UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Mangarizer: Aplicación Android para crear manga a partir de un archivo de vídeo

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Iván Sánchez Padilla

Tutor: Francisco J. Abad Cerdá

Cotutor: Tatsuya Harada (Universidad de Tokio)

Curso 2015/2016

Resumen

La proliferación los teléfonos inteligentes junto con la ubicuidad de redes inalámbricas de acceso a internet han dado origen a nuevas formas de comunicación, como pueden ser la compartición de fotos modificadas artísticamente o incluso cómics. Dichos cómics suelen utilizar imágenes obtenidas previamente o tomadas *ad hoc* con la cámara del dispositivo. En el presente documento se detalla la concepción y el desarrollo de una aplicación para dispositivos Android que permite al usuario la creación de un cómic con estética manga, utilizando para ello imágenes seleccionadas a partir de los fotogramas de un archivo de vídeo.

Palabras clave: aplicación Android, cómic, manga, vídeo.

Abstract

The proliferation of smart phones as well as the ubiquity of wireless networks providing Internet access have given rise to new forms of communication, such as sharing pictures artistically modified or even comics. These comics usually use images previously obtained or taken *ad hoc* with the device camera. The present paper details the conception and development of an Android application that allows the user to create a comic with manga aesthetics, using images selected from the frames of a video file.

Keywords: Android application, comic, manga, video.

Tabla de contenidos

1. Introducción

1.1. Presentación del problema

La expansión de los teléfonos inteligentes y la cuasipermanente disponibilidad de redes inalámbricas de acceso a internet han dado origen a nuevas formas de comunicación, como pueden ser la compartición de fotos mediante el uso de redes sociales o de mensajería instantánea. Se pueden encontrar numerosas aplicaciones móviles que permiten al usuario modificar artísticamente dichas fotos mediante el uso de diversos filtros y efectos. Algunas de estas aplicaciones permiten la creación de pequeños cómics a partir imágenes, pero en la actualidad no parece haber ninguna que permita utilizar archivos de vídeo como fuente para dichos cómics.

1.2. Motivación

Este proyecto surgió durante mi estancia como estudiante de intercambio en la Universidad de Tokio y viene motivado por mi afición al manga y mi interés por aprender a desarrollar aplicaciones para Android. Estas inquietudes unidas a la identificación de esta posible ampliación sobre las funcionalidades de los programas actuales motivaron el inicio de este proyecto.

1.3. Objetivos

El principal objetivo de este trabajo es el desarrollo de una aplicación para Android que permita al usuario la creación de un cómic con estética manga a partir de un archivo de vídeo capturado con la propia cámara del móvil. Este objetivo se puede desglosar en los siguientes subobjetivos:

- Se debe permitir al usuario seleccionar la distribución de las viñetas (entre una galería de distribuciones posibles).
- En cada viñeta se ubicará un fotograma del vídeo elegido por el usuario.
- Se aplicarán a las viñetas técnicas de renderizado no fotorrealista para dotar a las imágenes de una estética manga.
    - Detección de bordes.
    - Cuantización del color.
    - Aplicación de texturas (tramas).
- Se podrán situar bocadillos con texto en las viñetas.
- El cómic resultante podrá ser almacenado en el móvil y compartido en diversas redes sociales.

1.4. Características de la solución propuesta

- Permite utilizar imágenes extraídas de una lista (reducida) de los fotogramas de un vídeo almacenado en el dispositivo para la creación de una página de estética manga.
- Permite utilizar imágenes almacenadas en el dispositivo para la creación de una página de estética manga.
- Permite aplicar los efectos a una imagen almacenada en el dispositivo sin necesidad de crear una página.
- Permite elegir entre resolución reducida y resolución original cuando se aplican los efectos a una sola imagen.
- Aplica a las imágenes un algoritmo de detección de bordes.
- Aplica a las imágenes una cuantización del color.

- Reemplaza los colores originales de la imagen por 6 texturas distintas.
- Permite añadir bocadillos con texto a las páginas.
- Almacena la imagen resultante en el dispositivo con un nombre identificativo.
- Abre la imagen resultante con la aplicación galería del dispositivo, permitiendo así compartirla en redes sociales o por mensajería instantánea.

1.5. Dependencias

El programa requiere que el dispositivo tenga instalado un explorador de ficheros (se recomienda el ES File Explorer[1]), ya que se utiliza para seleccionar la imagen o vídeo a utilizar.

También se usa la galería de fotos del sistema.

1.6. Estructura de la memoria

Capítulo 2 - Antecedentes: Se presentan diversos productos comerciales relacionados con este trabajo, concretamente, las aplicaciones *Otaku Camera*, *Manga Camera* y *Comicize*. Así como una comparativa entre sus características principales y las de la solución propuesta.

Capítulo 3 - Análisis: Se ofrece una descripción detallada de la solución, pero sin hacer referencia concreta a la implementación ni a tecnologías específicas. Al final del mismo se puede encontrar un diagrama de flujo de la aplicación.

Capítulo 4 - Diseño: Se describe la solución aportada desde el punto de vista de la implementación, poniendo especial énfasis en el mangarizado.

Capítulo 5 - Resultado: Se confirma que se han cumplido con los objetivos marcados al inicio de este documento y se muestran una serie de ejemplos de uso de la aplicación.

Capítulo 6 - Conclusión: Se realiza un breve comentario sobre dificultades y hechos de interés ocurridos durante el desarrollo del proyecto, junto con una propuesta de ampliación.

[1] https://play.google.com/store/apps/details?id=com.estrongs.android.pop [Consulta: 1 de diciembre de 2015]

2. Antecedentes

En este capítulo se presentarán los productos comerciales existentes más relacionados con la temática de este trabajo final de grado.

2.1. Otaku Camera

Otaku Camera [1], de la compañía nipona Tokyo Otaku Mode Inc., es la aplicación más exitosa de las aquí referenciadas. Está disponible tanto para Android como para iOS y cuenta con cerca de 5 millones de descargas[2]. Estas son algunas de sus características:

- Permite modificar fotos tomadas previamente.
- Permite tomar y modificar fotos al momento.
- Permite añadir un marco a la imagen.
- Permite controlar en cierta medida los parámetros que afectarán al efecto manga de la imagen (con previsualización en tiempo real).
- Dimensiones de la imagen resultante: 480x480 píxeles.
- Permite guardar o compartir la imagen.

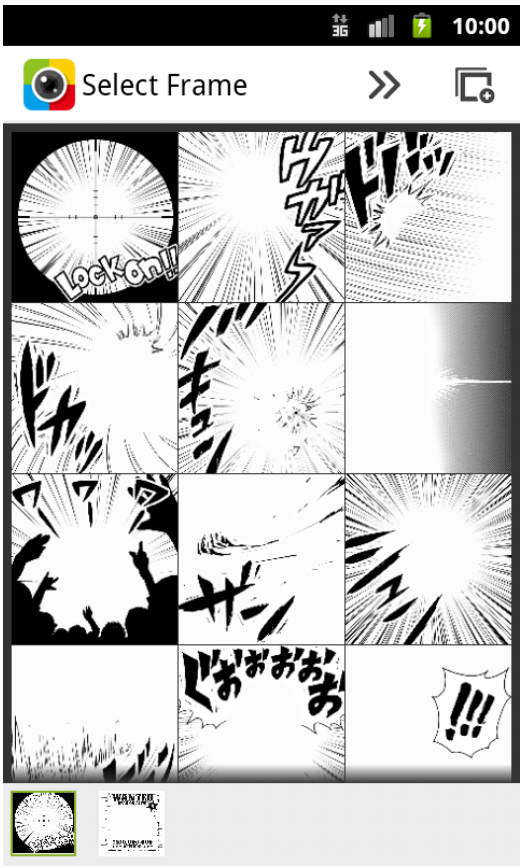

Figura 1: Selección de marco en Otaku Camera

Figura 2: Controles del efecto manga

[2] https://play.google.com/store/apps/details?id=com.otakumode.otakucamera [Consulta: 1 de diciembre de 2015]

2.2. Manga Camera

Manga Camera [2] viene ofrecida por la compañía Supersoftware Co. Ltd, también japonesa. Está disponible tanto para Android como para iOS, pero en su versión actual (1.3, de enero de 2014) no funciona correctamente en Android 5. Cuenta con entre 1 y 5 millones de instalaciones[3] y estas son sus principales características:

- Permite modificar fotos tomadas previamente.
- Permite tomar y modificar fotos al momento.
- Permite añadir un marco a la imagen.
- Permite controlar en cierta medida los parámetros que afectarán al efecto manga de la imagen (sin previsualización en tiempo real).
- Dimensiones de la imagen resultante: 480x640 píxeles.
- Permite guardar o compartir la imagen.

Figura 3: Selección de marco en Manga Camera

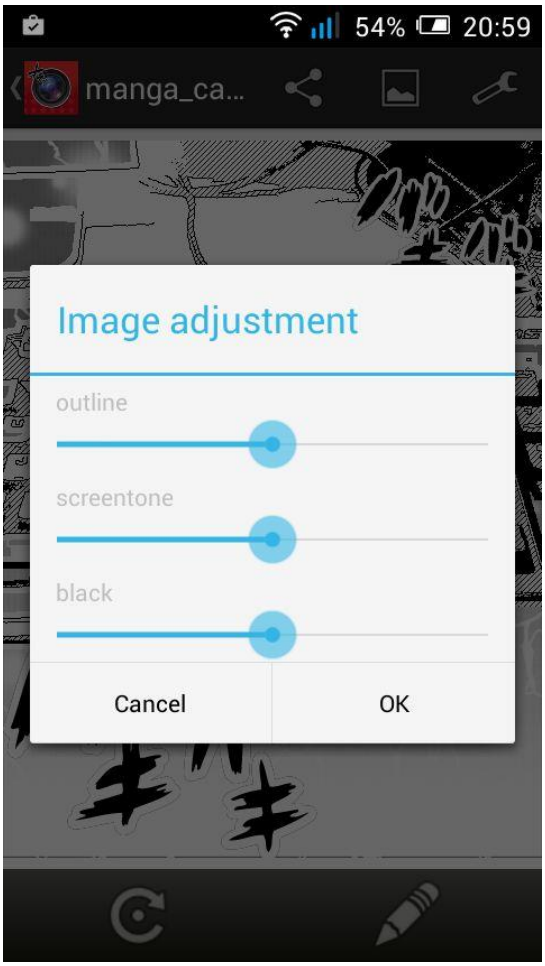

Figura 4: Controles del efecto manga

[3] https://play.google.com/store/apps/details?id=jp.co.supersoftware.mangacamera [Consulta: 1 de diciembre de 2015]

2.3. Comicize

Comicize [3] es la incorporación más reciente (20 de septiembre de 2015), y aún no existía al inicio de este proyecto. Está disponible para Android y de momento cuenta con entre 1000 y 5000 instalaciones[4]. Estas son sus principales características:

- Permite crear cómics.
- Permite utilizar fotos tomadas previamente o tomarlas al momento.
- Permite sobreimpresionar una serie de imágenes sobre la viñeta.
- Permite sobreimpresionar texto sobre la viñeta.
- Permite aplicar una serie de filtros a las imágenes (un efecto manga no está entre ellos)
- Dimensiones de la imagen resultante dependientes de la resolución de pantalla del dispositivo.
- Permite guardar o compartir las imágenes generadas.

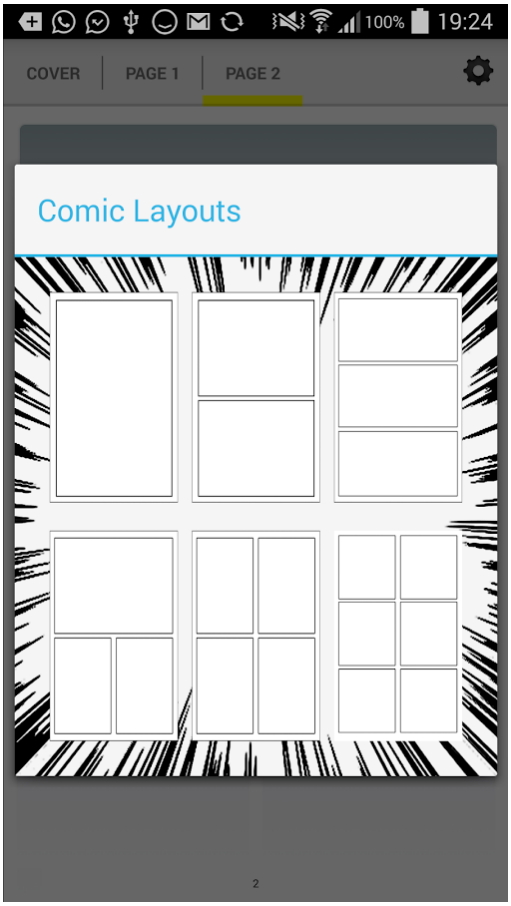

Figura 5: Selección de página en Comicize

Figura 6: Ejemplo de cómic en Comicize

[4] https://play.google.com/store/apps/details?id=com.graphicalize.comicize.app [Consulta: 1 de diciembre de 2015]

2.4. Tabla comparativa

A continuación, se presenta una tabla comparativa con las características de los productos mencionados y de la solución aportada. Se puede observar que dichos productos tienen una serie de carencias que la aplicación aquí presentada trata de suplir.

	Otaku Camera	Manga Camera	Comicize	Mangarizer
Permite modificar fotos tomadas previamente.	✓	✓	✓	✓
Permite tomar y modificar fotos al momento.	✓	✓	✓	✗
Permite añadir un marco a la imagen.	✓	✓	✗	✗
Permite controlar los parámetros del efecto aplicado.	✓	✓	✗	✗
Permite aplicar un efecto manga.	✓	✓	✗	✓
Permite aplicar otro tipo de efectos.	✗	✗	✓	✗
Permite crear un cómic.	✗	✗	✓	✓
Permite modificar una foto sin necesidad de crear un cómic.	✓	✓	✗	✗
Permite situar bocadillos con texto.	✗	✗	Indirecta-mente	✓
Resolución de la imagen resultante.	480x480	480x640	Depende dispositivo	Original / Depende*
Permite guardar y compartir la imagen resultante.	✓	✓	✓	✓

*Original si únicamente se modifica una foto, depende del dispositivo si se crea página manga.

3. Análisis

En este capítulo se ofrecerá una descripción detallada de la solución, pero sin hacer referencia concreta a la implementación ni a tecnologías específicas. Al final del mismo se puede encontrar un diagrama de flujo de la aplicación.

Al abrir la aplicación se mostrará un menú con dos ítems: el primero corresponderá a la opción *Mangarizar archivo* y el segundo a la opción *Crear manga*. Desde este menú también se podrá acceder a la ventana de configuración, que permitirá decidir si la opción *Mangarizar archivo* genera una imagen del tamaño de la original o si por el contrario se genera una de tamaño reducido.

3.1. Mangarizar archivo

Al seleccionar esta opción se pedirá al usuario que especifique el archivo a utilizar. La selección del archivo se realizará mediante un explorador de ficheros externo, de manera que no es necesario implementar esta funcionalidad en la aplicación *Mangarizer*.

Una vez seleccionado el archivo, comenzará el procesado de la imagen, consistente en la aplicación del algoritmo de Canny para detección de bordes [4], la conversión de la imagen de RGB a escala de grises, cuantización de los valores de gris y la aplicación de la trama adecuada en función del valor de gris cuantizado de cada píxel. Durante todo este proceso se mostrará al usuario un diálogo que informará de que la operación está en curso.

Una vez terminado el procesado de la imagen, esta se almacenará en el dispositivo y se mostrará automáticamente mediante el uso de la aplicación de galería propia del sistema Android (o a través de otra aplicación si así lo indica el usuario).

3.2. Crear manga

Al seleccionar esta opción, el usuario accederá a la pantalla de *Selección de página*.

3.2.1. Selección de página

En este momento el usuario deberá elegir la distribución de las viñetas sobre la página, entre una serie de modelos predefinidos. Esta pantalla mostrará miniaturas de dichas páginas (permitiendo desplazamiento por la colección en caso de no caber en la pantalla). Una vez seleccionada, se envía la información sobre distribución de viñetas elegida a la siguiente actividad, *Crear página*.

3.2.2. Crear página

Esta actividad servirá de vista previa para para el efecto final. La zona principal presentará la distribución de viñetas seleccionada previamente y, al presionar sobre cualquiera de ellas el usuario elegirá el archivo origen de la imagen que se emplazará en dicha viñeta. Dicho archivo origen podrá ser de dos tipos: una foto o un vídeo.

- En el primer caso, se volverá a la vista de la página, en la que ya se podrá encontrar la imagen seleccionada posicionada sobre en la viñeta adecuada.
- En el caso de utilizar un vídeo como origen, la aplicación pasará a una pantalla de *Selección de fotograma*, que mostrará un subconjunto de todos los fotogramas del vídeo. Cuando el usuario seleccione uno de los fotogramas, se volverá a la vista de la página, en

la que ya se podrá encontrar la imagen seleccionada posicionada sobre en la viñeta adecuada.

El proceso anterior deberá ser realizado tantas veces como número de viñetas contenga la página que se seleccionó previamente.

Esta pantalla también mostrará dos botones en la barra superior.

- El primero de ellos servirá para crear un nuevo bocadillo a partir del texto que el usuario tendrá que introducir en el cuadro de diálogo que aparecerá al pulsar sobre el botón. Una vez creado, el bocadillo se podrá desplazar sobre la página. También se permitirá cambiar de diseño del bocadillo, en función de la posición del hablante respecto a este. Debe ser posible eliminar un bocadillo si así lo desea el usuario.
- El segundo botón iniciará la creación del manga a partir de las imágenes seleccionadas. Se procesarán las imágenes una a una, de manera análoga al proceso realizado en *Mangarizar archivo*. Los bocadillos creados y bordes negros para las viñetas se añadirán a la imagen resultante, que se almacenará en el dispositivo y se mostrará automáticamente mediante el uso de la aplicación de galería propia del sistema Android (o a través de otra aplicación si así lo indica el usuario).

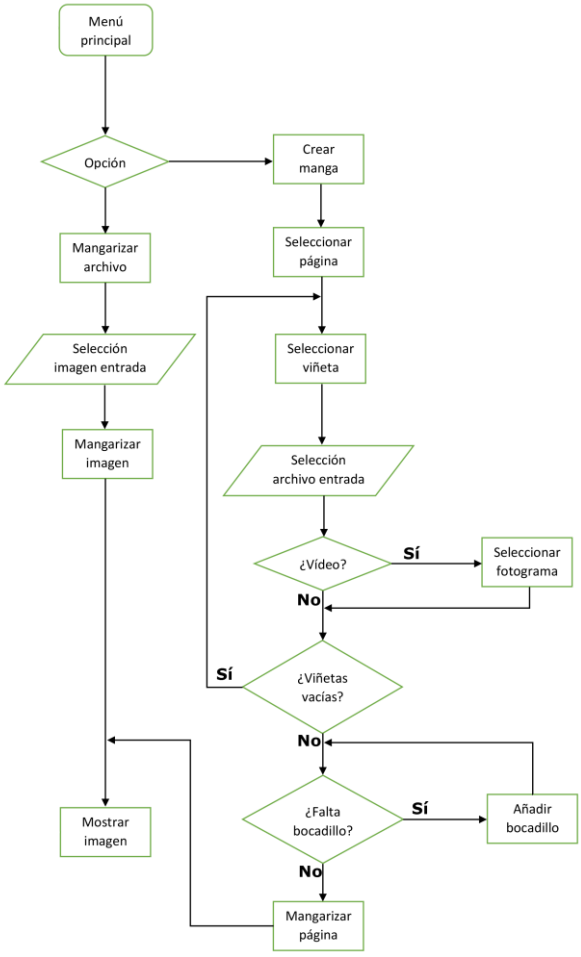

Figura 7: Diagrama de flujo de la aplicación

4. Diseño

En este capítulo se describe la solución aportada desde el punto de vista de la implementación. Se hace pone énfasis en la fase de mangarizado, por su especial relevancia e interés.

4.1. Análisis de las herramientas

Para el desarrollo de esta aplicación se han usado diversas herramientas y tecnologías. Algunas vienen impuestas por la plataforma de destino y otras se eligen por facilidad de uso, rapidez o dominio:

- Android Studio [5]: Es el entorno de desarrollo integrado oficial para Android. Multiplataforma, gratuito y publicado bajo licencia Apache 2.0. Se ha utilizado durante la mayor parte de la fase de implementación.
- Java [6]: Lenguaje de programación orientado a objetos. Multiplataforma, gratuito y publicado bajo licencia GNU GPL. Es el lenguaje que se utiliza habitualmente para programar en Android.
- JavaCV [7]: Interfaz que ofrece funciones de bibliotecas como OpenCV o FFmpeg. Publicado bajo licencia Apache 2.0
- OpenCV[8]: Biblioteca de visión por computador, multiplataforma y gratuita. Publicada bajo licencia BSD. Se ha utilizado para tratar las imágenes.
- FFmpeg[9]: Es una colección de software libre y bibliotecas destinados al manejo de datos multimedia. Publicada bajo licencia GNU LGPL. Se ha utilizado para la extracción de fotogramas del vídeo.
- Python[10]: Lenguaje de programación orientado a objetos. Multiplataforma, gratuito y publicado bajo licencia Python Software Foundation License. Se ha utilizado para el prototipado de la función mangarizar.
- Geany[11]: Editor de texto multiplataforma, gratuito y publicado bajo licencia GNU GPL. Se ha utilizado para programar en Python.
- WordPress[12]: Sistema de gestión de contenidos multiplataforma, gratuito y publicado bajo licencia GNU GPL. Se ha utilizado para crear un blog privado como ayuda al seguimiento y recopilación de información para el proyecto.
- Adobe Photoshop[13]: Editor de gráficos rasterizados, privativo, de pago y actualmente usa el modelo de *software como servicio* (SaaS). Se ha utilizado para la creación de los diversos elementos gráficos de la aplicación.

[5] http://developer.android.com/sdk/index.html [Consulta: 1 de diciembre de 2015]
[6] http://java.com/es/ [Consulta: 1 de diciembre de 2015]
[7] https://github.com/bytedeco/javacv [Consulta: 1 de diciembre de 2015]
[8] http://opencv.org/ [Consulta: 1 de diciembre de 2015]
[9] http://ffmpeg.org/ [Consulta: 1 de diciembre de 2015]
[10] https://www.python.org/ [Consulta: 1 de diciembre de 2015]
[11] http://www.geany.org/ [Consulta: 1 de diciembre de 2015]
[12] https://wordpress.org/ [Consulta: 1 de diciembre de 2015]
[13] http://www.adobe.com/products/photoshop.html [Consulta: 1 de diciembre de 2015]

- Microsoft Office[14]: Suite ofimática privativa y de pago. Actualmente está disponible bajo SaaS entre otras opciones. Se ha utilizado para redactar este documento.

4.2. Efectos de mangarizado

Como paso previo a la implementación del proyecto y a fin de determinar unos valores satisfactorios para la función Canny de OpenCV [5] o decidir las tramas que se aplicarían a las imágenes, se escribieron diversos prototipos en Python. Se muestran a continuación algunos resultados interesantes.

4.2.1. Detección de bordes

Ejemplo de barrido del umbral superior para la función Canny:

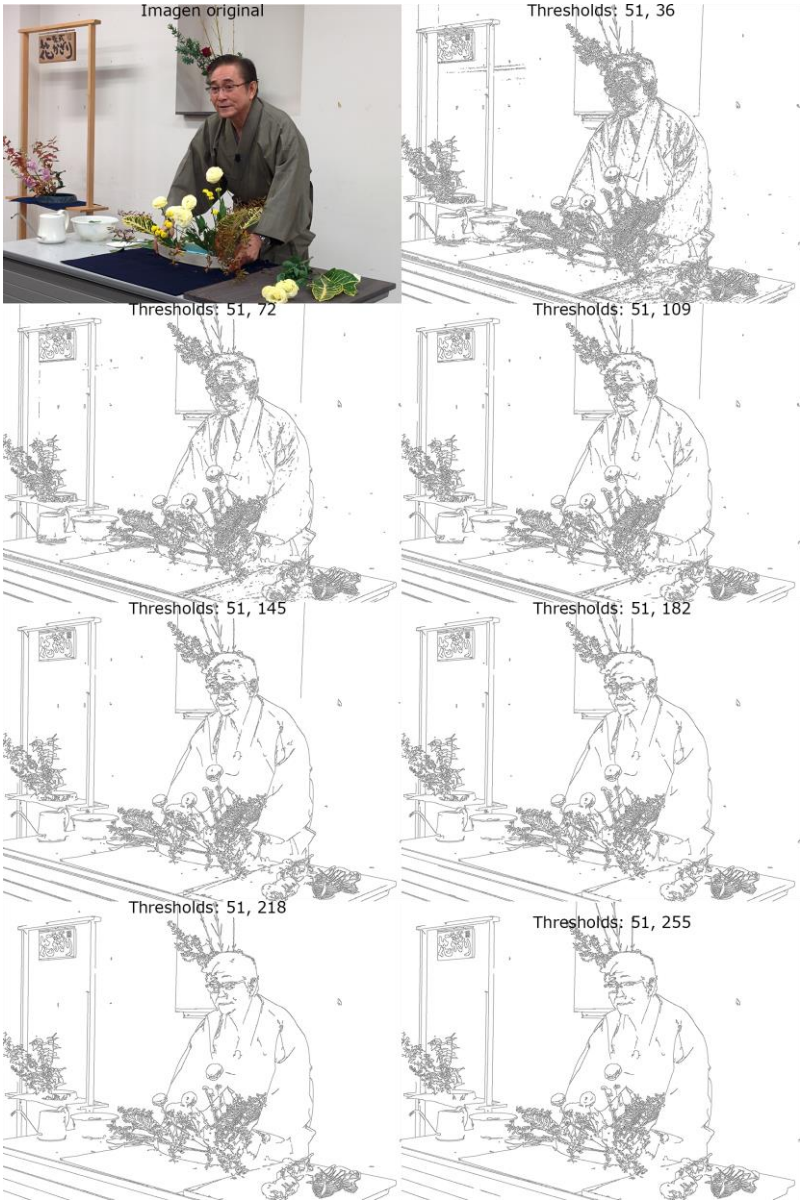

Figura 8: Ejemplo de barrido de umbral en la función Canny

[14] https://products.office.com/es-ES/ [Consulta: 1 de diciembre de 2015]

Finalmente se decidió fijar los valores umbral a 50 y 150 (la recomendación de Canny es que el umbral superior sea 3 veces el umbral inferior).

4.2.2. Cuantización del color

Para la cuantización del color también se hizo diversas pruebas. Se probó a realizar agrupamiento con el algoritmo K-Means y Mini Batch K-Means [6] (para 6 grupos, que pareció un número razonablemente alto para el objetivo que se buscaba). También se probó a realizar el agrupamiento pasando la imagen por distintos espacios de color (escala de grises, RGB y Lab), pero no se apreció una mejoría significativa en función del espacio de color empleado. En la siguiente figura podemos se puede observar la combinación de la detección de bordes y la cuantización del color.

Figura 9: Cuantización + detección de bordes

Finalmente, y para evitar la carga extra de estos algoritmos de agrupamiento en la aplicación móvil, se decidió crear 6 grupos de manera uniforme a lo largo de los 256 valores de la escala de grises: grupo 1 para valores de gris entre 0 y 256/6, grupo 2 para valores entre 256/6 y 2*256/6, etc.

4.2.3. Tramas

Para la selección de las tramas se creó una función que dibujaba una trama de líneas paralelas en función de una determinada pendiente, separación y grosor de línea; otra función que dibujaba líneas perpendiculares en forma de cruz, de una determinada separación y grosor; y una última función que las dibujaba en forma de aspa. A continuación, se muestran los distintos ejemplos obtenidos:

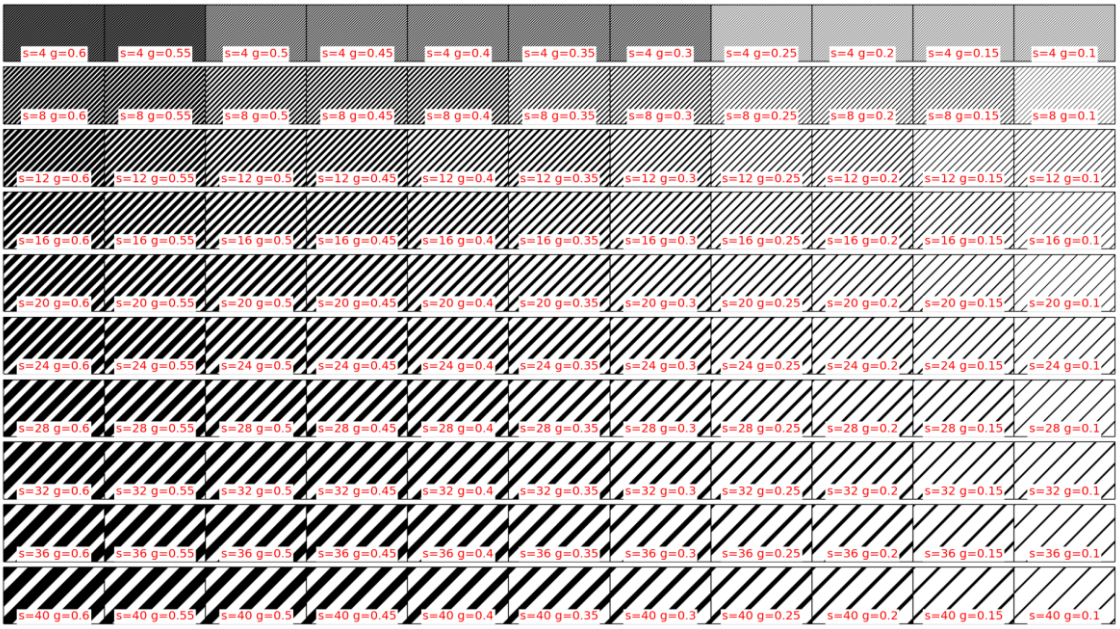

Figura 10: Tramas paralelas

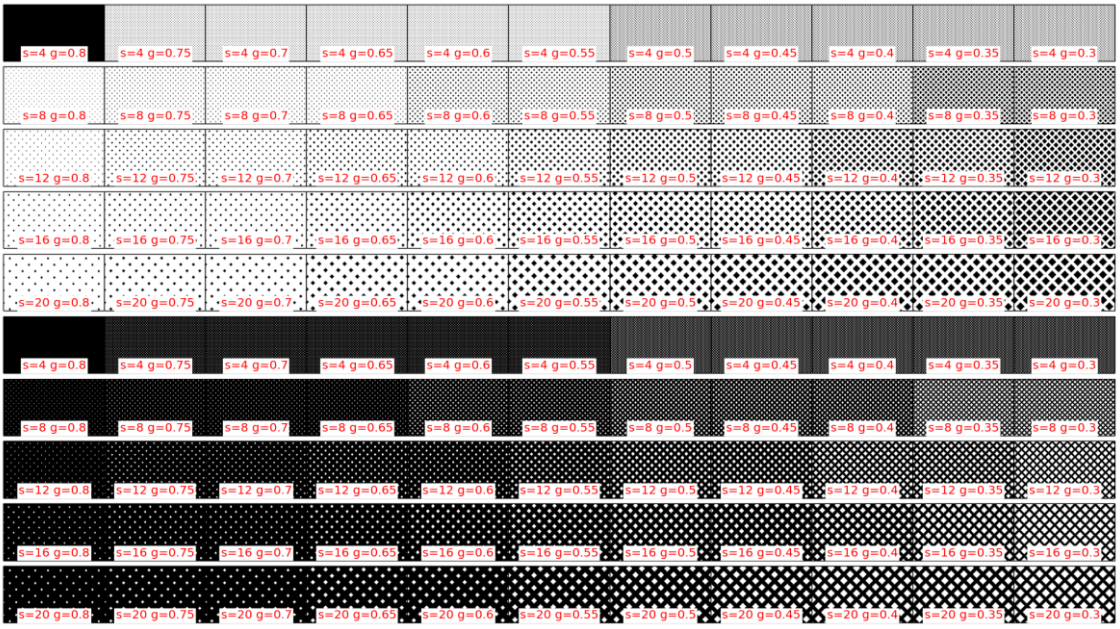

Figura 11: Tramas en aspa

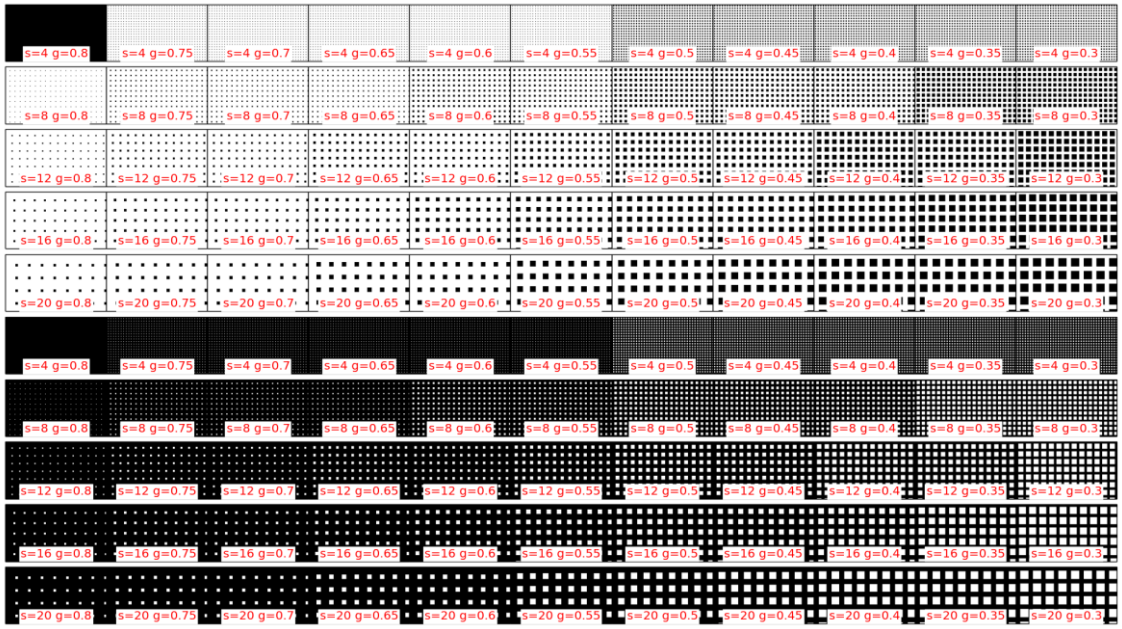

Figura 12: Tramas en cruz

Finalmente, se seleccionaron 6 tramas distintas, que son las que sustituirán a los 6 valores de gris obtenidos tras la cuantización:

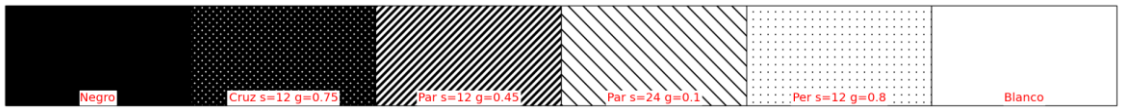

Figura 13: Tramas definitivas

A continuación, se muestran algunos ejemplos de fotos antes y después de aplicar las tramas.

Figura 14

Figura 15

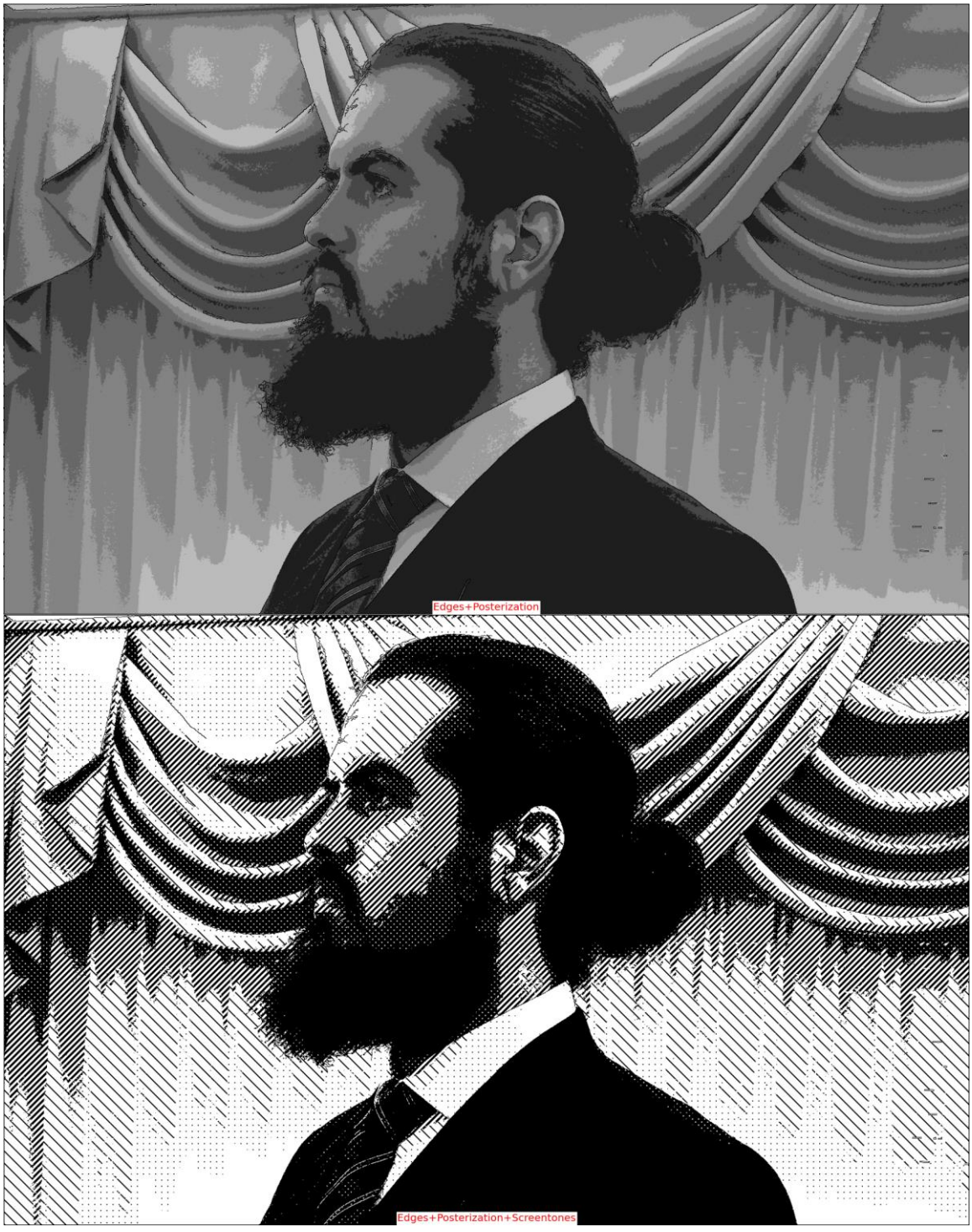

Figura 16

4.3. Algoritmo de mangarizado

4.3.1. Consideraciones previas

Parece interesante destacar, antes de hablar del algoritmo en cuestión, cómo se decide (sin ningún tipo de sentencia condicional, a fin de acelerar el proceso) qué trama debe aplicarse a cada píxel de la imagen en función de su valor de gris.

Inicialmente se define una interfaz *Screentone* y 6 clases St0, …, St5, una por trama (correspondiendo St0 con la trama negra y St5 con la blanca), que implementarán un único método: int getScreentoneValue(int i, int j). Dicho método devuelve el valor que tendría un píxel en la posición (i,j) de la imagen, si se le aplicara dicha trama.

Para cada trama, el valor de color de un píxel (blanco o negro en este caso) depende únicamente de la posición del píxel. Ejemplo de posible función trama:

$$putScreentone(i,j) = \begin{cases} 255, & \text{SI i\%4} < 3 \text{ OR j\%4} < 3 \\ 0, & \text{RESTO} \end{cases}$$

Con esta función, la trama consistiría en líneas blancas perpendiculares de grosor 3, con un píxel de separación entre ellas (entre las paralelas, obviamente).

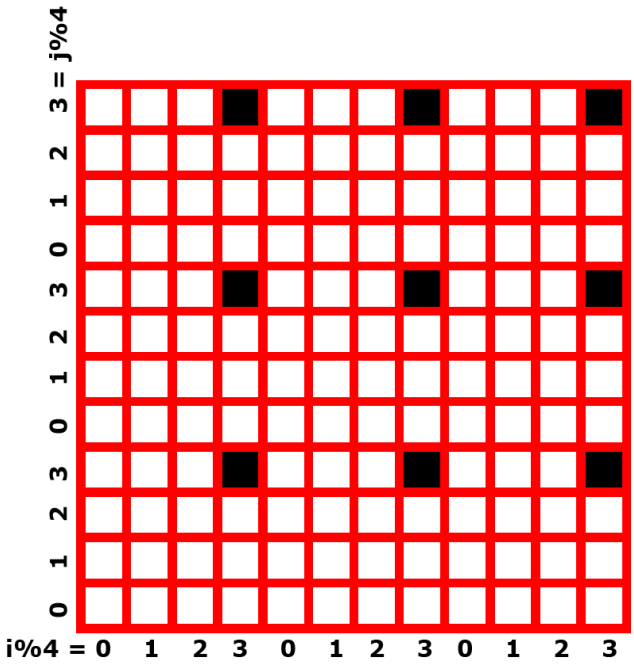

Figura 17: Ejemplo de trama

Se crea un vector de la clase interfaz *Screentone*, tamaño 6. Cada posición contiene una instancia de las clases St0, ..., St5. Puesto que las tramas están ordenadas por claridad ascendente, siendo la trama 0 negra y la trama 5 blanca, para averiguar qué trama se aplica a un valor de gris *pixValue*, basta con realizar la operación stId = pixValue*6/256. Para una entrada entre 0 y 255, esta operación devuelve un valor entre 0 y 5.

Por todo lo anterior, obtener el nuevo valor de color de un píxel dado tras aplicar la trama se reduce a realizar la operación

*stvalue = screentoneArray[pixValue*6/256].getScreentoneValue(x,y)*

4.3.2. Algoritmo

1. La función recibe una imagen origen *img*, anchura *w* y altura *h*.
2. Se inicializa el vector de la clase interfaz Screentone descrito en el punto anterior.
3. Se crean variables imagen *imgray*, *imcanny* e *imresult* de dimensiones *w* x *h*.
4. Se redimensiona *img* a *w* x *h*.
5. Se convierte *img* a escala de grises y se almacena en *imgray*.
6. Se detectan los bordes de *imgray* y se almacena la imagen resultante en *imcanny*.
7. Se crean un número predefinido de hilos que se repartirá el tratamiento de la imagen de forma paralela y se lanza su ejecución.
8. Para cada hilo:
    a. Para cada fila de imagen asignada al hilo:
        i. Para cada columna de la imagen:
            1. Se invierte el valor de color de imcanny (ya que se quieren los bordes negros, no blancos).
            2. Se obtiene el valor del píxel tras aplicar la trama adecuada (método descrito en el punto anterior).
            3. El valor de ese píxel en *imresult* es la operación AND bit a bit de los dos valores anteriores (se busca mantener los negros).
9. Se sincronizan los hilos y se prosigue con la ejecución secuencial.
10. Se libera el espacio reservado por *imgray* e *imcanny*.
11. Se devuelve *imresult*.

5. Resultados

A la vista de los siguientes resultados se puede observar que se han cumplido con los objetivos marcados al inicio de este documento. A continuación, se muestran una serie de ejemplos de uso de la aplicación:

5.1. Mangarizar archivo (tamaño reducido)

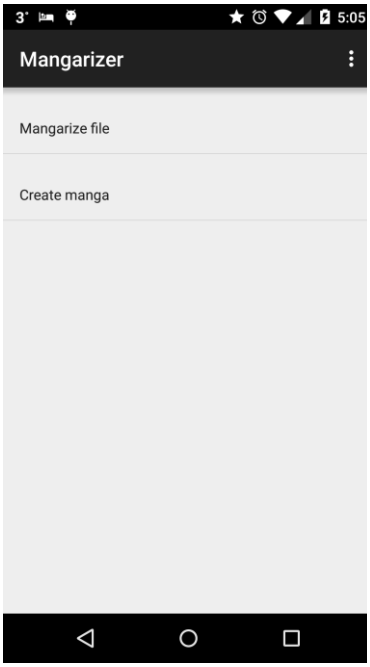

Figura 18: Menú principal

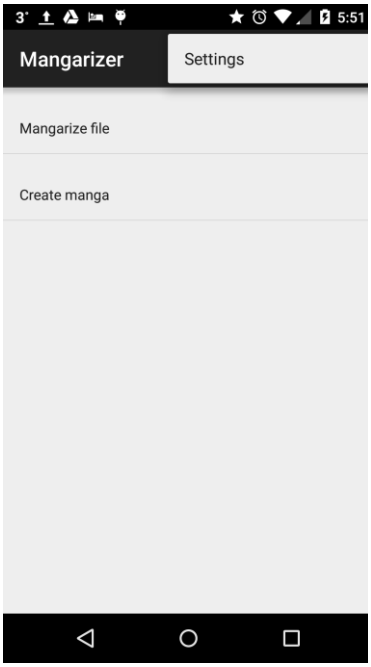

Figura 19: Accediendo a la configuración

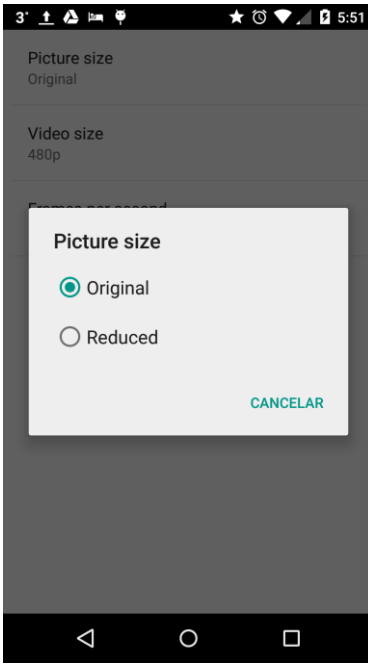

Figura 20: Eligiendo tamaño reducido

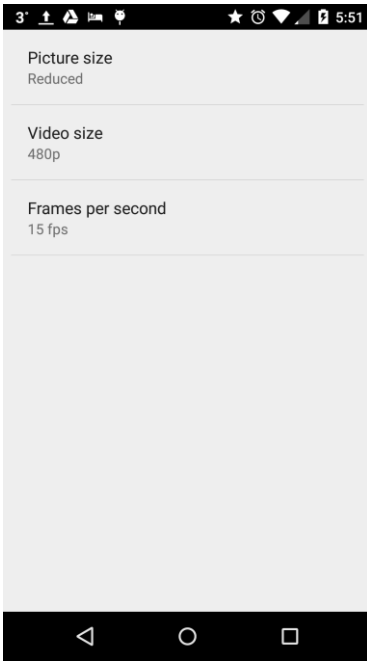

Figura 21: Tamaño reducido seleccionado

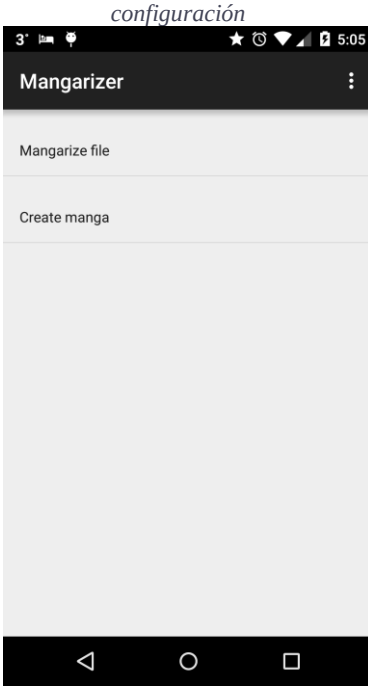

Figura 22: Seleccionar Mangarize file

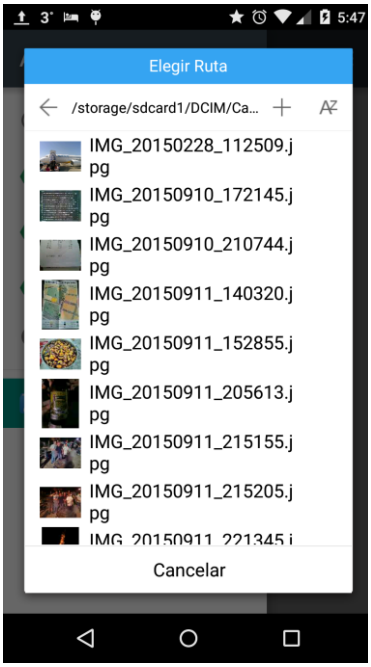

Figura 23: Seleccionar archivo deseado

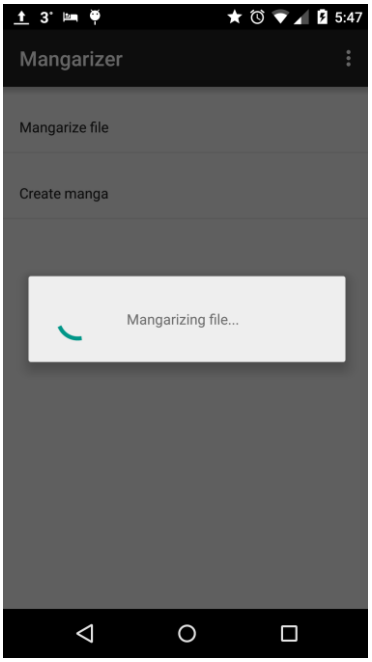

Figura 24: Esperando a que finalice el proceso

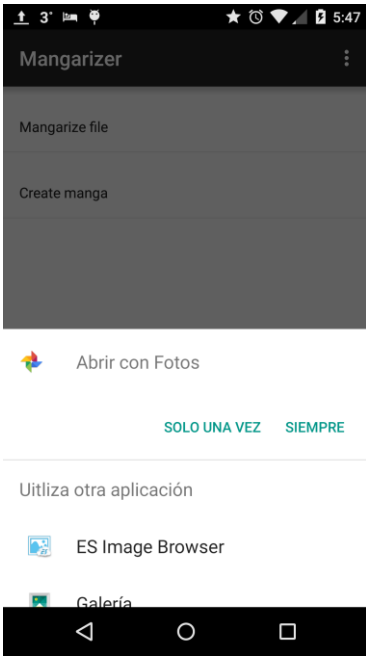

Figura 25: Elegir aplicación para visualizar y compartir el resultado

Figura 26: Resultado mostrado en la galería

Figura 27: Imagen original

A partir de la foto de 3200 x 2368 píxeles que se observa en la *Figura 16* se obtiene la imagen de 1672 x 1238 píxeles que se muestra a continuación:

Figura 28: Imagen mangarizada

5.2. Mangarizar archivo (tamaño original)

De manera análoga al ejemplo anterior, pero seleccionando mantener el tamaño original, se obtiene la siguiente imagen de 3200 x 2368 píxeles:

Figura 29: Imagen mangarizada

5.3. Crear manga (con un vídeo como archivo origen)

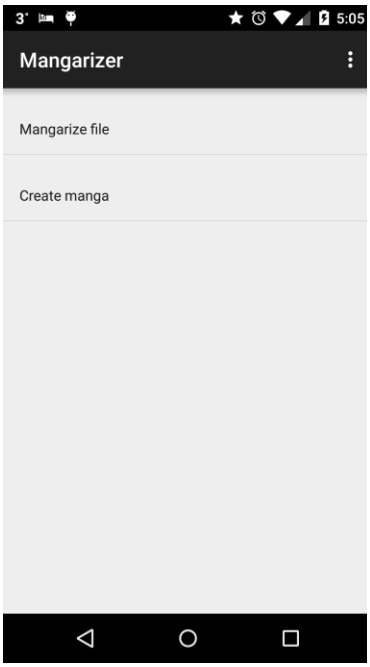

Figura 30: Menú principal

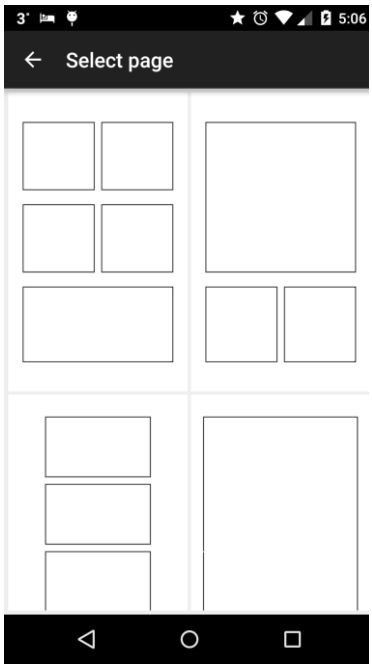

Figura 31: Selección de página

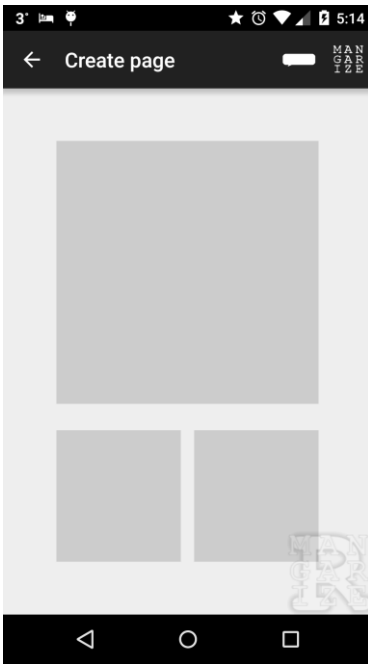

Figura 32: Creación de página

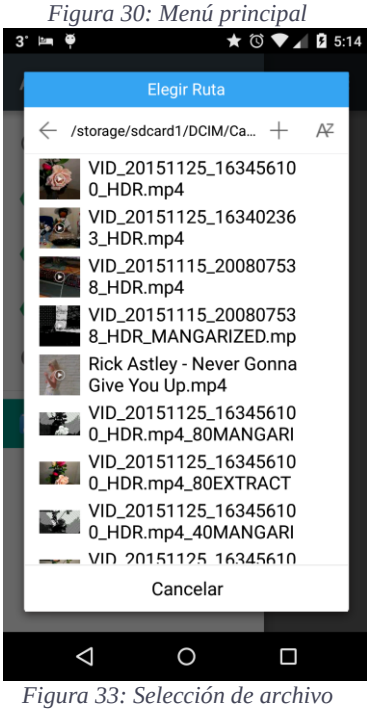

Figura 33: Selección de archivo origen

Figura 34: Selección de fotograma

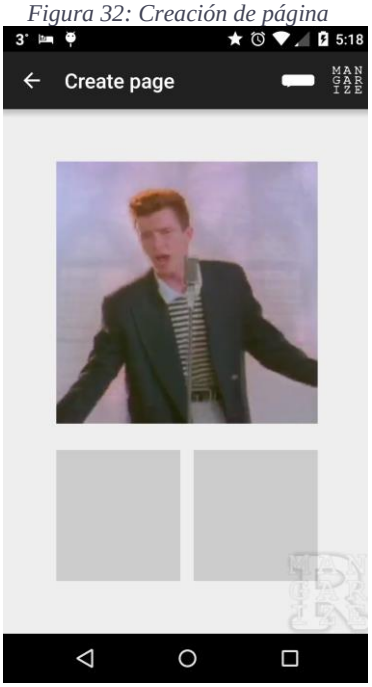

Figura 35: De vuelta a creación de página

Tras repetir los pasos de selección de fotograma para todas las viñetas:

Figura 36: Todas las viñetas con imagen

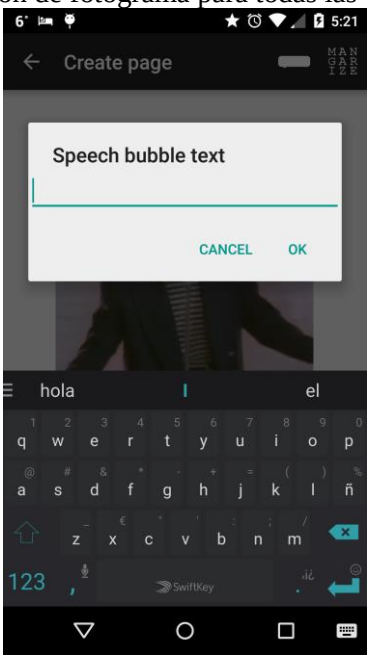

Figura 37: Tras pulsar el botón de bocadillo

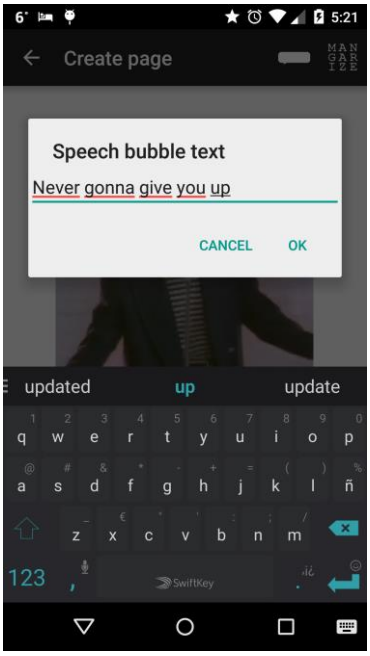

Figura 38: Se introduce el texto deseado y se acepta el diálogo

Figura 39: Bocadillo insertado

Se repite el proceso para tantos bocadillos como se deseen. Si es necesario eliminar un bocadillo, basta con arrastrarlo fuera de la pantalla.
Al pulsar sobre un bocadillo cambia su diseño para que pueda variar la posición del emisor del mensaje.

Figura 40: Se ha terminado de crear la página

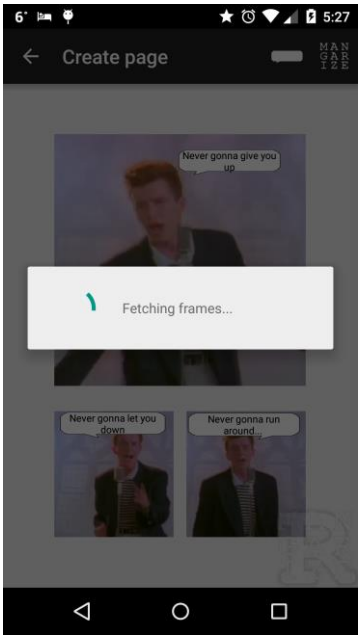

Figura 41: Se extraen los fotogramas desde el archivo de vídeo

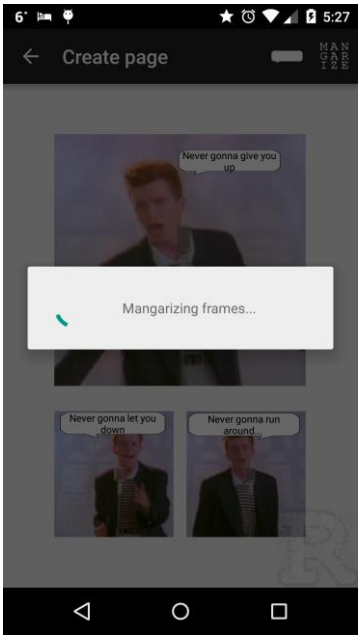

Figura 42: Se aplican los efectos a las imágenes

Figura 43: Resultado mostrado en la galería

A partir de un archivo de vídeo[15] se obtiene la imagen de 1080 x 1533 píxeles que se muestra a continuación:

Figura 44: Imagen resultante

[15] https://www.youtube.com/watch?v=dQw4w9WgXcQ [Consulta: 1 de diciembre de 2015]

5.4. Crear manga (con fotos como archivos de origen)

Ejemplo análogo al anterior, pero omitiendo la pantalla de selección de fotograma.

Figura 45: Fotografía inicial

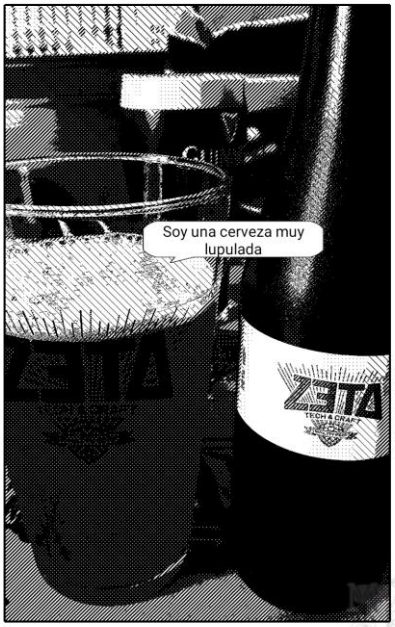

Figura 46: Imagen resultante

6. Conclusiones

6.1. Comentarios sobre el desarrollo

En un primer momento el proyecto comenzó teniendo un objetivo completamente distinto. El objetivo inicial era desarrollar una aplicación de realidad aumentada, que mostrara por pantalla en tiempo real y con efectos manga las imágenes obtenidas con la cámara del dispositivo. Esta idea se desechó rápidamente, ya que el tiempo de ejecución de los prototipos escritos en Python y ejecutados sobre un portátil equipado con un procesador i5 era demasiado dilatado. Lo que terminó de sentenciar la imposibilidad del proyecto fue la aplicación de OpenCV para Android[16]. En una de las demostraciones que incluía dicha aplicación, se aplicaba el algoritmo de Canny a la parte central de la imagen recibida por la cámara, y la tasa de fotogramas por segundo no resultaba satisfactoria para una aplicación en tiempo real. Si con una aplicación oficial de los creadores de la biblioteca no se conseguía un rendimiento satisfactorio, difícilmente iba a conseguirlo yo en lo que iba a ser mi primera aplicación para Android.

Como el asunto de mangarizar imágenes me seguía resultando atractivo, decidí eliminar la característica del tiempo real y la idea acabó evolucionando a la aplicación que se ha presentado en este trabajo.

Aun así, encontré otras dificultades. Al empezar a hacer pruebas en Android (tras realizar la mayor parte de un curso de Android en la plataforma Udacity, recordemos que no sabía nada en absoluto sobre desarrollo en esta plataforma) me encontré con el primer gran escollo: el soporte para Android anunciado en la página de la biblioteca OpenCV no era total, y en su versión 2.4 no soportaba el manejo de vídeos en Android (en teoría, la versión actual ya lo soporta). Tras algo de búsqueda di con JavaCV, una interfaz que ofrece funciones de bibliotecas como OpenCV o FFmpeg. Finalmente sería usando FFmpeg a través de JavaCV como sería capaz de abrir los vídeos de manera más o menos adecuada. Esto no fue fácil porque, una vez más, en ese momento no había documentación al respecto (en la propia se reconocía y se recomendaba echar un vistazo a un par de códigos de demostración).

Pese a todo, el proyecto fue progresando y, aunque me gustaría que tuviera muchas mejoras y ampliaciones, estoy razonablemente satisfecho con el resultado.

6.2. Trabajo futuro

- El tiempo de mangarizado no es nada despreciable, quizá se podría mejorar mediante el uso de shaders.
- Permitir mayor control del usuario en el proceso de mangarizado. El usuario podría elegir los parámetros de creación de las tramas, la cantidad de las mismas que se usará o los umbrales del algoritmo Canny.
- Permitir seleccionar qué área de la foto o fotograma origen se debe incluir en la viñeta. Actualmente se centra y se recorta automáticamente.
- Permitir aumentar el tamaño de los bocadillos y mejorar el ajuste del texto en su interior.
- Permitir elegir parámetros del texto, como pueden ser color, tamaño o fuente.

[16] https://play.google.com/store/apps/details?id=org.opencv.engine&hl=en [Consulta: 1 de diciembre de 2015]

- Aumentar la resolución de la imagen resultante de crear manga. Que no dependa de la resolución de la pantalla del dispositivo.
- Asegurarse del correcto funcionamiento de la aplicación en multitud de dispositivos.
- Añadir una mayor cantidad de distribuciones de viñetas.
- Permitir que las distribuciones de viñetas vayan definidas en un archivo complementario, un xml por ejemplo, en lugar de estar introducidas en el código.
- La aplicación contiene poco texto pero habría que asegurarse de que todo el que aparece durante la ejecución proviene del archivo xml de cadenas y traducir dicho archivo a distintos idiomas, para que la aplicación se ejecutara en el idioma del dispositivo.

Pese a que se puede introducir gran cantidad de mejoras o extensiones de las características iniciales, es conveniente no perder de vista que se trata de una aplicación móvil y no de un entorno de retoque fotográfico para escritorio. La inclusión de demasiadas características puede hacer perder usabilidad a la aplicación, llegando a abrumar al usuario.

Dedicatoria y agradecimientos

Dedico este trabajo, resultado de tantos años de estudio, a mis padres. Porque siempre han estado ahí y siempre he recibido todo su amor y su apoyo incondicional. No está a vuestra altura, pero este trabajo es para vosotros: gracias.

También quiero dedicarlo a mis amigos, a los que llevan más de veinte años conmigo y a los que llevan menos; a los que son de fuera, a los que se marcharon de Valencia y a los que siguen aquí; gracias a vosotros el mundo es un poco mejor.

No puedo dar por concluida esta memoria sin mostrar mi agradecimiento a Tatsuya Harada, de la Universidad de Tokio, y al resto de miembros de su laboratorio, donde me acogieron por un semestre; a Paco Abad, que aceptó tutorizar este proyecto; y a Francisco José Soto Portillo e Ignacio López Sais por la ayuda prestada.

Bibliografía

[1] TOKYO OTAKU MODE INC. *Otaku Camera. Mangatize yor photos!* <http://otakucamera.com/> [Consulta: 1 de diciembre de 2015]

[2] SUPERSOFTWARE CO. LTD. *漫画カメラ* <http://tokyo.supersoftware.co.jp/mangacamera/> [Consulta: 1 de diciembre de 2015]

[3] GRAPHICALIZE. *Comicize – the comics maker*. <http://www.graphicalize.net/index.php/8-recent/6-releasing-comicize-the-comics-maker> [Consulta: 1 de diciembre de 2015]

[4] CANNY, J. F. (1986). "A computational approach to edge detection" en *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 679-698.

[5] OPENCV. *Canny Edge Detector*. <http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html> [Consulta: 1 de diciembre de 2015]

[6] SCIKIT-LEARN. *Mini Batch K-Means*. <http://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans> [Consulta: 1 de diciembre de 2015]