de referencia, listas de caracteres permitidos , listas

de caracteres prohibidos y los diferentes modos de ejecución de la librería.

- Interfaz gráfico traducido a diferentes idiomas concretamente al español, catalán e inglés. Se han elegido estos idiomas para coincidir con los tres patrones de referencia utilizados por la aplicación, entre los cuales se encuentran los ficheros de patrones para los idiomas español (*spa.traineddata)*, inglés (*eng.traineddata*) y catalán (*cat.traineddata*). Por este motivo también se ha decidido adaptar y traducir la interfaz gráfica de la aplicación para poder trabajar en estos tres idiomas, facilitando de este modo la utilización de la aplicación por parte de usuarios que tengan como idioma materno alguno de los comentados.

- Documentación del código fuente mediante comentarios que pueden ser procesados por la herramienta Doxygen. Se ha creado un archivo de configuración, donde se recoge las opciones de configuración de la herramienta Doxygen para la generación de documentación en formato HTML del código fuente de la aplicación y de la librería, que podremos encontrar en el directorio /docs, junto con la última actualización de la documentación.

- Creación de una guía para la generación de nuevos patrones de referencia para facilitar de este modo, la posibilidad de construir nuevos patrones de referencia que ayuden en la adaptación del software a diferentes propósitos.

5.3 LÍNEAS FUTURAS

A la finalización de este proyecto y tras la fase pruebas se han obtenido un conjunto de resultados donde se observan algunas limitaciones que podrían ser mejoradas u optimizadas para ofrecer mejores prestaciones. A continuación se comentan algunas líneas de futuros trabajos para mejorar las carencias detectadas:

- Uno de los aspectos más críticos que influye directamente en los resultados ofrecidos por nuestra solución software es el proceso de adquisición de las imágenes, por lo que la óptica de la cámara y la calidad en el método de adquisición de las imágenes tienen una repercusión directa en el resultado obtenido. Por lo tanto cuanto menor sea la distorsión

introducida en el proceso de adquisición de imágenes mejor será el resultado obtenido.

Para mejorar esta situación existen diferentes líneas de trabajo que podemos abordar, por ejemplo concienciar y enseñar al usuario final en aquellos métodos más óptimos para la adquisición de las imágenes. También se puede limitar este software únicamente para aquellos terminales móviles más modernos que cuenten con una cámara con unas características mínimas apropiadas.

- Otro de los puntos críticos es la generación de un patrón de referencia general que sea capaz de detectar y reconocer los caracteres de forma correcta de la mayoría de los tickets que existen en el mercado. Tras el análisis y las pruebas realizadas se ha constado que casi todos los terminales TPV generan *tickets* con un tipo de fuente particular, por lo que si deseamos crear un patrón de referencia válido para todos los terminales TPV deberemos realizar un proceso de adaptación para cada uno de ellos. Para realizar este trabajo se necesitan imágenes de referencia tal como se comenta en el apartado *4.3.2 Obtención de imágenes de entrenamiento*, donde se muestran dos maneras de crear imágenes de entrenamiento y se comentan sus ventajas y desventajas.

  Para mejorar el proceso de creación de un nuevo patrón de referencia existe la posibilidad de realizar un estudio de los terminales TPV más extendidos en el mercado y realizar un proceso de adaptación exclusivo para ellos, de este modo se minimizaría el trabajo para la creación del nuevo patrón de referencia.

  En el caso de utilizar imágenes de muestra reales, tal como se comenta en el apartado *4.3.2.1 Imágenes reales*, es necesario conseguir un amplio conjunto de muestras de *tickets*, por lo que sería interesante disponer o tener acceso a los terminales TPV para la generación de muestras de textos.

  Otra línea de trabajo en caso de utilizar imágenes simuladas, tal como se comenta en el apartado *4.3.2.2 Imágenes simuladas*, es utilizar algún software que identifique con el menor margen de error posible el tipo de fuente con el que han sido escritos los *tickets* que deseamos incluir en nuestro nuevo patrón de referencia.

ANEXOS Y BIBLIOGRAFÍA

6. ANEXOS

ANEXO A – INFORMACIÓN ALGORITMOS OCR

El Reconocimiento Óptico de Caracteres (OCR) [19], o generalmente conocido como reconocimiento de caracteres, es un proceso dirigido a la digitalización de textos que identifica automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos y así poder interactuar con éstos mediante un programa de edición de texto o similar.

En los últimos años la digitalización de la información (textos, imágenes, sonido, etc) ha devenido un punto de interés para la sociedad. En el caso concreto de los textos, existen y se generan continuamente grandes cantidades de información escrita, tipográfica o manuscrita, en todo tipo de soportes. En este contexto, poder automatizar la introducción de caracteres evitando la entrada por teclado implica un importante ahorro de recursos humanos y un aumento de la productividad, al mismo tiempo que se mantiene, o hasta se mejora, la calidad de muchos servicios.

A.1 Problemas con el Reconocimiento Óptico de Caracteres

El proceso básico que se lleva a cabo en el Reconocimiento Óptico de Caracteres es convertir el texto que aparece en una imagen, en un archivo de texto que podrá ser editado y utilizado como tal por cualquier otro programa o aplicación que lo necesite.

Partiendo de una imagen perfecta, es decir, una imagen con sólo dos niveles de gris, el reconocimiento de estos caracteres se realizará básicamente comparándolos con unos patrones o plantillas que contienen todos los posibles caracteres. Ahora bien, las imágenes reales no son perfectas, por lo que el Reconocimiento Óptico de Caracteres se encuentra con varios problemas:

- El dispositivo que obtiene la imagen puede introducir niveles de grises al fondo que no pertenecen a la imagen original.

- La resolución de estos dispositivos puede introducir ruido en la imagen, afectando los píxeles que han de ser procesados.

- La distancia que separa a unos caracteres de otros, al no ser siempre la misma, puede provocar errores de reconocimiento.

- La conexión de dos o más caracteres por píxeles comunes también puede producir errores.

A.2 Esquema básico de un algoritmo de Reconocimiento Óptico de Caracteres

Todos los algoritmos de Reconocimiento Óptico de Caracteres tienen la finalidad de poder diferenciar un texto presente en una imagen cualquiera. Para hacerlo se basan en 4 etapas: binarización, fragmentación o segmentación de la imagen, adelgazamiento de los componentes y comparación con patrones.

A.2.1 Binarización

La mayor parte de algoritmos de OCR parten como base de una imagen binaria (dos colores) por lo tanto es conveniente convertir una imagen de escala de grises, o una de color, en una imagen en blanco y negro de tal forma que se preserven las propiedades esenciales de la imagen. Una forma de hacerlo es mediante el histograma de la imagen donde se muestra el número de píxeles para cada nivel de gris que aparece en la imagen. Para binarizarla tenemos que escoger un umbral adecuado, a partir del cual todos los píxeles que no lo superen se convertirán en negro y el resto en blanco.

Mediante este proceso obtenemos una imagen en blanco y negro donde quedan claramente marcados los contornos de los caracteres y símbolos que contiene la imagen. A partir de aquí podemos aislar las partes de la imagen que contienen texto (más transiciones entre blanco y negro).

A.2.1 Fragmentación o segmentación de la imagen

Este es el proceso más costoso y necesario para el posterior reconocimiento de caracteres. La segmentación de una imagen implica la detección mediante procedimientos de etiquetado determinista o estocástico de los contornos o regiones de la misma, basándose en la información de intensidad o información espacial.

Permite la descomposición de un texto en diferentes entidades lógicas que han de ser suficientemente invariables, para ser independientes del escritor, y suficientemente significativas para su reconocimiento.

No existe un método genérico para llevar a cabo esta segmentación de la imagen que sea lo suficientemente eficaz para el análisis de un texto. Aunque las técnicas más utilizadas son variaciones de los métodos basados en proyecciones lineales.

Una de las técnicas más clásicas y simples para imágenes de niveles de grises consiste en la determinación de los modos o agrupamientos ("clusters") a partir del histograma, de tal forma que permitan una clasificación o umbralización de los píxeles en regiones homogéneas.

A.2.3 Adelgazamiento de los componentes

Una vez aislados los componentes conexos de la imagen, se les tendrá que aplicar un proceso de adelgazamiento para cada uno de ellos. Este procedimiento consiste en ir borrando sucesivamente los puntos de los contornos de cada componente de forma que se conserve su tipología.

La eliminación de los puntos ha de seguir un esquema de barridos sucesivos para que la imagen continúe teniendo las mismas proporciones que la original y así conseguir que no quede deforme.

Se tiene que hacer un barrido en paralelo, es decir, señalar los píxeles borrables para eliminarlos todos a la vez. Este proceso se lleva a cabo para hacer posible la clasificación y el reconocimiento, simplificando la forma de los componentes.

A.2.4 Comparación con patrones

En esta etapa se comparan los caracteres obtenidos anteriormente con unos teóricos (patrones) almacenados en una base de datos. El buen funcionamiento del OCR se basa en gran medida en una buena definición de esta etapa. Existen diferentes métodos para llevar a cabo la comparación. Uno de ellos es el Método de Proyección, en el cual se obtienen proyecciones verticales y horizontales del carácter a reconocer y se comparan con el alfabeto de caracteres posibles hasta encontrar la máxima coincidencia.

Existen otros métodos como por ejemplo: Métodos geométricos o estadísticos, Métodos estructurales, Métodos *Neuro-miméticos*, Métodos *Markovianos* o Métodos de *Zadeh*.

ANEXO B – MANUAL DE USUARIO

Este manual de usuario se ha realizado para el idioma predeterminado de la aplicación, que en este caso es el español, aunque esta aplicación tiene adaptada la interfaz gráfica para los idiomas español, inglés y catalán.

B.1 Primer arranque

Una vez tengamos instalada la aplicación en nuestro terminal móvil, la primera vez que se inicie mostrará una pantalla como la que se muestra en la *Figura 12,* informándonos de los aspectos más importantes a tener en cuenta antes de empezar a usar la aplicación.

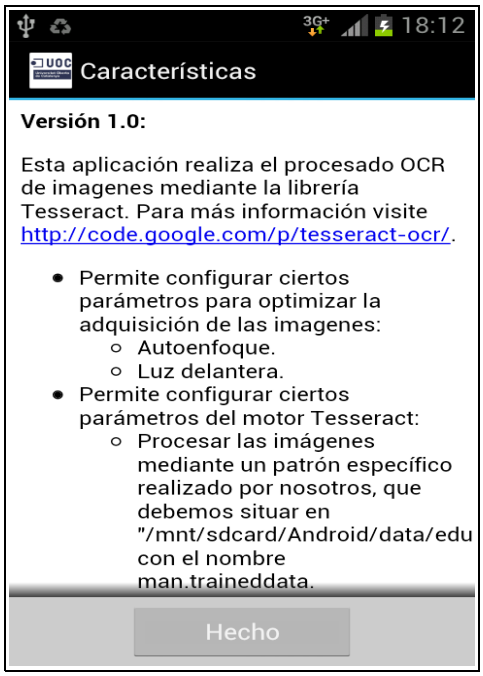

Figura 12: Pantalla inicial posterior a la instalación.

Durante el primer inició de la aplicación, esta debe tener acceso a internet para poder descargar desde http://tesseract-ocr.googlecode.com/files/ los archivos necesarios, que en este caso son el patrón de referencia para el idioma español (*spa.traineddata*) y patrón para la detección de la

orientación de la imagen (*osd.traineddata*).

En caso de no tener acceso a internet o haber eliminado el patrón de referencia manual (*man.traineddata*) aparecerá un aviso como el que se muestra en la *Figura 13*.

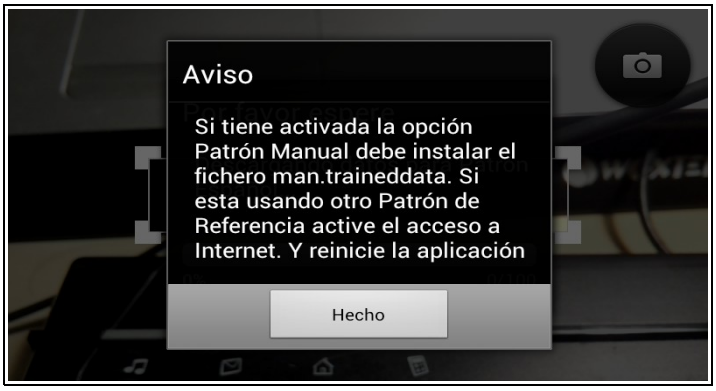

Figura 13: Pantalla de información en caso de error.

Si el proceso de descarga e instalación de los archivos se ejecuta correctamente, la aplicación conseguirá iniciarse correctamente y mostrará una imagen como la de la *Figura 14.*

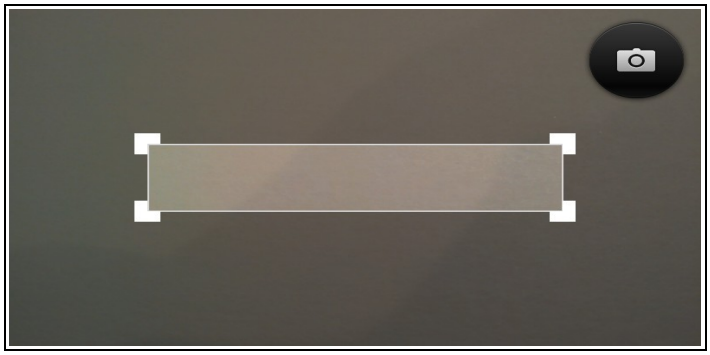

Figura 14: Pantalla principal de la aplicación.

B.2 Opciones

En la siguiente pantalla se muestran las opciones que aparecen en la aplicación cuando pulsamos en el terminal móvil la tecla que despliega los diferentes menús. Como se puede ver en la *Figura 15* a

través del botón de la izquierda accedemos a las opciones de configuración de la aplicación y desde botón de la derecha accedemos a información acerca de esta aplicación.

Figura 15: Pantalla que muestra los menús de la aplicación.

Al desplegar el menú que muestra **información a cerca de la aplicación** aparecerá la siguiente imagen tal como muestra la *Figura 16*. Si pulsamos sobre la opción de características se muestra una pantalla con información similar a la mostrada en la *Figura 12*.

Figura 16: Pantalla principal del menú "Acerca de OCR Configurable"

Al desplegar el menú que muestra **las opciones de configuración de la aplicación** se mostrarán las siguientes opciones de configuración que posee la aplicación, tal como aparece en las siguientes imágenes *Figura 17* y *Figura 18*.

En la *Figura 17* se muestran las opciones relativas a la configuración de la cámara del teléfono y del

sonido que emitirá la aplicación al capturar una imagen.

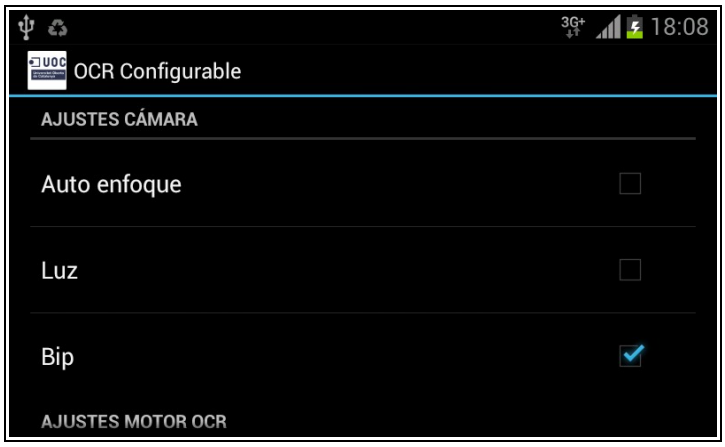

Figura 17:Pantalla principal del menú "Opciones de Configuración"

En la parte inferior de la *Figura 18* se muestra las diferentes opciones de configuración que ofrece la aplicación para adaptar el proceso de reconocimiento óptico de caracteres.

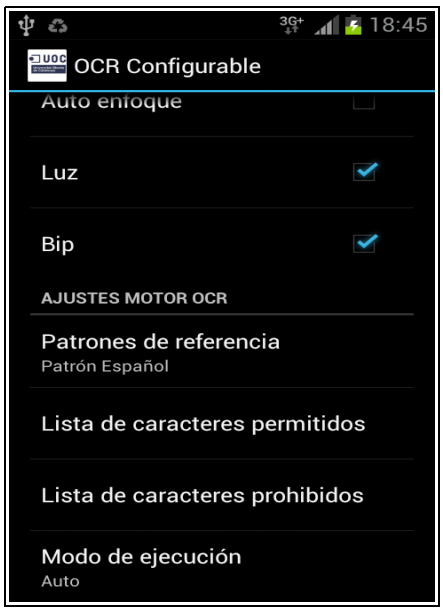

Figura 18:Pantalla principal del menú "Opciones de Configuración"

B.2.1 Opción - Patrón de referencia

Si pulsamos sobre esta opción de configuración se nos mostrará una pantalla como la que aparece en la *Figura 19*, donde podremos elegir entre los diferentes patrones de referencia que ofrece la

aplicación. Si seleccionamos alguno de los patrones generales Catalán o Inglés deberemos tener acceso a internet para realizar su descarga. Por el contrario si intentamos seleccionar la opción de patrón manual sin haber instalado el fichero *man.traineddata* en el directorio de trabajo de la aplicación *"/mnt/sdcard/Android/data/edu.uoc.ocr/files/mounted"* se nos mostrará una aviso como el de la *Figura 20*.

Figura 19:Menú desplegado con la opción "Patrones de referencia"

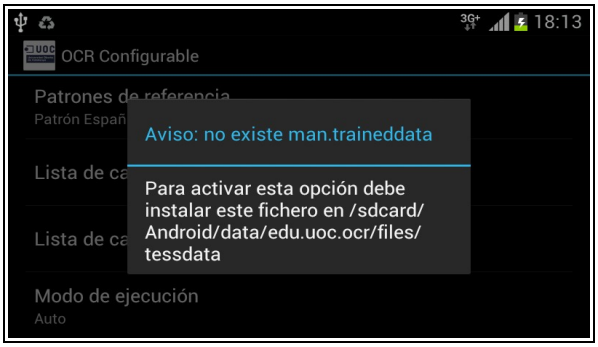

Figura 20: Aviso en caso de faltar el patrón man.traineddata

B.2.2 Opción - Lista de caracteres permitidos

Si pulsamos sobre esta opción de configuración se nos mostrará una pantalla como la que aparece en la *Figura 21*, donde podremos definir la lista de caracteres que van a aparecer en la imagen, de este modo la librería Tesseract se centrará en buscar coincidencias únicamente con la lista de caracteres introducidos.

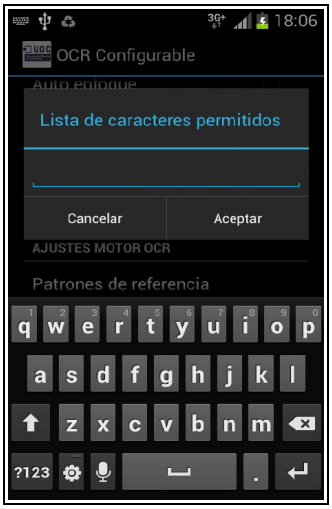

Figura 21: Menú desplegado con la opción "Lista de caracteres permitidos"

B.2.3 Opción - Lista de caracteres prohibidos

Si pulsamos sobre esta opción de configuración se nos mostrará una pantalla como la que aparece en la *Figura 22*, donde podremos definir la lista de caracteres que **no** van a aparecer en la imagen, de este modo la librería Tesseract evitará buscar coincidencias con la lista de caracteres introducidos.

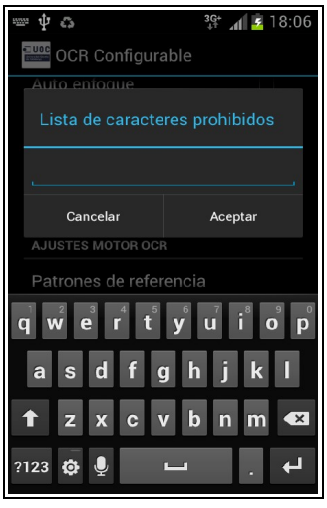

Figura 22: Menú desplegado con la opción "Lista de caracteres prohibidos"

B.2.3 Opción - Modo de ejecución

Si pulsamos sobre esta opción de configuración se nos mostrará una pantalla como la que aparece en la *Figura 19*, donde podremos elegir entre los diferentes modos de ejecución que ofrece la librería Tesseract. De forma predeterminada arrancará con el modo Auto.

Figura 23: Desplegable mostrado con la opción "Modo de ejecución"

Los diferentes modos de ejecución ofrecen la posibilidad de optimizar el proceso de OCR cuando las características del texto coincidan con alguno de ellos:

1. **Auto:** Modo de ejecución automático para la búsqueda sobre cualquier tipo de texto.
2. **Auto (sin OSD):** Modo de ejecución automático para la búsqueda sobre cualquier tipo de texto, sin realiza ningún tipo de comprobación sobre la orientación del texto.
3. **Un bloque de texto:** Modo de ejecución para la búsqueda sobre un bloque de texto uniforme.
4. **Solo un carácter:** Modo de ejecución para la búsqueda de un sólo carácter.
5. **Solo una columna:** Modo de ejecución para la búsqueda de una columna de texto uniforme.
6. **Solo una línea:** Modo de ejecución para la búsqueda de una línea de texto uniforme.
7. **Solo una palabra:** Modo de ejecución para la búsqueda de una única palabra.
8. **Un bloque vertical:** Modo de de ejecución para la búsqueda sobre un bloque vertical de texto uniforme.

B.3 Ejemplo de uso

Antes iniciar el proceso de reconocimiento de caracteres sobre *tickets* debemos instalar nuestro fichero de patrones (*man.traineddata*) sobre el directorio de la aplicación *"/mnt/sdcard/Android/data/edu.uoc.ocr/files/mounted"*. Una vez instalado el archivo podremos seleccionar la opción de patrón manual, por lo que nuestra pantalla de configuración quedará tal como muestra la *Figura 24*.

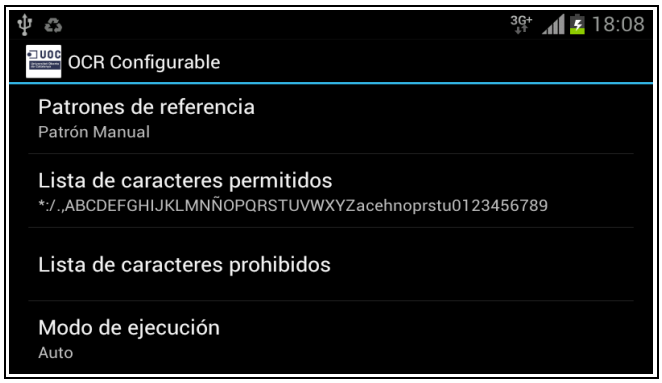

Figura 24: Pantalla mostrada al seleccionar el patrón manual

Seguidamente procederemos a iniciar el proceso OCR sobre un ticket por lo que primero posicionaremos el visor sobre la zona que deseamos reconocer y una vez tengamos centrada la imagen pulsaremos el botón de disparo, tal como muestra la *Figura 25.*

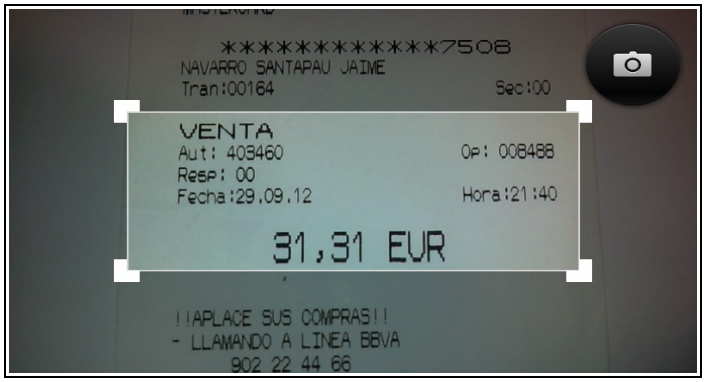

Figura 25:Pantalla principal con la imagen de un ticket

Una vez haya finalizado el procesado de la imagen se mostrará una pantalla como la de la *Figura 26*, donde se mostrará el resultado del proceso OCR. El texto que se ofrece como resultado sobre la parte izquierda de la pantalla puede ser compartido o copiado pulsando sobre él, por lo que si

pulsamos sobre el texto se nos mostrará una pantalla como la de la *Figura 27.*

Figura 26:Pantalla principal con los resultados del procesamiento OCR

Figura 27: Pantalla principal después de pulsar sobre el texto.

ANEXO C – PRESUPUESTO DEL PROYECTO

En este último punto, se muestra un posible presupuesto para este proyecto, donde se recogen los gastos previstos para su elaboración y ejecución.

A continuación se muestra el coste empresarial de diferentes perfiles profesionales. Se han calculado pensando en su salario bruto[21] anual, más el coste de mantener en plantilla a este profesional por parte de la empresa[22]. Además de los gastos mostrados también se deben tener presentes otros gastos como la provisión de despidos o los derivados de baja por enfermedad y por supuesto, el margen de benéfico que la empresa desearía obtener por cada perfil.

Perfil	Sueldo Bruto Anual	Seguridad Social (31-35%)	Vacaciones (10,5 %)	Seguro Obligatorio de Convenio	Recon. Medico	Otros (Portátil, teléfono, material de oficina, etc.)	**TOTAL(Euros)**
Jefe de Proyecto	45000	15750	4725	100	80	500	**66155**
Analista	33000	11550	3465	100	80	500	**48695**
Programador	26000	9100	2730	100	80	500	**38510**

Tabla 19:Coste de diferentes perfiles profesionales

Para el cálculo final del coste por jornada de cada perfil se ha tenido en cuenta el posible convenio colectivo de nuestra empresa[23], por lo que un año laboral consta de 220 jornadas laborales y estas jornadas constan de 8 horas laborales.

Perfil	**Coste/Jornada**	**Coste/Hora**
Jefe de Proyecto	300,70	37,59
Analista	**221,34**	**27,67**
Programador	175,05	21,88

Tabla 20:Coste por horas y jornadas de cada perfil profesional

21 Referencias para el calculo del salario bruto: http://salarios.infojobs.net/ y http://www.tusalario.es/main/espana-funcion-y-sueldo

22 Información acerca de los costes de personal para una empresa: http://blog.sage.es/contabilidad-fiscalidad-laboral/calcula-todos-los-costes-de-personal-con-la-nomina-en-la-sombra/

23 Posible convenio laboral: http://www.comfia.net/html/10307.html

Finalmente se define un pequeño presupuesto que tiene en cuenta el coste de los materiales hardware, software y el coste de realizar los trabajos por parte de un perfil de **analista**.

Concepto	Descripción	Unidades (Horas)	Total (Euros)
Elemento HW			
	Portátil	1	500
	Samsung Galaxy SII	1	425
Elementos SW			
	Licencias SL	10	0
Trabajos			
	Planificación	7,4	204,76
	Análisis	27,75	767,86
	Diseño	37	1023,79
	Desarrollo	55,5	1535,68
	Pruebas	46,25	1279,74
	Validación	11,1	307,14
TOTAL			**6043,97**

Tabla 21: Coste total del proyecto

ANEXO D – HERRAMIENTAS Y TECNOLOGÍAS UTILIZADAS

Para la elaboración de este proyecto se han utilizado o adquirido conocimientos sobre las siguientes herramientas o lenguajes:

- Java nivel alto
- Android
- C++ nivel alto

A continuación se definen las herramientas hardware y software que han sido utilizadas para llevar a cabo el desarrollo de este proyecto:

- Requisitos Hardware:
  1. Un Portátil con Ubuntu 10.04, RAM 4 GB, Procesador core duo 2,53GHz.
  2. Un teléfono Samsumg Galaxy SII, con Android 4.0.3 - Ice Cream Sandwich.

- Requisitos Software:
  1. **Linux Ubuntu v10.04.**
  2. **Android 4.0.3 - Ice Cream Sandwich.**
  3. **Eclipse Classic v3.7.2.**
  4. **Plugin ADT v20.0.3 para Eclipse.**
  5. **SDK Android v21.0.1.**
  6. **NDK Android v8b.**
  7. **Git v1.7** – Repositorios para código y documentación.
  8. **Doxygen v.7.6.1** – Documentación del código fuente.
  9. **OpenProj 1.4 –** Para la planificación y seguimiento del proyecto.
  10. **Tesseract v3.2**

11. **Leptonica v1.69**

ANEXO E – LICENCIAS DEL SOFTWARE UTILIZADO

En este anexo se enumeran las diferentes licencias de software libre de las que consta el código fuente o las librerías sobre las que se sustenta la solución software construida a lo largo de este TFM:

1. **Tesseract v3.2:** Apache License, Versión 2.0[24]
2. **Leptonica v1.69:** Creative Commons Attribution 3.0 United States License.[25]
3. **Tesseract-android-tools v1.0:** Apache License, Versión 2.0
4. **Zxing v4.0:** Apache License, Versión 2.0
5. **Android-ocr v0.5.13:** Apache License, Versión 2.0
6. **Eyes-free v0.6.1:** Apache License, Versión 2.0
7. **Jtar v1.0.4:** Apache License, Versión 2.0

Por lo tanto para poder utilizar, distribuir o modificar la solución software presentada en este TFM habría que respetar las condiciones que se detallen en las licencias enumeradas anteriormente.

24 Para más información sobre esta licencia: http://www.apache.org/licenses/LICENSE-2.0.html
25 Para más información sobre esta licencia: http://leptonica.com/about-the-license.html

7. BIBLIOGRAFÍA

[1] Wikimedia Foundation, Inc.: http://en.wikipedia.org/wiki/GOCR. Última visita Diciembre 2012.

[2] Proyecto GOCR: http://jocr.sourceforge.net/index.html. Última visita Diciembre 2012.

[3] SourceForge: http://sourceforge.net/p/javaocr/wiki/Home/. Última visita Diciembre 2012.

[4] Proyecto JavaOCR: http://sourceforge.net/projects/javaocr/. Última visita Diciembre 2012.

[5] Wikimedia Foundation, Inc.: http://es.wikipedia.org/wiki/Ocrad. Última visita Diciembre 2012.

[6] Proyecto Ocrad: http://www.gnu.org/software/ocrad/. Última visita Diciembre 2012.

[7] Wikimedia Foundation, Inc.: http://en.wikipedia.org/wiki/Tesseract_(software). Última visita Diciembre 2012.

[8] Proyecto Tesseract-Ocr : https://code.google.com/p/tesseract-ocr/, Última visita Diciembre 2012.

[9] Wikimedia Foundation, Inc.: http://en.wikipedia.org/wiki/CuneiForm_(software). Última visita Diciembre 2012.

[10] Proyecto Cuneiform: https://launchpad.net/cuneiform-linux/, Última visita Diciembre 2012.

[11] Wikimedia Foundation, Inc.: http://en.wikipedia.org/wiki/OCRopus. Última visita Diciembre 2012.

[12] Proyecto OCRopus : http://code.google.com/p/ocropus/. Última visita Diciembre 2012.

[13] Wikimedia Foundation, Inc.: http://en.wikipedia.org/wiki/OCRFeeder. Última visita Diciembre 2012.

[14] Proyecto OCRFeeder: https://live.gnome.org/OCRFeeder. Última visita Diciembre 2012.

[15] Wikimedia Foundation, Inc.: http://es.wikipedia.org/wiki/OpenCV. Última visita Diciembre 2012.

[16] Proyecto OpenCV: http://opencv.org/, http://sourceforge.net/projects/opencvlibrary/. Última visita Diciembre 2012.

[17] Samsung Galaxy SII: http://www.samsung.com/global/microsite/galaxys2/html/. Última visita Diciembre 2012.

[18] Documentación Proyecto Tesseract-Ocr: http://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3. Última visita Diciembre 2012.

[19] Wikimedia Foundation, Inc.: *http://es.wikipedia.org/wiki/Reconocimiento_óptico_de_caracteres.* Última visita Diciembre 2012.