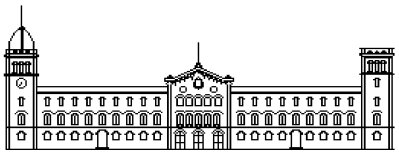


Trabajo final de carrera

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Facultad de Matemáticas
Universidad de Barcelona

Reconocimiento Automático de Formas

Andrés Juan Sanz

Director: Oriol Pujol Vila
Realizado en: Departamento de Matemática Aplicada y Análisis. UB
Barcelona, 20 de septiembre de 2013

Resumen

En muy pocos años, los teléfonos móviles han evolucionado a un ritmo vertiginoso pasando de ser dispositivos para comunicarse hasta convertirse en pequeños ordenadores (SmartPhones).

Android y iOs son unos de los sistemas operativos más conocidos para terminales móviles. En este proyecto, nos hemos centrado en la plataforma Android, la cual es Open Source y utiliza el lenguaje de programación Java.

Este proyecto introduce los conceptos de captura, procesado y reconocimiento de imágenes. El objetivo del mismo es realizar un software capaz de reconocer automáticamente una imagen, o parte de ella, y mostrar la imagen más parecida a ella comparándola en una base de datos local del terminal.

Para poder realizar esto, el proyecto se ha dividido en dos grandes puntos.

El primero se realizó con la herramienta de software matemático MATLAB.
A partir de una imagen proporcionada por el usuario desde su galería de imágenes, el programa muestra las cinco imágenes más similares extrayendo y comparando de una base de datos local del disco duro del ordenador.
Esto se realizó mediante el algoritmo de detección de bordes “Canny” y la función “Bwdist”, que calcula la distancia de todo píxel de la imagen al píxel de color blanco más cercano. Estos algoritmos y la manera de funcionar del programa los explicaremos más adelante.

El segundo punto ha sido realizar una aplicación Android con el mismo objetivo que el primer punto. Únicamente, la base de datos, en este caso, reside en la tarjeta SD del dispositivo y la imagen proporcionada por el usuario puede venir de la galería de imágenes del teléfono o de la cámara de fotos del mismo terminal.
Esta aplicación utiliza cálculos realizados en Matlab y almacenados en un fichero de extensión txt que, a su vez, está guardado en la memoria SD del teléfono.

Esta aplicación podrá ser un punto de partida para futuras ampliaciones de la misma como por ejemplo: ser una aplicación cliente-servidor, poder ampliar la base de datos de comparación subiendo fotos en tiempo real, permitir la conexión a internet, ubicación GPS de la imagen origen, etc.

Resum

En pocs anys, els telèfons mòbils han evolucionat a un ritme vertiginós passant de ser dispositius per comunicar-se fins a convertir-se en petits ordinadors (smartphones).

Android y iOs son uns dels sistemes operatius més coneguts per terminals mòbils. En aquest projecte, ens hem centrat en la plataforma Android, la qual es Open Source i utilitza el llenguatge de programació Java.

Aquest projecte introdueix els conceptes de captura, processament i reconeixement d'imatges. L'objectiu del mateix es realitzar un software capaç de reconèixer automàticament una imatge, o una part d'ella, i mostrar la imatge més semblant a ella comparant-la amb una base de dades local del terminal.

Per poder realitzar això, el projecte s'ha dividit en dos grans punts. El primer es va realitzar amb l'eina de software matemàtic MATLAB. Amb ell es va crear un programa per mostrar les cinc imatges mes semblants, extretes d'una base de dades del disc dur de l'ordinador, a partir d'una imatge proporcionada per l'usuari des de la seva galeria d'imatges.
Això es va poder aconseguir mitjançant l'algoritme de detecció de vores "Canny" i la funció "Bwdist", que calcula la distancia de tot píxel de la imatge al píxel de color blanc més proper. Aquests algoritmes i la manera de funcionar del programa els explicarem més endavant.

El segon punt ha sigut realitzar una aplicació Android amb el mateix objectiu que el primer punt. Únicament, la base de dades, en aquest cas, resideix en la targeta SD del dispositiu i, la imatge proporcionada per l'usuari pot vindre de la galeria d'imatges o de la càmera de fotos del mateix terminal.
Aquesta aplicació utilitza càlculs realitzats en la part de Matlab i emmagatzemats en un fitxer d'extensió txt que, al mateix cop, està guardat a la memòria SD del telèfon.

Aquesta aplicació podrà ser un punt de partida per futures ampliacions de la mateixa com per exemple: ser una aplicació client-servidor, poder ampliar la base de dades de comparació pujant fotos en temps real, permetre la connexió a Internet, ubicació de GPS de la imatge origen, etc.

Summary

In a very few years, the mobile phones have evolved at a vertiginous pace, starting as communication devices and turning into small computers (Smart Phones)

Android and iOs are ones of the most known operating systems for mobile terminals. In this project, we focused on Android platform, which is Open Source and uses the Java programming language.

This project introduces the concepts of capturing, processing and image recognition. The aim of the plan is to make an able software to automatically recognise an image, or part of it, and show the closest image to it by comparing the original in a local database of the terminal.

In order to make this possible, the project has been divided in two big points.
The first one, was performed using the MATLAB, mathematical software tool.
From an image stored on the user's image gallery, and through a program created with the tool MATLAB, the five more similar images to the original one, were extracted from a database local computer hard drive.
This process was done by the edge detection algorithm "Canny" and the "Bwdist", both of them calculate the distance of every pixel in the image to the nearest white pixel. At a later time we will be back to explain these algorithms and how the program runs.

The second point of the project was to perform an Android app with the same goal as the first point. The difference is in the database, in this case, lies in the mobile SD card, so the image provided by the user may come from the phone's picture gallery or the camera of the terminal.
This app uses calculations performed in Matlab and stored in a txt file, which in turn is stored in the phone's SD memory.

This application may be a starting point for future extensions of the app, such as: become a client-server application; be able to expand the comparison data base by uploading photos in real time; allow internet connection; GPS location of the source image; etc.

Agradecimientos

Quisiera agradecer a mis dos tutores del proyecto, Francesco Ciompi y Oriol Pujol por la paciencia que han tenido en mis numerosas complicaciones, retrasos y convocatorias que he retrasado esta entrega ya fuesen por problemas personales, laborales o conceptuales.

Igualmente agradecer a mi familia y amigos por apoyarme todos estos años, para algunos “interminables”, ya que sin ellos no estaría donde estoy hoy.

A los compañeros de universidad, por motivarnos unos a otros y por ser la base diaria de conocimientos, explicaciones, demostraciones y soluciones que, acertadas o no, ayudan en la formación académica y personal.

Índice

1. Introducción

1.1 Ámbito del proyecto

Este proyecto consiste en la implementación de un software que pueda reconocer una imagen (o parte de ella) facilitada por el usuario y, mediante los algoritmos de detección, procesado y reconocimiento de imágenes, mostrarnos otra imagen, almacenada en una base de datos interna, siendo ésta lo más parecida a la original.

1.2 Motivación

La motivación principal fue el poder diseñar e implementar una aplicación desde cero, con un sistema de desarrollo Android prácticamente desconocido por mí, únicamente el lenguaje de programación utilizado (Java), y utilizar unos sistemas de detección y reconocimiento de objetos e imágenes tanto en Matlab como en librerías Android.

Estamos en una época en la que casi cada persona posee un teléfono móvil o dispositivo electrónico al alcance de su mano y, como podemos observar en el día a día, las aplicaciones móviles cada vez están más extendidas en nuestras acciones cotidianas.
Poder crear una aplicación y poder llegar a tanta gente, más adelante veremos la cuota de mercado actual de Android, crea una motivación especial al proyecto.

Profundizar en el mundo de aplicaciones para móviles y profundizar en el mundo Android es, a su vez, adquirir buenos conocimientos de cara al mundo laboral y eso, hoy en día, es una motivación muy importante.

1.3 Objetivos generales

El objetivo general de este proyecto es, como se ha comentado anteriormente, la creación de un software, aplicación Android, que permita reconocer una imagen, o una parte de la misma recortada por el usuario, seleccionada desde la cámara o desde la galería del dispositivo y que nos muestre como resultado final, la imagen más parecida a la original encontrada en una base de datos de imágenes guardada en la memoria del dispositivo.

En cuanto a los lenguajes de programación, Java para programar la parte técnica de la aplicación y XML para programar la interfaz gráfica de la misma.

Para la elaboración de la memoria y diagramas se utilizó MS Office 2007, MS Visio 2007, Dia y Pacestar UML Diagrammer.

1.4 Objetivos específicos

Para conseguir el objetivo principal, el proyecto se subdivide en varios objetivos específicos y/o apartados:

a) Crear una base de datos de imágenes para poder realizar la comprobación con la enviada por el usuario a la aplicación

b) Generar programa de reconocimiento de imágenes en Matlab, utilizar algoritmos de detección de bordes, distancia euclidea, distancia de pixels, operaciones con imágenes, etc.

c) Generar archivo de texto con el cálculo del mapa de distancias de las imágenes de la base de datos para poder utilizarla en la ejecución de la aplicación Android. Esta operación se realiza en Matlab.

d) Diseñar e implementar aplicación para móvil, unificar todos los apartados y realizar entorno de pruebas

1.5 Estudio de mercado

En este apartado podríamos distinguir entre Matlab y Android.
Podemos encontrar por la red mucha información relacionada con el procesamiento digital de imágenes con Matlab. La mayoría corresponde a programas o tutoriales relacionados con el reconocimiento facial o con la detección de bordes de una imagen.
Parte de esta información nos ha servido para desarrollar nuestro software como más adelante veremos.

Para poder ver si existen aplicaciones parecidas en Android, debemos buscar en Google Play Store. Realmente no hemos encontrado aplicaciones que desarrollen el mismo objetivo que la nuestra, aunque muchas son parecidas o van más allá.
Existen juegos estilo memory, aplicaciones de reconocimiento de un objeto a partir de la imagen del mismo (seleccionando un patrón), aplicaciones de reconocimiento de texto e imágenes para ofrecer información sobre la misma, etc.
Podemos ver dos ejemplos a continuación:

- Demo Reconocimiento Patrones: El objetivo es probar un algoritmo de reconocimiento de patrones implementado en OpenCV, que pueda determinar si el patrón aprendido a partir de una imagen, se encuentra en una serie de imagenes capturadas a continuación. El algoritmo utiliza a su vez los algoritmos SURF (para el cálculo de descriptores), FLANN (para buscar la relación en los descriptores del patrón y del resto de imágenes), y al final calcula una homografía para obtener la proyección del patrón sobre la imagen (si es que lo encuentra).

Figura 1: Aplicación demo reconocimiento patrones

- Google Goggles: Busca haciendo una foto: apunta con la cámara de tu móvil a un cuadro, a un lugar famoso, a un código de barras o QR, a un producto o a una imagen popular. Si Goggles lo encuentra en su base de datos, te ofrecerá información útil.
  Goggles puede reconocer texto en francés, inglés, italiano, español, portugués, turco y ruso, y puede traducirlo a otros idiomas.
  Goggles también sive como escáner de códigos de barras y QR.

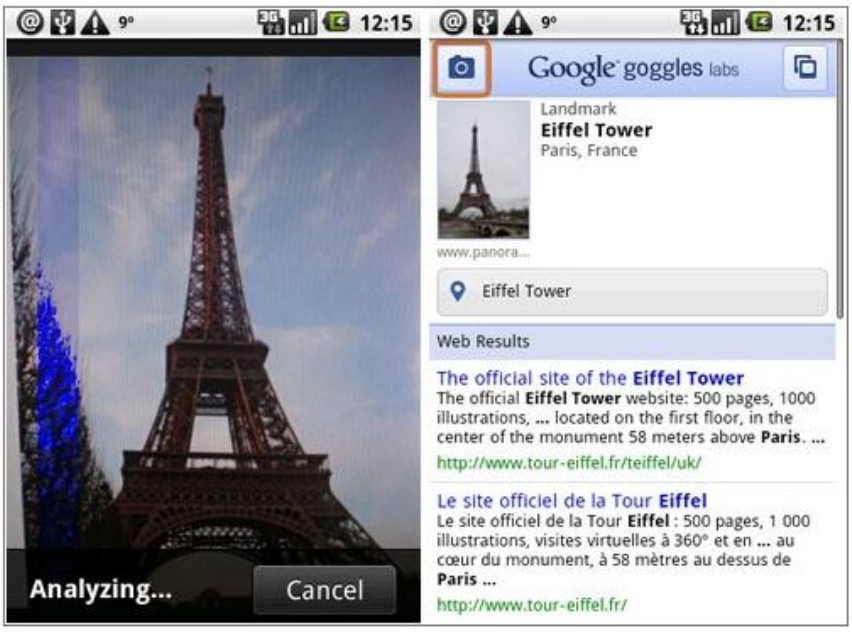


Figura 2: Aplicación Google Goggles

2. Tecnologías

2.1 Android

2.1.1 ¿Qué es Android?

Andoid es un sistema operativo basado en Linux, inicialmente pensado para terminales móviles, al igual que Blackberry OS o iOS (Apple).

Actualmente se utiliza en muchos otros dispositivos como tabletas, lectores de libros electrónicos o incluso televisores.
Su desarrollo busca crear una plataforma de software abierta que pueda ser usada para proporcionar al usuario la mejor experiencia con dispositivos móviles.

Desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005, es el principal producto de la Open Handset Alliance, un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio, entre los que destacan Google, T-Mobile, HTC, Qualcomm y Motorola entre otros.

Figura 3: Imagen Android

2.1.2 Breve historia de Android

En Octubre de 2003 es fundada Android Inc. en Palo Alto, California, Estados Unidos por Andy Rubin, Rich Miner, Nick Sears y Chris White.

Android Inc. desarrolló durante sus inicios Android, hasta que la compañía fue adquirida por Google en Julio de 2005. De este modo, el desarrollo continuó de la mano de Google y la Open Handset Alliance, un conglomerado de fabricantes y desarrolladores hardware, software y operadores de servicios (fundada el 5 de Noviembre de 2007).

Poco después de la fundación de la Open Handset Alliance, el 12 de Noviembre de 2007, se publicó la versión Beta de la SDK de Android.

El 23 de Septiembre del 2008 aparece el primer dispositivo con Android, el HTC Dream(G1) con Android 1.0 obteniendo un cuota de mercado del 0.50%.

Hasta Julio de 2013 han aparecido nada menos que siete versiones de Android, que por orden cronológico (y alfabético, teniendo en cuenta sus nombres referentes a postres) son: Android 1.0, Android 1.5 Cupcake, Android 1.6 Donut, Android 2.0 Eclair, Android 2.2 Froyo, Android 2.3 Gingerbread, Android 3.2 Honeycomb, Android 4.0 Ice Cream Sandwich y Android Jelly Bean en versiones 4.1 y 4.2 y 4.3.

En abril de 2013 se hizo público que Android alcanzó el 92% en ventas de nuevos *smartphones* para el trimestre comprendido entre diciembre 2012 y febrero 2013 en España, seguido de iOS con un 4.4%

En la actualidad existen aproximadamente 1.000.000 de aplicaciones para Android y se estima que 1.500.000 teléfonos móviles se activan diariamente, lo que hace que se llegue a los 1.000 millones de Smartphones Android en el mundo.

2.1.3 ¿Por qué desarrollar en Android?

Para empezar, el crecimiento que ha tenido este sistema operativo en tan poco tiempo ha sido espectacular, ocupando actualmente casi el 80% de la cuota de mercado. Esto hace

Top Smartphone Operating Systems, Shipments, and Market Share, 2013 Q3 (Units in Millions)

Operating System	2Q13 Unit Shipments	2Q13 Market Share	2Q12 Unit Shipments	2Q12 Market Share	Year-over-Year Change
Android	187.4	79.3%	108	69.1%	73.5%
iOS	31.2	13.2%	26	16.6%	20.0%
Windows Phone	8.7	3.7%	4.9	3.1%	77.6%
BlackBerry OS	6.8	2.9%	7.7	4.9%	-11.7%
Linux	1.8	0.8%	2.8	1.8%	-35.7%
Symbian	0.5	0.2%	6.5	4.2%	-92.3%
Others	N/A	0.0%	0.3	0.2%	-100.0%
Total	**236.4**	**100.0%**	**156.2**	**100.0%**	**51.3%**

Source: IDC Worldwide Mobile Phone Tracker, August 7, 2013

Figura 4: Cuota mercado sistemas operativos para móvil

Seguidamente ofrece una capacidad muy grande para desarrollar las interficies gráficas fácilmente y con una calidad suficientemente alta (ya sea vía código o vía XML).

Además, el lenguaje de programación utilizado es Java, integrando XML en la parte visual o gráfica como hemos comentado.
Es un lenguaje que en la carrera hemos utilizado comúnmente y además es un lenguaje fácil de programar, entender y aprender debido a que está muy extendido.

Esto hace que existan millones de desarrolladores y que se dispongan de múltiples recursos de aprendizaje como comunidades oficiales, foros de debate, manuales, cursos, etc.

Es un lenguaje multiplataforma por lo que nos permite programar en Windows, Linux o MAC simplemente descargando el Android-SDK, con todas las librerías Java que utiliza Android. Actualmente existen paquetes software, como “Eclipse Android Developer Tools”, que integran todo lo necesario para programar en Java-Android.

2.1.4 Funcionamiento de Android

El Sistema Operativo Android está estructurado mediante capas. Cada una de estas capas utiliza elementos de la capa inferior para realizar sus funciones. Por ese motivo, a este tipo de arquitectura se le denomina pila. La pila de software de Android es la siguiente:

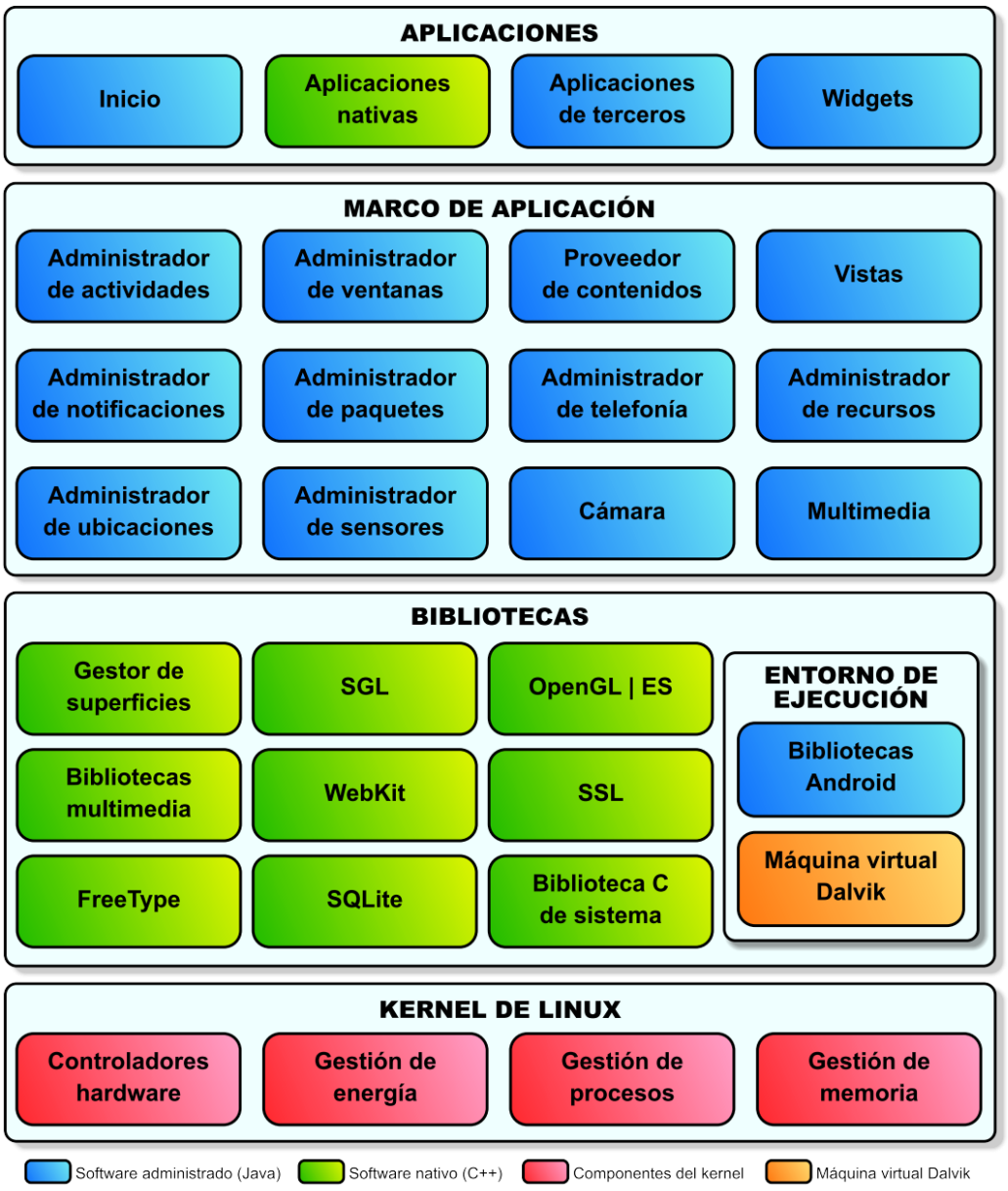


Figura 5: arquitectura Android

1- Núcleo de Linux (Linux Kernel).
Versión del kernel de Linux adaptada a las características y requerimientos del hardware en el que va a ser ejecutado (normalmente un smartphone). Proporciona controladores hardware y se encarga de gestionar los diferentes recursos del teléfono (memoria, energía,…) y del sistema operativo en sí: procesos, elementos de comunicación, etc.

2- Librerías (Libraries)
Ofrecen los servicios de código y datos a las capas superiores. Están escritas en C y C++.
Su cometido es proporcionar funcionalidad a las aplicaciones, para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma más eficiente.
Cada librería nos proporciona unas capacidades o servicios diferentes. Por ejemplo, OpenGL y ES son librerías 3D, Surface Manager nos da la capacidad para imprimir en la pantalla y ventanas, Media Framework almacena los codecs necesarios para la reproducción de videos y música, FreeType es para las fuentes, etc.

3- Android Runtime
Es la capa encargada de gestionar las peticiones de las aplicaciones. El entorno de ejecución también está formado por bibliotecas, de ahí que, a veces, no es considerado como una capa en sí.
El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik, componente que ejecuta todas y cada una de las aplicaciones no nativas de Android. Las aplicaciones se codifican normalmente en Java y son compiladas, pero no para generar un ejecutable binario compatible con la arquitectura hardware específica del dispositivo Android. En lugar de eso, se compilan en un formato específico para la máquina virtual Dalvik, que es la que las ejecuta. Esto permite compilar una única vez las aplicaciones y distribuirlas ya compiladas teniendo la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera cada aplicación.
Las aplicaciones Android se ejecutan cada una en su propia instancia de la máquina virtual Dalvik, evitando así interferencias entre ellas, y tienen acceso a todas las bibliotecas mencionadas antes y, a través de ellas, al hardware y al resto de recursos gestionados por el kernel.

4- Armazón de Aplicaciones (Aplication Framework)
Esta capa la forman todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones y que, obviamente, se apoyan en las bibliotecas y en el entorno de ejecución (Android runtime) que ya hemos detallado. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos a través de la máquina virtual Dalvik.

5- Aplicaciones (Aplications)
   Esta capa, como su nombre indica, la forman las aplicaciones. En este saco se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, tanto las nativas (programadas en C o C++) como las administradas (programadas en Java), tanto las que vienen de serie con el dispositivo como las instaladas por el usuario.
   Aquí está también la aplicación principal del sistema: **Inicio** (*Home*), también llamada a veces lanzador (*launcher*), porque es la que permite ejecutar otras aplicaciones proporcionando la lista de aplicaciones instaladas y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso pequeñas aplicaciones incrustadas o *widgets*, que son también aplicaciones de esta capa.

2.1.5 Componentes principales

Activities - por analogía podemos pensar que es como una ventana de Windows, con la diferencia que en Windows puedes tener varias ventanas a la vez, cuando en Android sólo puedes tener una al frente. Otra analogía que podríamos pensar es que es un proceso de un SO, con la diferencia que tiene una componente gráfica. Pueden estar activas(al frente) o en segundo plano (aunque nos arriesgamos que el SO la “mate”).

Content Providers - es como tener una especie de interfaz donde proveer los datos. Permite modificar o cancelar datos, pero de una forma controlada por el proveedor. Por ejemplo, podemos pensar tener un Content Providers de nuestros contactos del teléfono, no sería muy bueno que alguien entre y modifique lo que quiera sin ninguna consistencia.

Services - análogo a un servicio de Windows o un daemon de Unix, algo que está ejecutándose por atrás.

Intents - es cuando un componente llama a otro, como un servicio de mensajería. La versión más fácil es cuando una Activity llama a otra. Hemos de tener en cuenta que están controlados por el sistema, por lo que no es una llamada directa. En cierto modo podemos verlo como un intento de hacer algo, ya que el SO es quien finalmente se encarga de todo.

2.1.6 Estrucutra aplicación Android

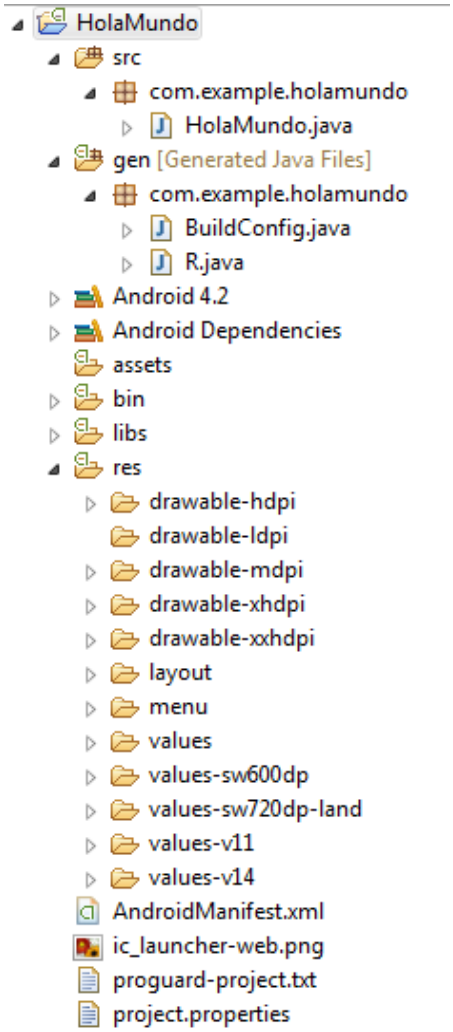


Figura 6: Estructura aplicación Android

1. assets - Este directorio es un repositorio de archivos, es decir, un lugar donde se coloca cualquier tipo de fichero externo que sea necesario en cualquier momento por la aplicación.
2. bin - Aquí irán compilados los binarios de nuestra aplicación.
3. res - Directorio donde van alojados los resources (recursos) tales como las imágenes. Este directorio está formado por 3 Subdirectorios: Drawable (aquí van las pngs de la aplicación), Layout (archivos xml encargados de la confección de la interface) y Values (archivos encargados de idiomas, colores, variables, etc.).
4. gen - Contiene un fichero java (R.java) el cual guarda identificadores de los recursos de la interfaz gráfica.
5. src - En este directorio va toda la programación escrita en Java para el funcionamiento de la aplicación.
6. .project,.classpath - Este directorio es el creado por Eclipse para su correcto funcionamiento y/o compilación del proyecto.
7. AndroidManifest.xml → Este archivo es imprescindible en cada aplicación Android. Es el encargado de gestionar los componentes, servicios, datos y clases de cada paquete de la aplicación.

2.1.7 Programación en Android Java

Para poder realizar nuestra aplicación, como hemos visto, hemos utilizado el lenguaje de programación Java y el entorno de desarrollo Eclipse.

Java es un lenguaje de programación orientado a objetos, desarrollado por SUN Microsystems a principios de los años 90. Realmente dispone de una sintaxis parecida a C y C++ pero intenta eliminar herramientas de bajo nivel.
Actualmente Java pertenece a Oracle después de que ésta comprase a Sun Microsystem por 7.400 millones de dólares.

2.2 Eclipse

Como hemos dicho, Eclipse es un entorno de desarrollo que fue creado originalmente por IBM.
Actualmente es desarrollado por una fundación sin ánimo de lucro, Fundación Eclipse, que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse permite desarrollar proyectos un múltiples lenguajes de programación como C, C++, Java, COBOL, Python, etc., para ello únicamente hay que instalar las librerías necesarias para cada lenguaje.

Para nuestro proyecto, hemos utilizado una herramienta Android Development Tools (ADT), que es un plugin de desarrollo para el IDE de Eclipse.
Extiende las capacidades de Eclipse para permitirte configurar rápidamente nuevos proyectos de Android, crear una interfaz de usuario de la aplicación, añadir componentes en función de la API de Android Framework, depurar tus aplicaciones mediante las herramientas de Android SDK e incluso exportar APKs con (o sin) firma con el fin de distribuir tu aplicación.

En general, el uso de Eclipse con ADT es un enfoque altamente recomendado para el desarrollo de Android y es la forma más rápida para empezar.

Figura 7: Logo presentación Andoid Developer Tools

2.3 Matlab

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas de Unix, Windows y Apple. Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes. Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes).

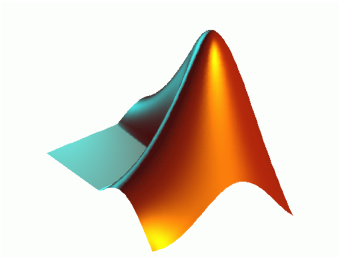


Figura 8: Logo Matlab

2.4 Hardware

Se han utilizado dos dispositivos como emuladores para realizar las pruebas de la aplicación. Estos han sido un Smartphone modelo HTC Desire con una versión de Android 2.2 y una tablet ARCHOS 101 XS con una versión de Android 4.1

Figura 9: Tablet ARCHOS 101 XS

Figura 10: HTC Desire

3. Prototipado de imágenes con MATLAB

La primera parte del proyecto consiste en abordar el objetivo de reconocimiento de imágenes desde MATLAB.

Dicho objetivo consiste en crear una interfaz gráfica, conocida también como GUI (graphical user interface), donde, el usuario seleccione una imagen del disco duro del ordenador, recorte, o no, la imagen para seleccionar la parte deseada a procesar y, finalmente, el sistema muestre las tres imágenes más parecidas a la original seleccionadas de una base de datos local almacenada en el propio disco duro del ordenador.
Como hemos comentado MATLAB nos permite la manipulación de imágenes a partir de la implementación de algoritmos y de la manipulación de matrices.

3.1 Esquema aplicación

A continuación podemos ver el esquema (ver figura 11) de la aplicación o pipeline. Mostramos los pasos más importantes y principales que ejecuta la aplicación
En la parte inferior, o derecha, de cada objeto, indicamos las acciones vitales (ya sean funciones Matlab, algoritmos o interacciones con el usuario) que se realizan en ese paso del esquema.

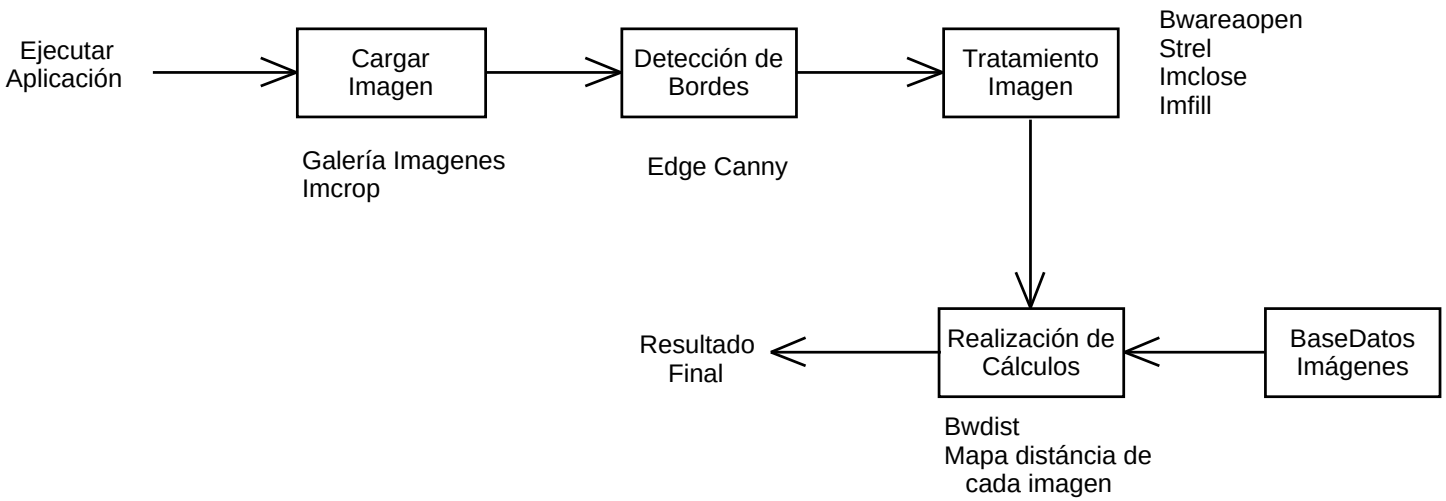


Figura 11: Esquema prototipado

En los siguientes puntos explicaremos cada uno de los pasos del esquema de prototipado, así como los algoritmos más importantes utilizados, tanto en su teoría como en sus funciones en la aplicación.

3.2 Cargar Imagen

La carga de la imagen a tratar es una interacción entre el usuario y el sistema. Éste muestra por pantalla una ventana donde el usuario, navegando por las carpetas que desee, selecciona la imagen a tratar. Una vez seleccionada, la imagen es mostrada por pantalla.

Matlab ofrece una función llamada **'imcrop'** la cual se encarga de recortar un rectángulo de la imagen seleccionada. Esto nos será útil cuando únicamente queramos procesar una parte de la imagen.
La función presenta la estructura siguiente:

img_recortada = imcrop(im1,area)

Donde 'im1' es la imagen a recortar y área es el siguiente vector, area = [Xmin, Ymin, ancho, alto]. Esta función también ofrece la opción de dibujar el rectángulo directamente en la imagen y el programa mismo se encarga de fijar las posiciones de area y cortar la imagen.
En nuestro caso, area la seleccionaremos dibujando directamente el rectángulo con el ratón encima de la imagen.

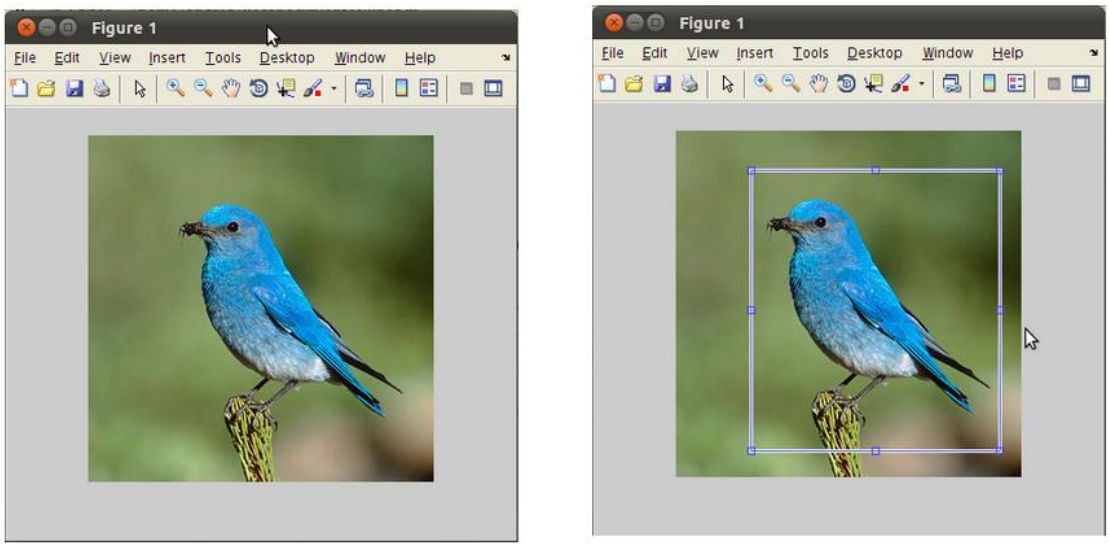


Figura 12: Funcionamiento de imcrop

Una vez obtenida la imagen que deseamos procesamos, redefinimos su tamaño y la convertimos en escala de grises. De este modo nos aseguramos que todas las imágenes sean iguales en tamaño y no haya problemas en el momento de realizar el procesado.
El hecho de convertirla en escala de grises es necesario para después poder aplicarle el algoritmo de detección de bordes como veremos a continuación.

3.3 Detección de bordes

La parte más importante del programa consiste en la detección del borde de la imagen, tanto la imagen que queremos procesar como las imágenes de la base de datos.
La detección de bordes es la forma más utilizada en segmentación para encontrar discontinuidades en valores de intensidad, esto, en teoría, permite delimitar su tamaño y su región.

La IPT (Image Processing Toolbox) de Matlab dispone de la función **edge**, desde la que se puede detectar los bordes de una imagen utilizando diferentes métodos.
La función presenta la estructura siguiente:

img_contorno = edge(im_res)

La función recibe como entrada una imagen binaria o en escala de grises 'im_res' y devuelve una imagen binaria 'img_contorno' del mismo tamaño que 'im_res', con 1's donde la función haya encontrado bordes y 0's en el resto.

El algoritmo edge dispone de varios métodos para detectar los bordes. En nuestro programa, después de realizar pruebas con los diferentes métodos disponibles, nos decantamos por **Edge Canny** ya que era el que más prestaciones nos ofrecia ya que detecta bordes débiles y no se ve tan afectado por el ruido que pueda aparecer en la imagen.

Igualmente, vamos a enumerar los diferentes algoritmos probados en el programa:

- El método Sobel (método por defecto si no se especifica ninguno en la función), detecta los bordes usando la aproximación Sobel. El operador Sobel calcula el gradiente de la intensidad de una imagen en cada punto (píxel). Así, para cada punto, este operador da la magnitud del mayor cambio posible, la dirección de éste y el sentido desde oscuro a claro. El resultado muestra qué tan abruptamente o suavemente cambia una imagen en cada punto analizado y, en consecuencia, que tan probable es que éste represente un borde en la imagen y, también, la orientación a la que tiende ese borde.
- El método Prewitt detecta los bordes usando la aproximación Prewitt. Suele obtener resultados parecidos a los obtenidos con el método Sobel.
- El método del Laplaciano Gaussiano ('log') detecta los bordes buscando cortes con cero después de filtrar 'img_contorno' con un filtro Laplaciano Gaussiano. Obtiene, normalmente, mejores resultados que el método Sobel pero a costa de tener que modificar parámetros por defecto

Finalmente, el método Canny, detecta los bordes buscando máximos locales del gradiente de 'img_contorno'. Este método es el más potente de todos y por ello el más utilizado. Es menos propenso a confundirse por el ruido y más propenso a detectar bordes débiles.

El algoritmo de detección de bordes de Canny utiliza un filtro basado en la primera derivada de una gaussiana. Ya que es susceptible al ruido presente en datos de imagen sin procesar,la imagen original es transformada con un filtro gaussiano.

El resultado es una imagen un poco borrosa respecto a la versión original. Esta nueva imagen no se ve afectada por un píxel único de ruido en un grado significativo.

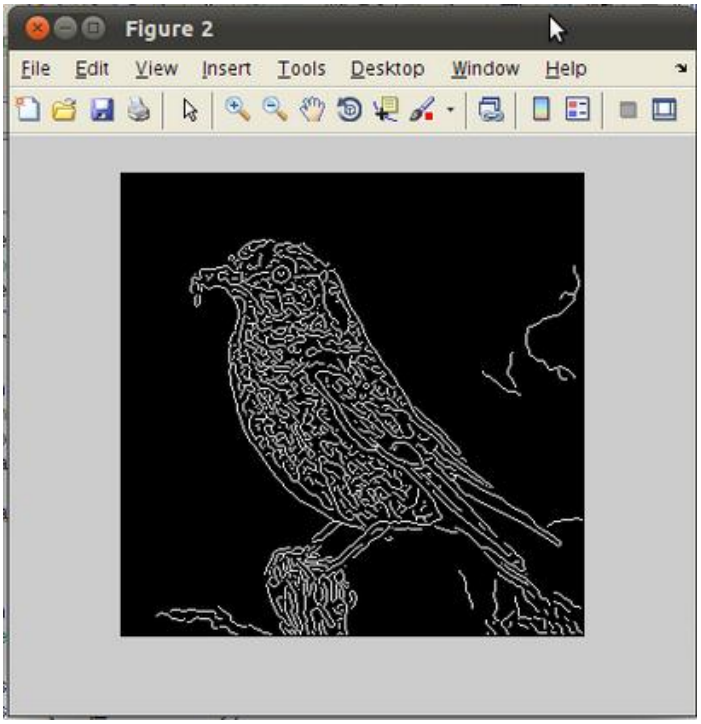


Figura 13: Algoritmo Edge Canny

3.4 Tratamiento imagen

Las siguientes funciones que utilizamos en el programa nos permiten definir y tratar más concretamente el borde de la imagen, estas són:

· **bwareaopen**:

BW2 = bwareaopen(BW, P)

Elimina de una imagen binaria todos los objetos concatenados que tengan menos de P píxeles, guardando su resultado, otra imagen binaria, en BW2.
Para ver un ejemplo, una vecindad de 4 píxeles, un par de píxeles colindantes forman parte del mismo objeto solo si están activos y conectados en vertical u horizontal

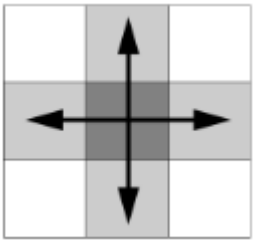


Figura 14: Vecindad de 4 píxeles

En nuestro programa hemos establecido una vecindad de 15 píxeles para eliminar píxeles sueltos de las imágenes exteriores.

· **strel:**

SE=strel(forma,radio)

Crea un elemento estructurante de una forma determinada y un radio de x píxeles. Hay diferentes tipos de formas que se diferencian en la manera que distribuyen los valores de los píxeles en la matriz de 1s y 0s. Su propio nombre indica qué forma adoptan, como por ejemplo, ‘diamond’, ‘disk’, ‘arbitrary’, ‘line’.
En el reconocimiento de formas de nuestro proyect hemos seleccionado una forma de disco (‘disk’) de radio 8 píxeles.

· **imclose**

Im2 = imclose(im,se)

Cierra morfológicamente la imagen binaria o en escala de grises IM, devolviendo la imagen procesada IM2. El elemento estructurante SE debe ser individual, no un array de objetos. La operación morfológica de cierre o clausura de una imagen consiste una dilatación seguida por una erosión usando el mismo elemento estructurante en ambas operaciones.

· **imfill**

bw2 = imfill(bw1,locations)

Se utiliza para evitar huecos en negro dentro de los principales y así no puedan ser interpretados como otros objetos. Imfill rellena estos huecos de los objetos principales.

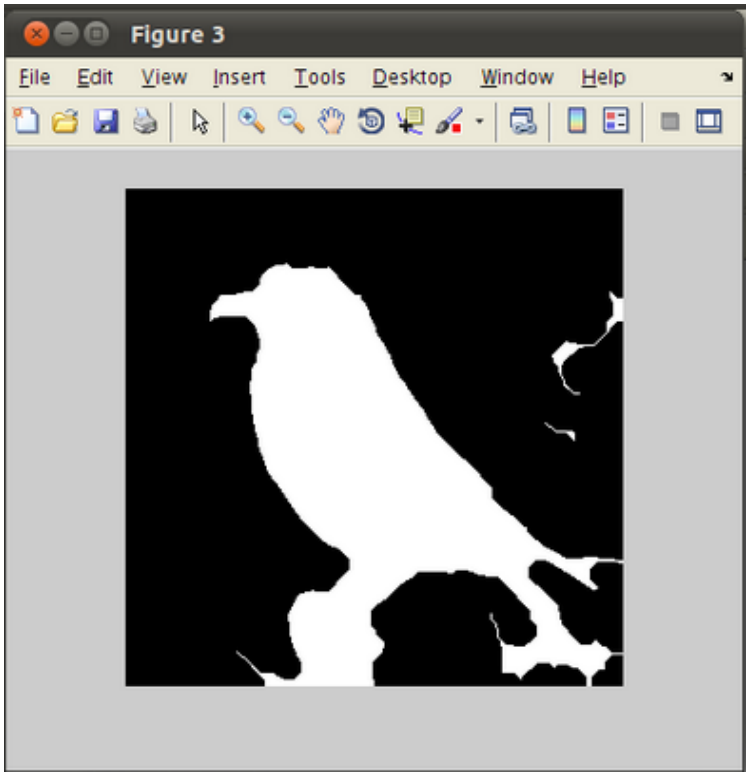


Figura 15: Resultado al aplicar las funciones de limpieza de imágenes

Como último paso a realizar con la imagen a reconocer proporcionada por el usuario, reescalamos a unos valores por defecto por si se han producido alteraciones de tamaño y creamos una variable donde almacenaremos el sumatorio de todos los píxeles de la imagen.

3.5 Realización de cálculos

Una vez finalizado el tratamiento de la imagen cargada por el usuario en tiempo real, el sistema realiza los cálculos con las imágenes de la base de datos para encontrar las tres más similares.

Como veremos más adelante, en propuestas de mejora, hay una parte de este apartado que ya podríamos tener calculado y así el proceso total sería mucho más corto de tiempo.

Para realizar estos cálculos, disponemos de un bucle finito de 0 a n, siendo n el número de imágenes que haya en la base de datos, dónde se recorre una a una cada imagen y se calcula el contorno de la misma manera que se ha realizado en la imagen original.

Dentro del mismo bucle, realizamos el cálculo de la distancia euclidiana de cada imagen de la base de datos con la función **bwdist** (transformada de distancia, medida de separación entre píxeles).
La transformada de distancia se puede interpretar cómo la distancia de cada pixel al pixel más distinto de cero. La función presenta la estructura siguiente:

D = bwdist(BW,metodo)

Calcula la transformada de distancia de la imagen binaria BW especificando una métrica alternativa en la variable método. A cada pixel de la imagen, se le asigna un número que es la distancia entre ese pixel y el pixel distinto de cero más cercano. Como hemos comentado, bwdist utiliza por defecto la métrica euclídea. El resultado es del mismo tamaño que BW.

El método métrico seleccionado en nuestro caso es ‘euclidean’, en el cual la distancia euclidea es la distancia real en línea recta entre dos píxeles.

0	0	0	1.41	1	1.41
0	1	0	1	0	1
0	0	0	1.41	1	1.41

Figura 16: Métrica Euclídea

Finalmente, como último cálculo, obtenemos el valor único de distancia de cada imagen de la base de datos respecto a la imagen cargada por el usuario cómo el sumatorio de cada pixel de la imagen original, multiplicado por cada pixel de la forma euclídea que hemos calculado con la función bwdist a la imagen de la base de datos. Todo ello dividió entre el sumatorio de los píxeles de la imagen elegida por el usuario que hemos calculado antes.

Gráficamente sería:

$$\sum(imagen\_usuario(:).*\ forma\_uclidea\_bbdd(:)) \div \sum imagen\_usuario(:)$$

Figura 17: Fórmula cálculo

Notación: el término '.' se refiere a realizar el producto elemento a elemento
*Notación: el término '(:)' indica el contenido total de una matriz

Una vez realizados todos los cálculos para cada una de las imágenes de la base de datos, recordemos que estamos dentro de un bucle finito de elementos, el programa ordena de menor a mayor el valor único de distancia de cada imagen.

Con esta ordenación, y realizando un bucle de tantos elementos como imágenes queramos mostrar, obtenemos y mostramos las imágenes más parecidas a la original.

Según la teoría que hemos visto de la métrica euclídea, cuanto menor sea la distancia, querrá decir que los píxeles son más parecidos.

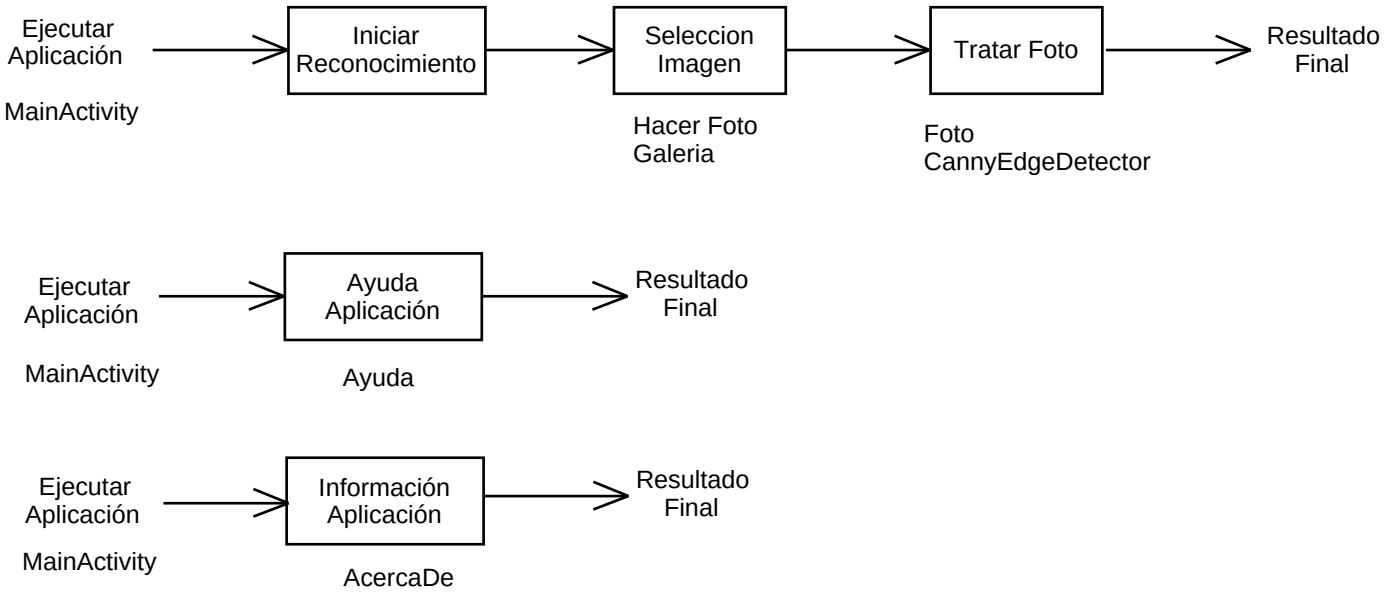
En el punto de la memoria 'Pruebas', podremos ver las pruebas realizadas en la aplicación Matlab y los resultados obtenidos.

4. Reconocimiento de imágenes con Android

La segunda parte del proyecto consiste en crear una aplicación Android capaz de reconocer imágenes, obtenidas por el usuario, y finalmente mostrar la más parecida a ella a partir de una comparación en sus píxeles.

4.1 Esquema aplicación

Como en el apartado de prototipado, a continuación podemos ver el esquema (ver figura 18) de la aplicación o pipeline.



Los pasos que realiza el esquema, y las respectivas clases y métodos que los forman, están explicados en los siguientes puntos, tanto en los casos de uso como en los diagramas de clase y secuencia.

4.2 Requerimientos

En este apartado realizamos el estudio del funcionamiento de la aplicación para poder encontrar cuál es el problema o necesidad final, pronosticar posibles errores, fallos del sistema y, sobretodo, para conseguir un buen resultado final que cumpla con los objetivos que hemos expuesto en anteriores apartados.

4.2.1 Aplicación Android

La aplicación Android ha de contener las siguientes características:

- Interfaz visual, sencilla y agradable, que facilite el maneja de la aplicación a cualquier tipo de usuario, tenga o no conocimientos de manejo de móviles
- Pedir al usuario de donde quiere seleccionar la fotografía (cámara/galeria)
- Dependiendo de la selección, abrir cámara de fotos del dispositivo o abrir galería de fotos y mostrar en pantalla la imagen
- Realizar el recorte de la imagen
- Mostrar pantalla final con imagen original e imágenes reconocidas
- Disponer de una ayuda
- Disponer de información de la aplicación

4.2.2 Requisitos funcionales

Son aquellos que describen el funcionamiento del sistema, es decir, los servicios que el sistema debe proporcionar, cómo debe reaccionar a una entrada particular del usuario en la aplicación y cómo se debe comportar ante situaciones particulares de la aplicación.

Los requerimientos de comportamiento para cada requerimiento funcional se muestran en los casos de uso, que veremos en el punto 4.2.

4.2.3 Requisitos no funcionales

Los requisitos no funcionales, son todos aquellos que no definen la funcionalidad de la aplicación, pero que son necesarios para su correcto funcionamiento. Los requisitos no funcionales a tener en cuenta durante el desarrollo de la aplicación de reconocimiento de formas son:

- Se debe tener espacio suficiente en la tarjeta SD para almacenar la base de datos
- Se garantiza la compatibilidad de la aplicación en cualquier dispositivo con el sistema operativo Android
- Se garantiza la no duplicidad de información en la base de datos para garantizar su correcto funcionamiento. Todas las imágenes son únicas y el usuario no puede modificar (en esta versión) la base de datos.
- La interfície gráfica tendrá que ser sencilla y clara, manteniendo el mismo tipo de estilo en todas las pantallas de la aplicación
- Debemos asegurar la seguridad y la fiabilidad en la aplicación. Por lo tanto, ésta no puede presentar inconsistencias, tendremos que garantizar, siempre que sea posible, que todos los datos que el usuario introduzca en el sistema para tratar el reconocimiento sean imágenes.

4.3 Base de datos y cálculos externos

Hemos utilizado parte del prototipado de Matlab para realizar la aplicación Android. La base de datos, es decir, las imágenes que utilizamos para comparar la original, las tenemos almacenadas en la tarjeta SD del teléfono o dispositivo de pruebas.

En la misma tarjeta SD, tenemos almacenados dos archivos con extensión de texto. El primero, llamado “datos_euclidea.txt”, contiene los cálculos de la distancia euclídea (ver punto 3.5) de cada imagen. Esto representa un valor numérico por cada píxel que contenga dicha imagen. Dependiendo del tamaño que le hayamos asignado en Matlab a la imagen (ver punto 3.2) tendremos más o menos píxeles.
Esto influye en el tamaño final del archivo de datos que tenemos que cargar en nuestro teléfono.
Diferenciamos cada imagen en el archivo de texto con un carácter ‘a’ al final de todos los píxeles que la forman.

El segundo archivo de texto ‘nombres.txt’ simplemente contiene el nombre de las imágenes de la base de datos ordenadas alfabéticamente.

4.4 Diagrama de casos de uso

Un diagrama de casos de uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa.

En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior.
Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea.

Los elementos que pueden aparecer en u diagrama de casos de uso son:

- Actores: Un actor es algo con comportamiento, como una persona, un sistema informatizado u organización, y que realiza algún tipo de interacción con el sistema. Se representa mediante una figura humana dibujada con palos. Esta representación sirve tanto para actores que son personas como para otro tipo de actores.

- Casos de Uso: Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en el diagrama de casos de uso una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar el área específica que el actor desea llevar a cabo usando el sistema.

- Relaciones entre Casos de Uso. Un caso de uso, en principio, debería describir una tarea que tiene un sentido completo para el usuario. Sin embargo, hay ocasiones en las que es útil describir una interacción con un alcance menor como caso de uso. La razón para utilizar estos casos de uso no completos suele ser mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación de casos de uso. Las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes dos tipos:

    - Incluye. Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especificado en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso.

    - Extiende. Cuando un caso de uso base tiene ciertos puntos en los cuales, dependiendo de ciertos criterios, se va a realizar una interacción adicional. El caso de uso que extiende describe un comportamiento opcional del sistema.

4.4.1 Diagrama de casos de uso de la aplicación

A continuación tenemos representado el diagrama de casos de uso de la aplicación y, seguidamente, detallaremos uno a uno cada caso de uso.

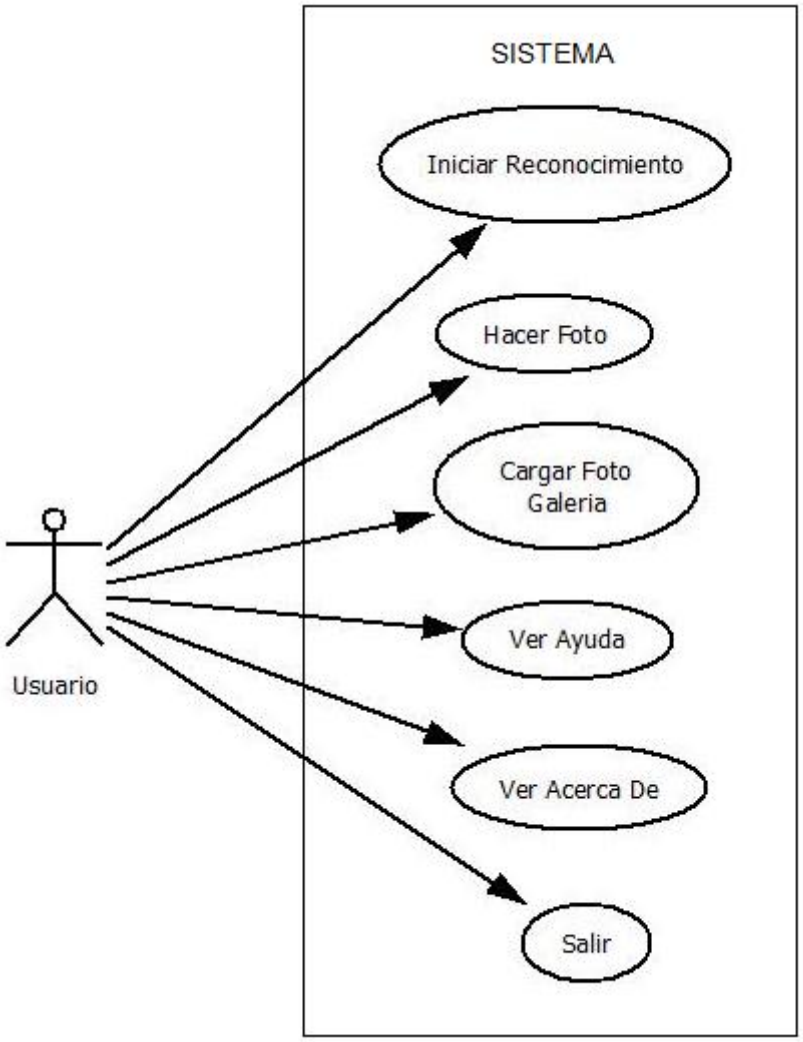


Figura 19: Diagrama casos de uso aplicación Android

4.4.2 Casos de uso de la aplicación

En todos los casos de uso que vamos a detallar a continuación, suponemos que la aplicación está ejecutada

UC0 – Iniciar reconocimiento

Caso de Uso: Iniciar Reconocimiento
Objetivo: Realizar el reconocimiento de una imagen
Actor: Usuario (U)
Precondiciones La aplicación debe estar ejecutada y en el menú principal
Pasos: 1. El caso de uso se inicia cuando el usuario selecciona Iniciar Reconocimiento 2. U: Navega por el menú y selecciona Iniciar Reconocimiento. 3. S: Muestra pantalla de selección de imagen. 4. U: Selecciona como quiere cargar la imagen, desde la cámara (ver UC1) o desde la galería (ver UC2) 5. S: Muestra resultado del reconocimiento por pantalla 6. Fin del caso de uso
Extensiones: 3.1. El usuario quiere volver a la pantalla inicial 3.1.1 U: Pulsa botón Volver en la pantalla 3.1.2 S: Retorna al menú principal 5.1. El usuario quiere volver a la pantalla inicial 5.1.1 U: Pulsa botón Volver en la pantalla 5.1.2 S: Retorna al menú principal

Diagrama secuencia

Flujo normal

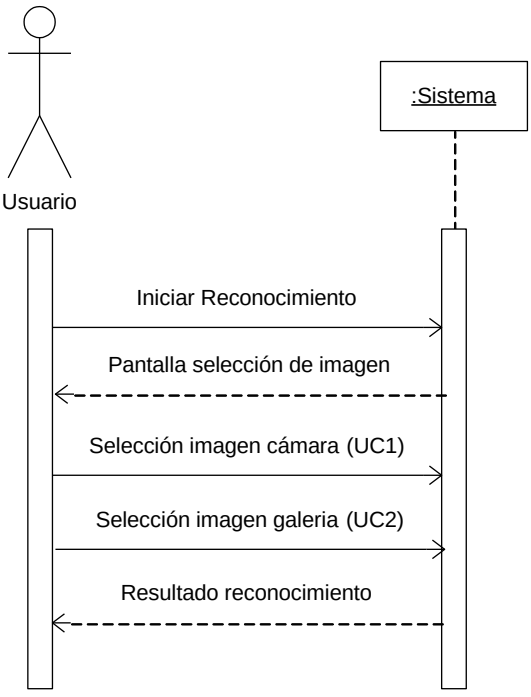


Figura 20: Diagrama secuencia UC0 normal

Extensión 3.1

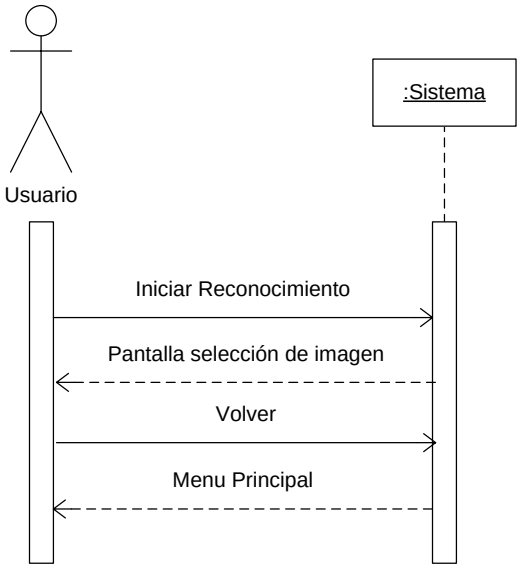


Figura 21: Diagrama secuencia UC0 ext. 3.1

Extensión 5.1

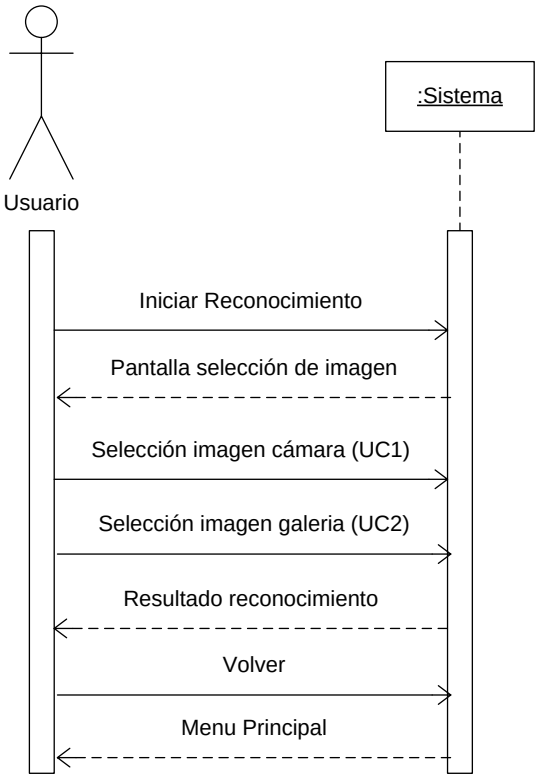


Figura 22: Diagrama secuencia UC0 ext. 5.1

UC1 – Hacer Foto

Caso de Uso: Hacer Foto
Objetivo: Realizar una foto
Actor: Usuario (U)
Precondiciones: Debe estar ejecutándose el Caso de Uso Iniciar Reconocimiento (UC0)
Pasos: 1. El caso de uso se inicia cuando el usuario selecciona hacer una foto para ser tratada 2. U: Navega por el menú y selecciona Cámara 3. S: Abre la cámara de fotos del dispositivo 4. U: Captura la fotografía 5. S: Muestra en pantalla la imagen realizada y botones de confirmar o repetir la acción 6. U: Selecciona Listo 7. S: Muestra mensaje de carga de la fotografía 8. S: Muestra en pantalla completa la fotografía con un recuadro de color verde superpuesto 9. U: Selecciona, ampliando o disminuyendo el recuadro, la parte que desea tratar de la imagen 10. U: Selecciona el botón Guardar 11. Retorna al flujo principal de UC0 (paso 5) 12. Fin del caso de uso
Extensiones: 5.1. El usuario quiere repetir la fotografía 5.1.1 U: Selecciona en la pantalla el botón de repetir fotografía 5.1.2 S: Abre la cámara de fotos del dispositivo 5.1.3 Retorna al flujo principal (paso 4) 9.1. El usuario no desea recortar la imagen 9.1.1. U: Selecciona el botón Guardar sin retocar el recuadro verde 9.1.2 Retorna al flujo principal (paso 11) 10.1. El usuario desea cancelar la selección de la fotografía 10.1.1 U: Selecciona botón Cancelar en la pantalla 10.1.2 Retorna al flujo principal (paso 4)

Diagrama secuencia

Flujo Normal

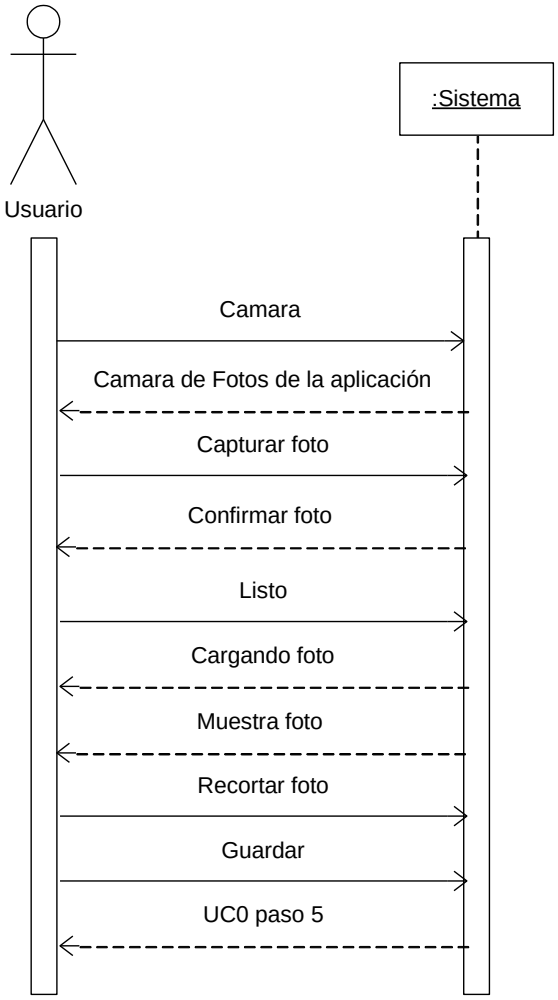


Figura 23: Diagrama secuencia UC1 normal

Extension 5.1

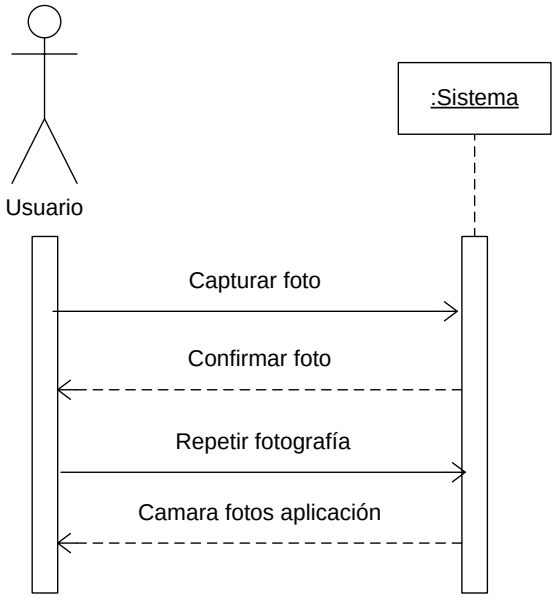


Figura 24: Diagrama secuencia UC1 ext. 5.1

Extensión 9.1

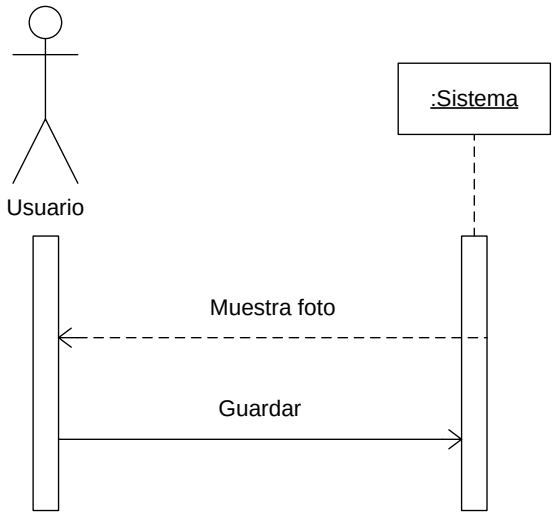


Figura 25: Diagrama secuencia UC1 ext. 9.1

Extensión 10.1

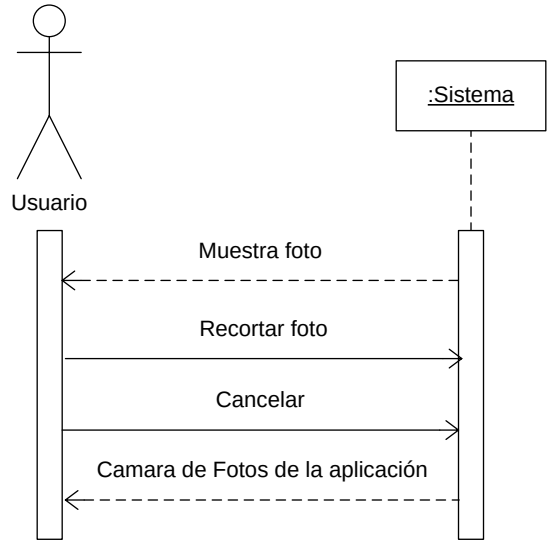


Figura 26: Diagrama secuencia UC1 ext. 10.1

UC2 – Cargar Foto Galería

Caso de Uso: Cargar Foto Galeria
Objetivo: Cargar una foto desde la Galería
Actor: Usuario (U)
Precondiciones: Debe estar ejecutándose el Caso de Uso Iniciar Reconocimiento (UC0)
Pasos: 1. El caso de uso se inicia cuando el usuario selecciona cargar una foto de la galeria para ser tratada. 2. U: Navega por el menú y selecciona Galeria 3. S: Muestra pantalla con los álbumes de fotos que dispone en el ordenador 4. U: Selecciona el álbum donde se encuentra la imagen que quiere cargar 5. S: Muestra pantalla con las fotos del álbum seleccionado 6. U: Selecciona la fotografía 7. S: Muestra mensaje de carga de la fotografía 8. S: Muestra en pantalla completa la fotografía con un recuadro de color verde superpuesto 9. U: Selecciona, ampliando o disminuyendo el recuadro, la parte que desea tratar de la imagen 10. U: Selecciona el botón Guardar 11. Retorna al flujo principal de UC0 (paso 5) 12. Fin del caso de uso
Extensiones: 3.1. El usuario dispone de más de un gestor de carpetas en el dispositivo 3.1.1 S: Muestra mensaje con las diferentes opciones que dispone en el dispositivo para abrir la Galeria 3.1.2 U: Selecciona método para abrir la Galeria 3.1.3 Retorna al flujo principal (paso 4) 9.1. El usuario no desea recortar la imagen 9.1.1. U: Selecciona el botón Guardar sin retocar el recuadro verde 9.1.3 Retorna al flujo principal (paso 11) 10.1. El usuario desea cancelar la selección de la fotografía 10.1.1 U: Selecciona botón Cancelar en la pantalla 10.1.2 Retorna al flujo principal (paso 5)

Diagrama Secuencia

Flujo Normal

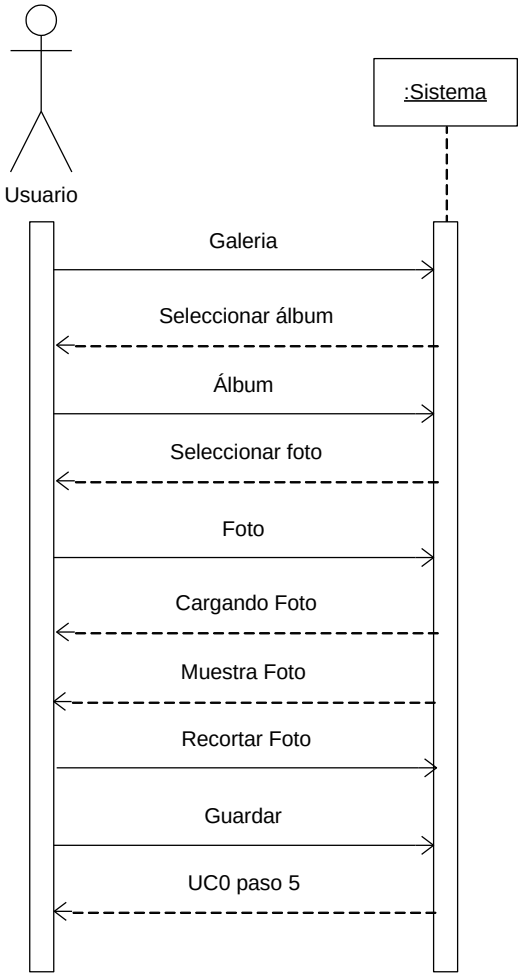


Figura 27: Diagrama secuencia UC2 normal

Extensión 3.1

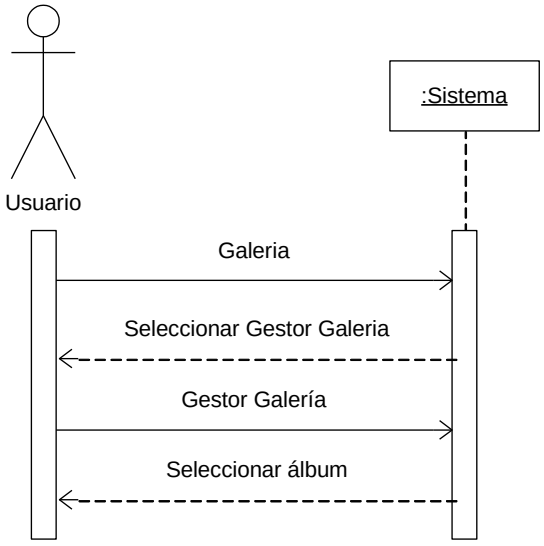


Figura 28: Diagrama secuencia UC2 ext. 3.1

Extensión 9.1

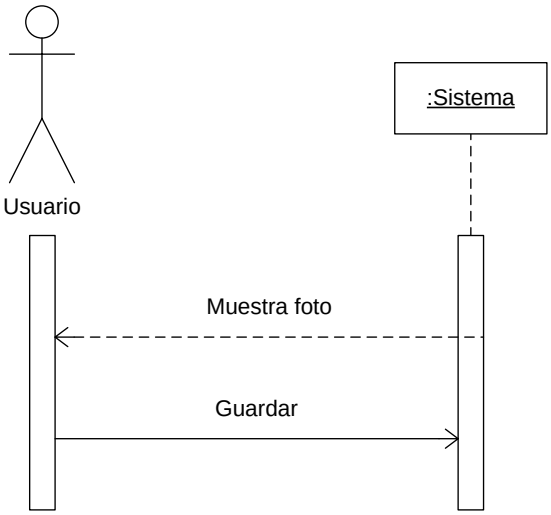


Figura 29: Diagrama secuencia UC2 ext. 9.1

Extensión 10.1

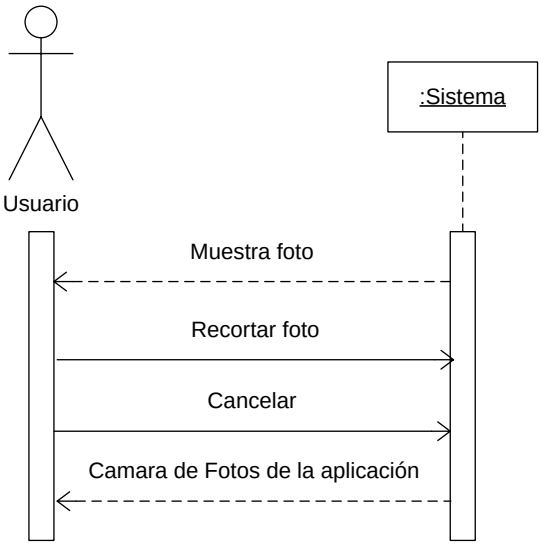


Figura 30: Diagrama secuencia UC2 ext. 10.1

UC3 – Ayuda

Caso de Uso: Ayuda
Objetivo: Consultar ayuda de la aplicación
Actor: Usuario (U)
Precondiciones: La aplicación debe estar ejecutada y en el menú principal
Pasos: 1. El caso de uso se inicia cuando el usuario quiere ver la ayuda de la aplicación. 2. U: Navega por el menú y selecciona Ayuda 3. S: Abre la pantalla de ayuda 4. Fin del caso de uso
Extensiones: 3.1. El usuario quiere volver a la pantalla inicial 3.1.1 U: Pulsa botón Volver en la pantalla 3.1.2 S: Retorna al menú principal

Diagrama de secuencia

Flujo Normal

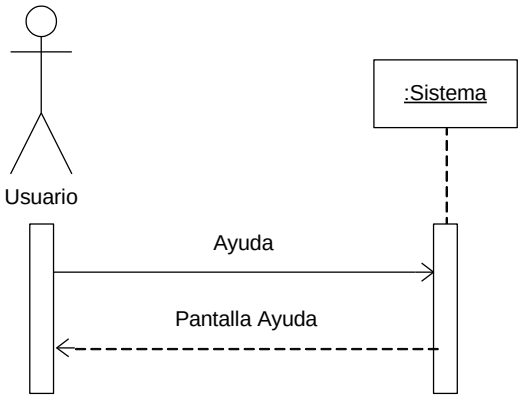


Figura 31: Diagrama secuencia UC3 normal

Extensión 3.1

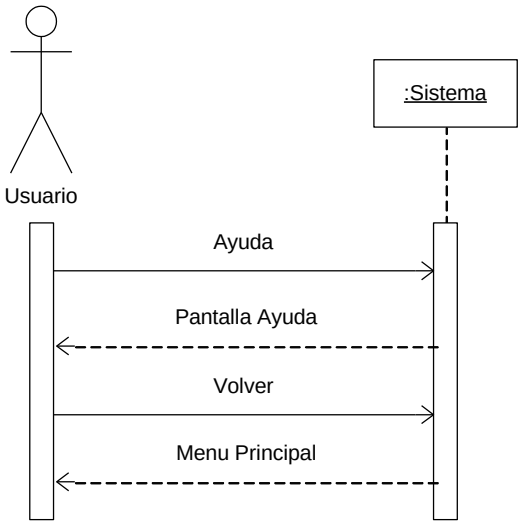


Figura 32: Diagrama secuencia UC3 ext. 3.1

UC4 – Información Aplicación

Caso de Uso: Información Aplicación
Objetivo: Consultar la información de la aplicación
Actor: Usuario (U)
Precondiciones: La aplicación debe estar ejecutada y en el menú principal
Pasos: 1. El caso de uso se inicia cuando el usuario quiere ver la información de la aplicación. 2. U: Navega por el menú y selecciona Acerca de 3. S: Abre la pantalla Acerca de 4. Fin del caso de uso
Extensiones: 3.1. El usuario quiere volver a la pantalla inicial 3.1.1 U: Pulsa botón Volver en la pantalla 3.1.2 S: Retorna al menú principal

Diagrama de secuencia

Flujo Normal

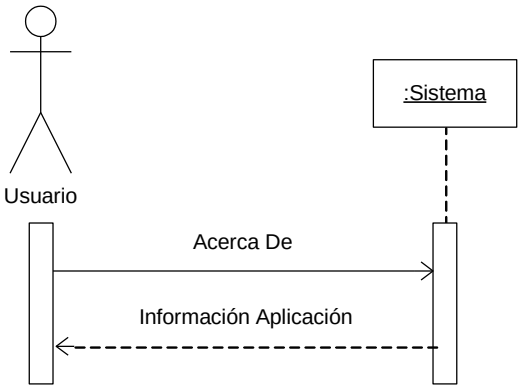


Figura 33: Diagrama secuencia UC4 normal

Extensión 3.1

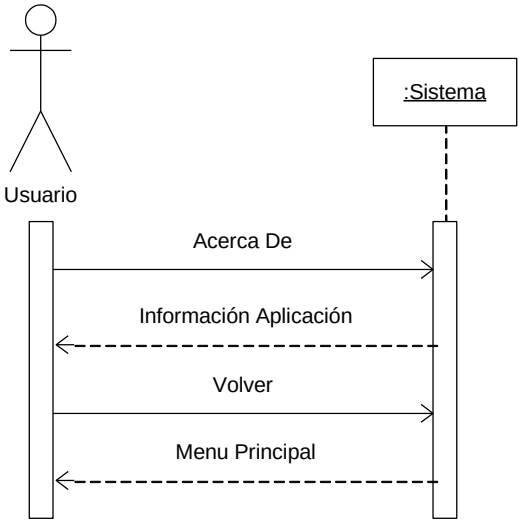


Figura 34: Diagrama secuencia UC4 ext. 3.1

UC5 – Salir

Caso de Uso: Salir
Objetivo: Salir de la aplicación
Actor: Usuario (U)
Precondiciones: La aplicación debe estar ejecutada y en el menú principal
Pasos: 1. El caso de uso se inicia cuando el usuario quiere salir de la aplicación. 2. U: Navega por el menú y selecciona Salir 3. S: Cierra la aplicación 4. Fin del caso de uso
Extensiones:

Diagrama de secuencia

Flujo Normal

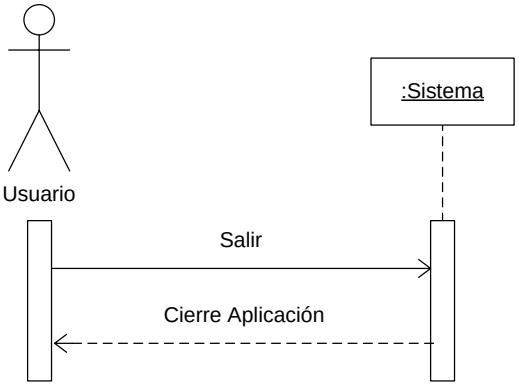


Figura 35: Diagrama secuencia UC5 normal

4.3 Arquitectura del sistema

En este apartado de la memoria, vamos a ver cómo es la arquitectura de nuestra aplicación Android.

Cuando hablamos de la arquitectura de la aplicación, nos referimos a cómo está estructurada, es decir, como se organizan e interactúan internamente entre sí los elementos para satisfacer las peticiones.

Como parte de la arquitectura del sistema, veremos los diagramas de clases y los diagramas de secuencia.

4.3.1 Diagramas de clases

Como diagrama de clases entendemos a un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y relaciones entre ellas. Permite visualizar las relaciones entre las clases que involucran el sistema.
Los diagramas de clase son utilizados en el proceso de análisis y diseño del sistema.

4.3.2 Diagrama de clases aplicación

A continuación podemos ver el diagrama de clases de la aplicación, en el diagrama no aparecen los atributos ni métodos debido a que no se apreciaban visualmente por espacio en la hoja.
A continuación se explicara y mostrará cada clase por separado con todos sus argumentos.
Nuestra aplicación dispone de las siguientes clases:

- MainActivity: Actividad principal de la aplicación. Primera en ejecturase
- AcercaDe: Actividad donde se puede consultar la información relacionada con la aplicación
- Ayuda: actividad donde encontramos la ayuda de la aplicación
- CannyEdgeDetector: actividad donde se realiza el cálculo del algoritmo
- Edge Canny para averiguar el contorno de la foto a tratar.
- Foto: Actividad donde se realizan los cálculos de reconocimiento y se muestran los resultados
- ObtenerFoto. actividad donde se selecciona la foto origen o foto a tratar. Se carga, desde la galería, o se realiza la foto, desde la cámara, y se hace el recorte de la imagen.

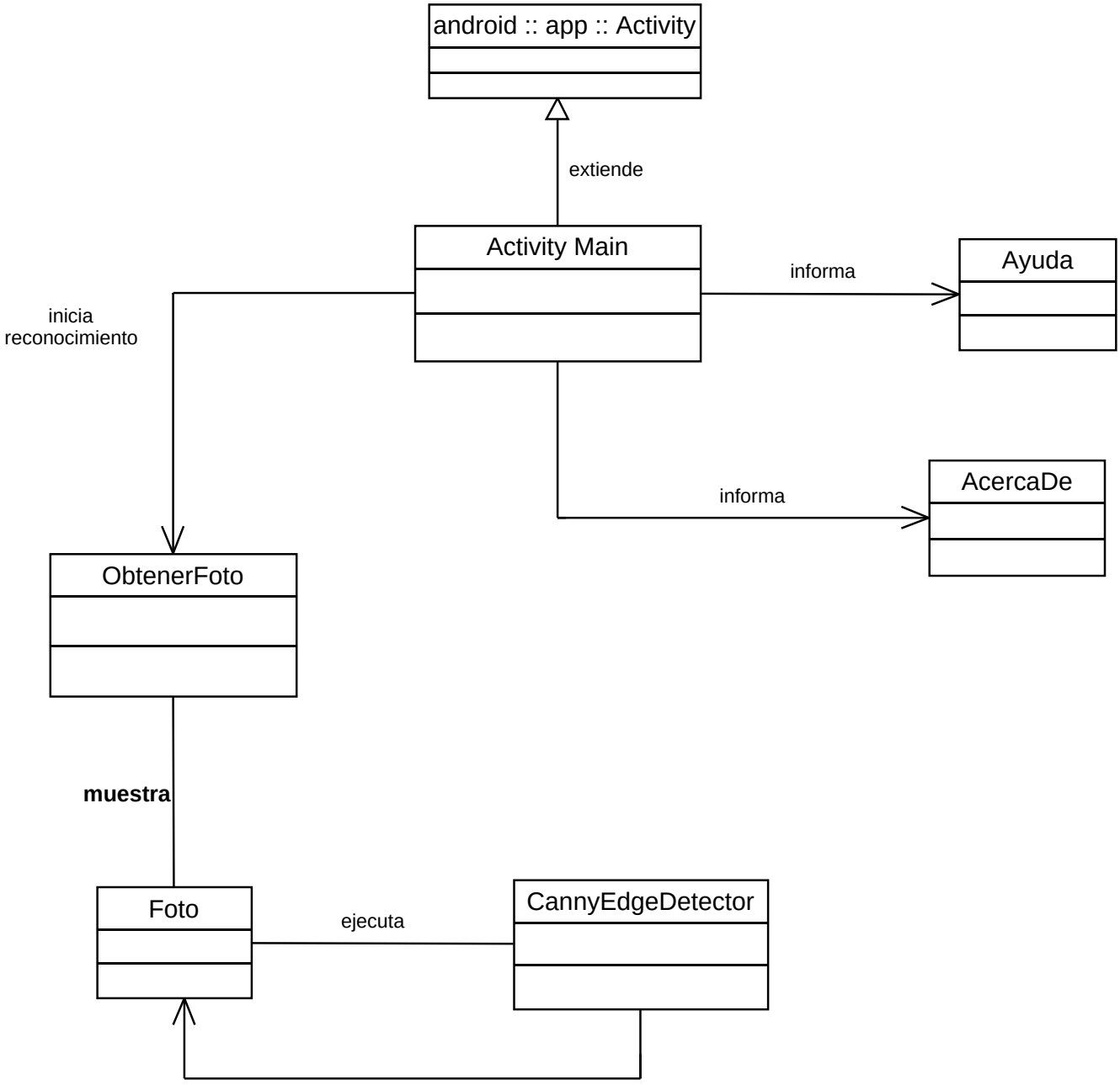


Figura 36: Diagrama de Clases aplicación

4.3.3 Diagrama de Clase: MainActivity (menú principal)

Es el diagrama de clases que corresponde cuando el usuario inicia la aplicación, cuando se ejecuta el menú principal.

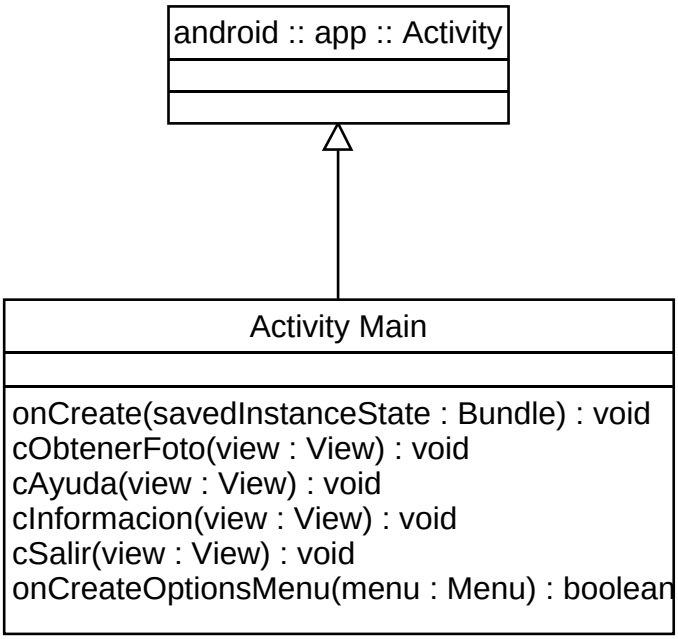


Figura 37: Diagrama clase Main Activity

Es la clase principal, en ella realizamos las llamadas a las diferentes clases que tenemos programadas cuando el usuario pulsa los botones de la aplicación

4.3.4 Diagrama de Clase: AcercaDe

Corresponde cuando el usuario pulsa el botón de la aplicación 'Acerca de'

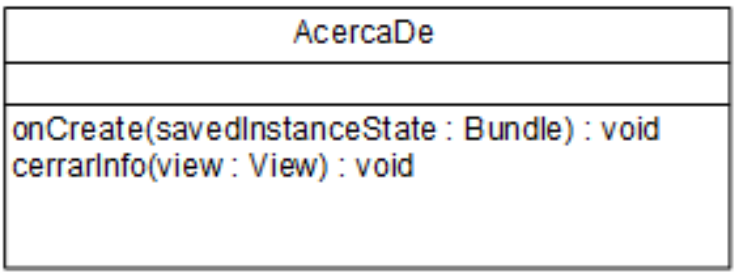


Figura 38: Diagrama clase AcercaDe

Esta clase simplemente crea y ejecuta el layout que contiene la información sobre la aplicación. Es en el mismo layout dónde escribimos dicha información a base de TextViews relacionados con sus respectivas variables de String que contienen el texto que queremos mostrar

4.3.5 Diagrama de Clase: Ayuda

Corresponde cuando el usuario pulsa el botón de la aplicación ‘Ayuda’

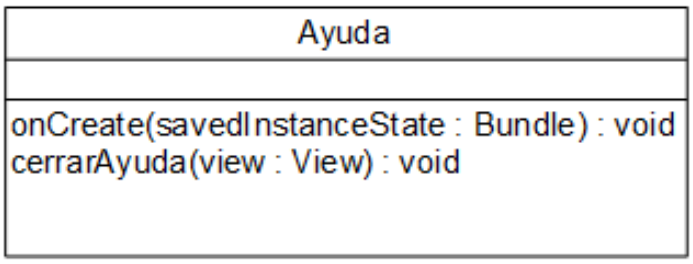


Figura 39: Diagrama clase Ayuda

Esta clase, al igual que la anterior, simplemente crea y ejecuta el layout que contiene la ayuda de la aplicación. Es en el mismo layout dónde escribimos dicha información a base de TextViews relacionados con sus respectivas variables de String que contienen el texto que queremos mostrar

4.3.6 Diagrama de Clase: Obtener Foto

Corresponde cuando el usuario pulsa el botón de la aplicación ‘Iniciar Reconocimiento’

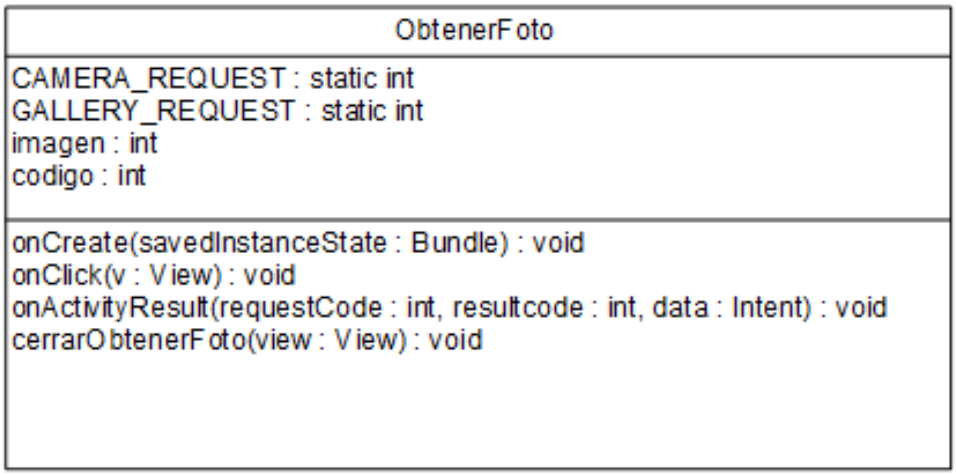


Figura 40: Diagrama clase ObtenerFoto

Es la clase que se ejecuta cuando el usuario inicia el reconocimiento de la imagen.
Se encarga de obtener la fotografía que se va a utilizar para tratar el reconocimiento.
En el método onclick() diferencia entre si la petición de subir la foto es desde la cámara o desde la galería.
Cuando ya se ha seleccionado la foto inicia la Clase Foto

4.3.7 Diagrama de Clase: Foto

Corresponde cuando el usuario pulsa el botón de la aplicación ‘Foto’ viene derivada de la Clase ‘ObtenerFoto’

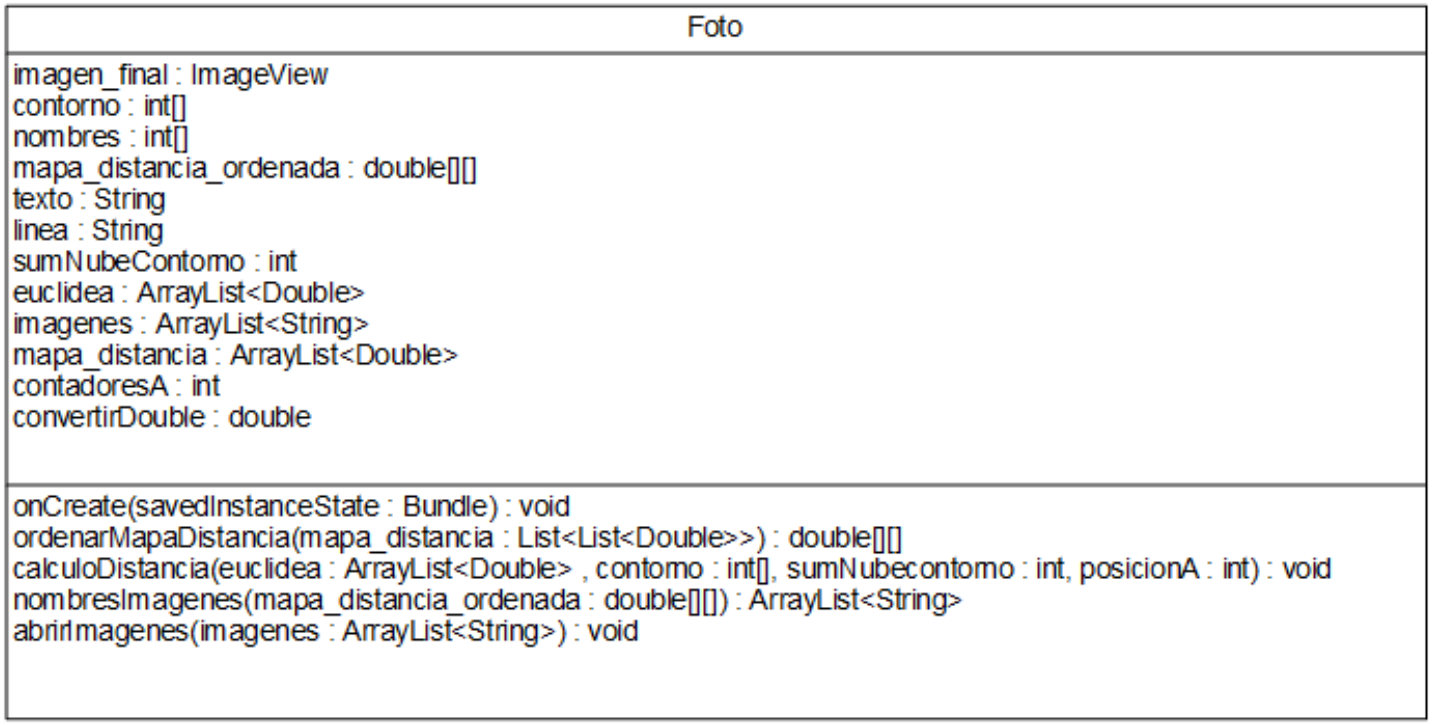


Figura 41: Diagrama clase Foto

Es la clase que realiza los cálculos y muestra los resultados por pantalla. Recibe de la anterior clase ‘ObtenerFoto’ la fotografía origen que ha seleccionado el usuario.

Ejecuta la clase ‘CannyEdgeDetector’ la cual le devuelve el contorno de la imagen origen.

Compara y realiza los cálculos necesarios con el archivo de la base de datos y ejecuta el layout con los resultados obtenidos

4.3.8 Diagrama de Clase: CannyEdgeDetector

Es una clase que activa la clase 'Foto' cuando recibe la imagen seleccionada. El usuario no interactua con esta clase

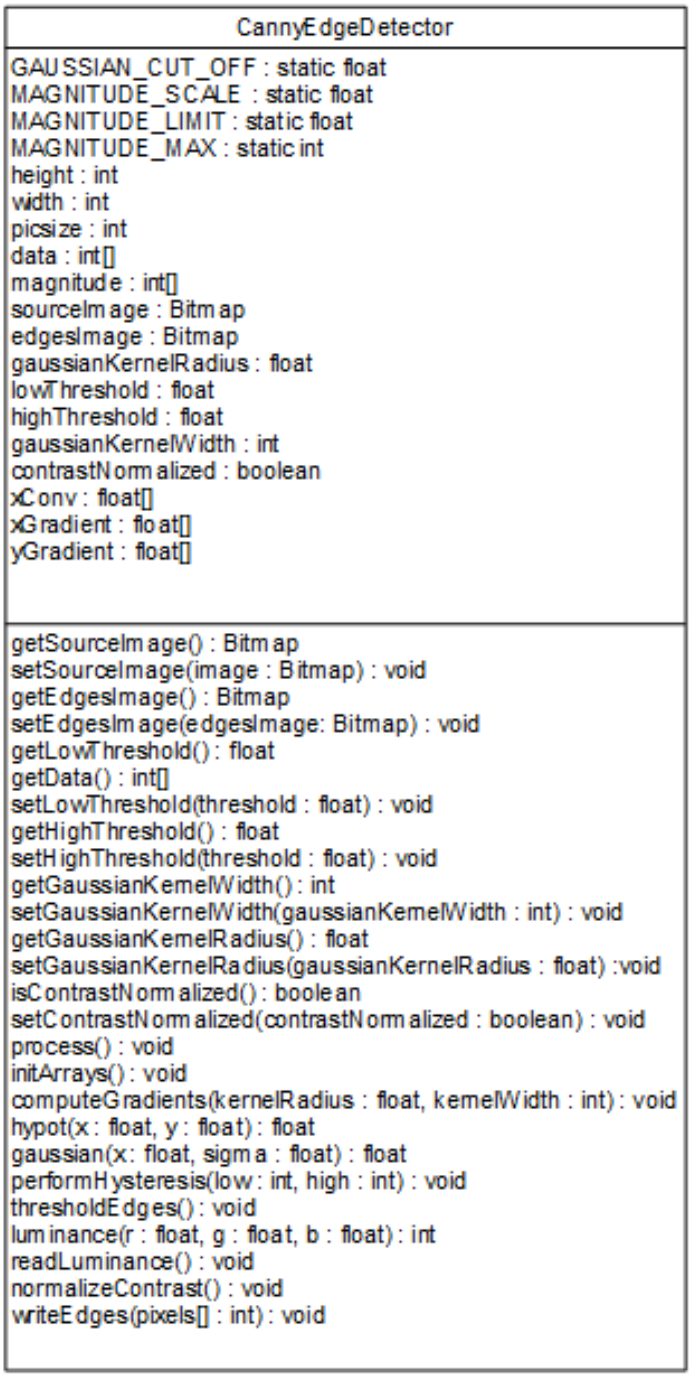


Figura 42: Diagrama clase CannyEdgeDetector

Es la clase encargada de realizar el cálculo del contorno de la imagen original seleccionada por el usuario. Los getters y setters permiten que desde la clase 'Foto' se puedan obtener los datos.

El método que "activa" al resto de métodos y que inicia los cálculos es process() y es el método que llamamos desde la clase 'Foto'

4.3.9 Diagramas de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo.

A continuación mostramos los diagramas de secuencia que podemos encontrar en la aplicación

4.3.10 Diagrama secuencia: Iniciar Reconocimiento

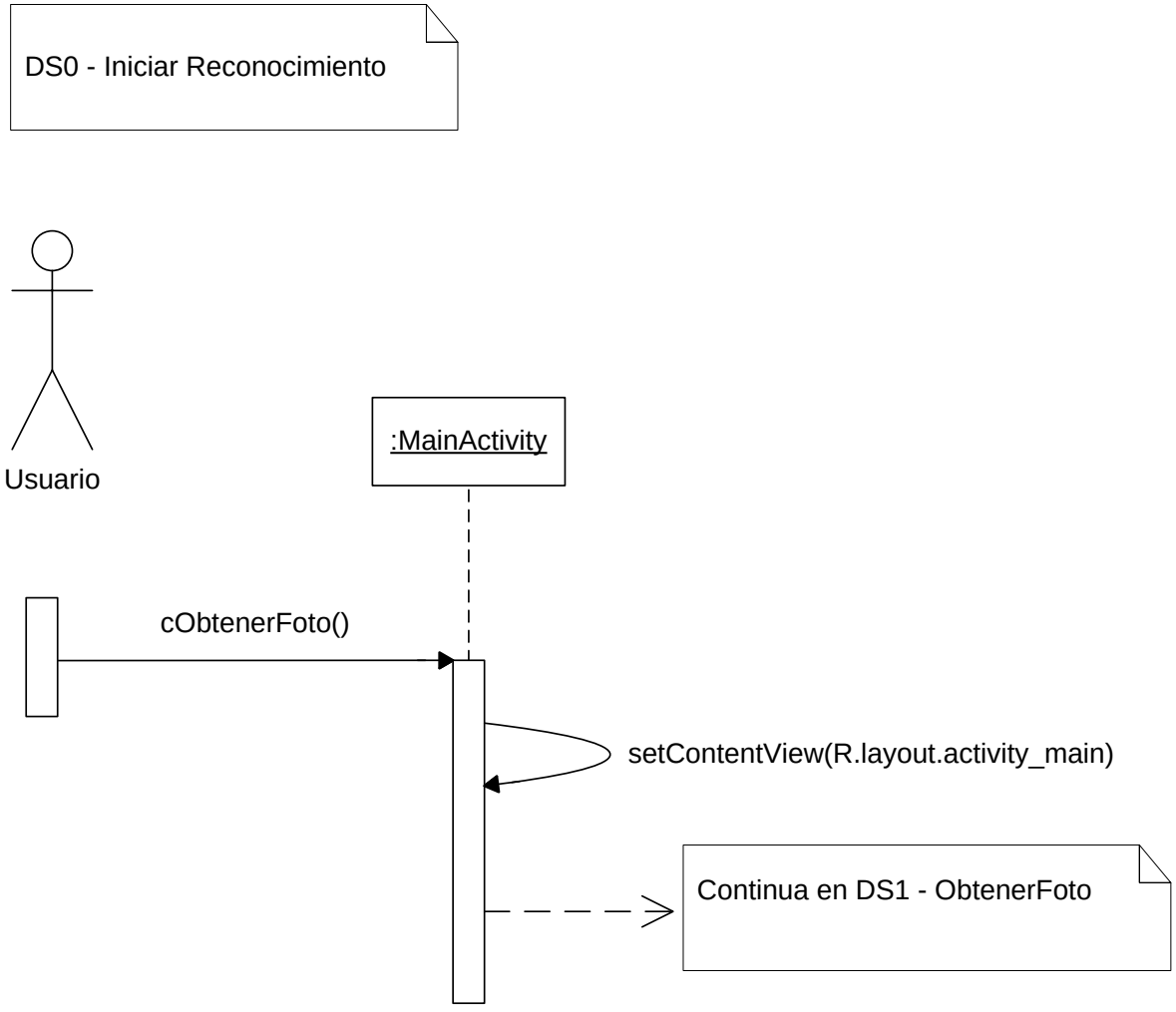


Figura 43: Diagrama secuencia Iniciar Reconocimiento

4.3.11 Diagrama secuencia: Obtener Foto

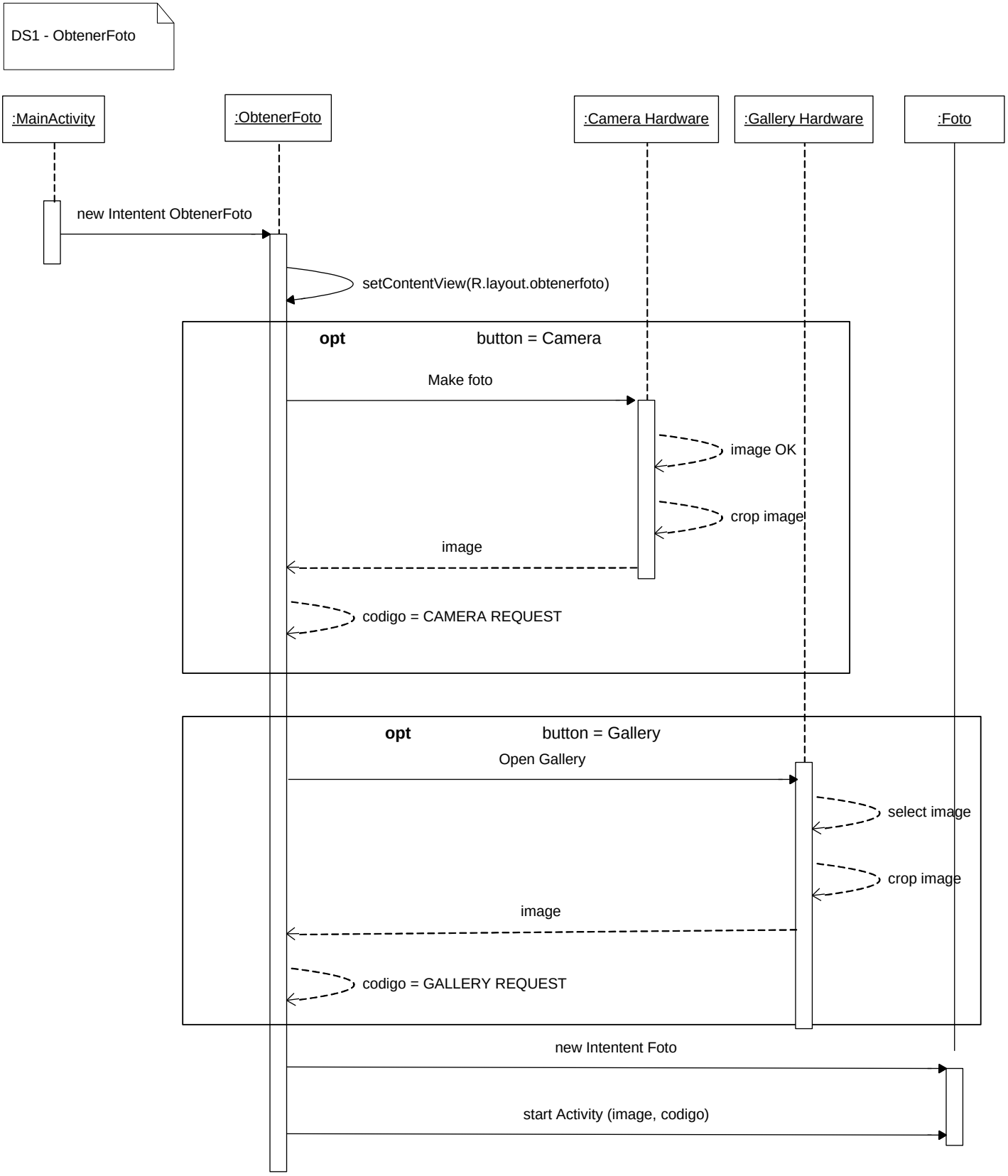


Figura 44: Diagrama secuencia Obtener Foto

4.3.12 Diagrama secuencia: Foto

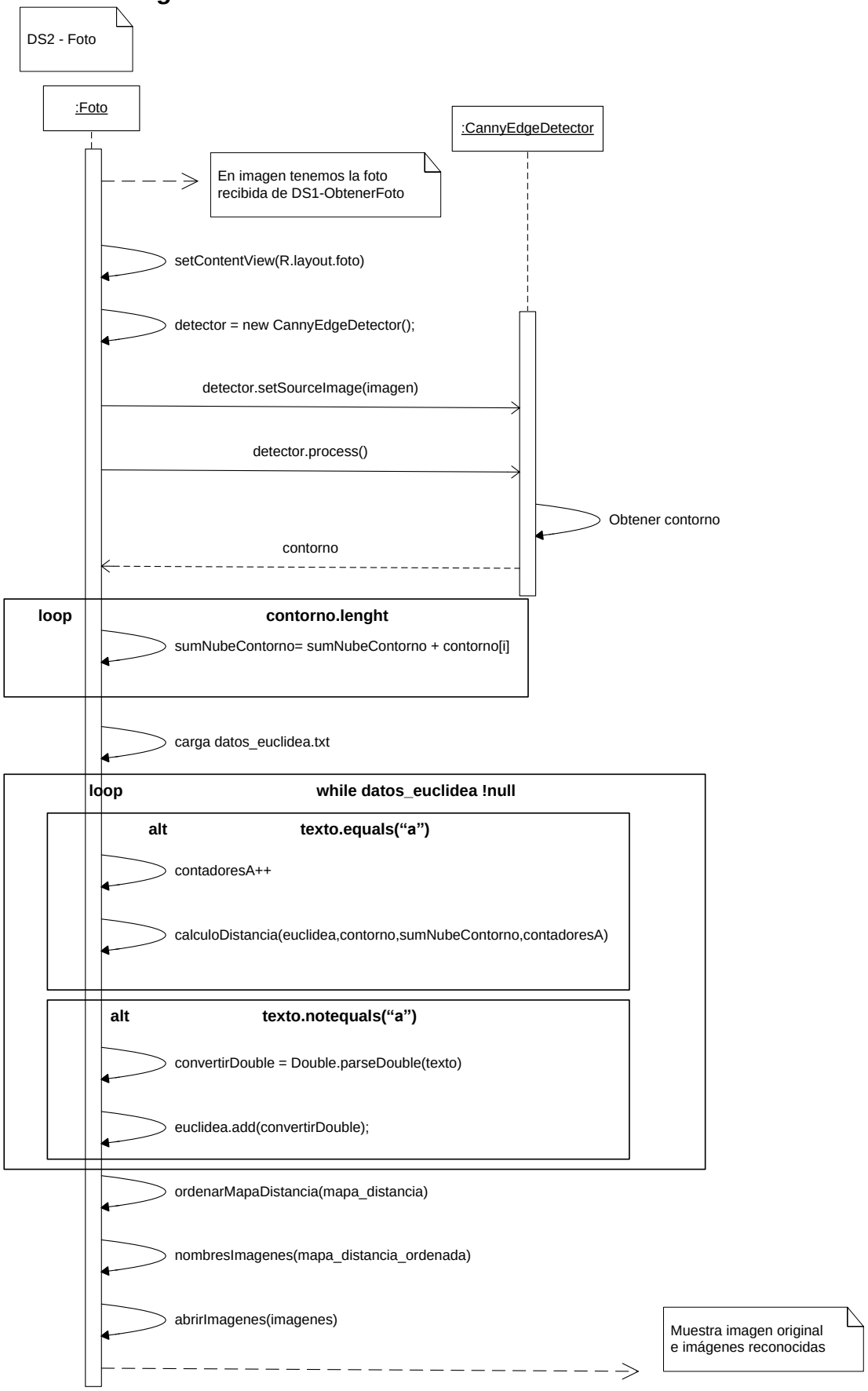


Figura 44: Diagrama secuencia Obtener Foto

4.3.13 Diagrama Secuencia: Ayuda

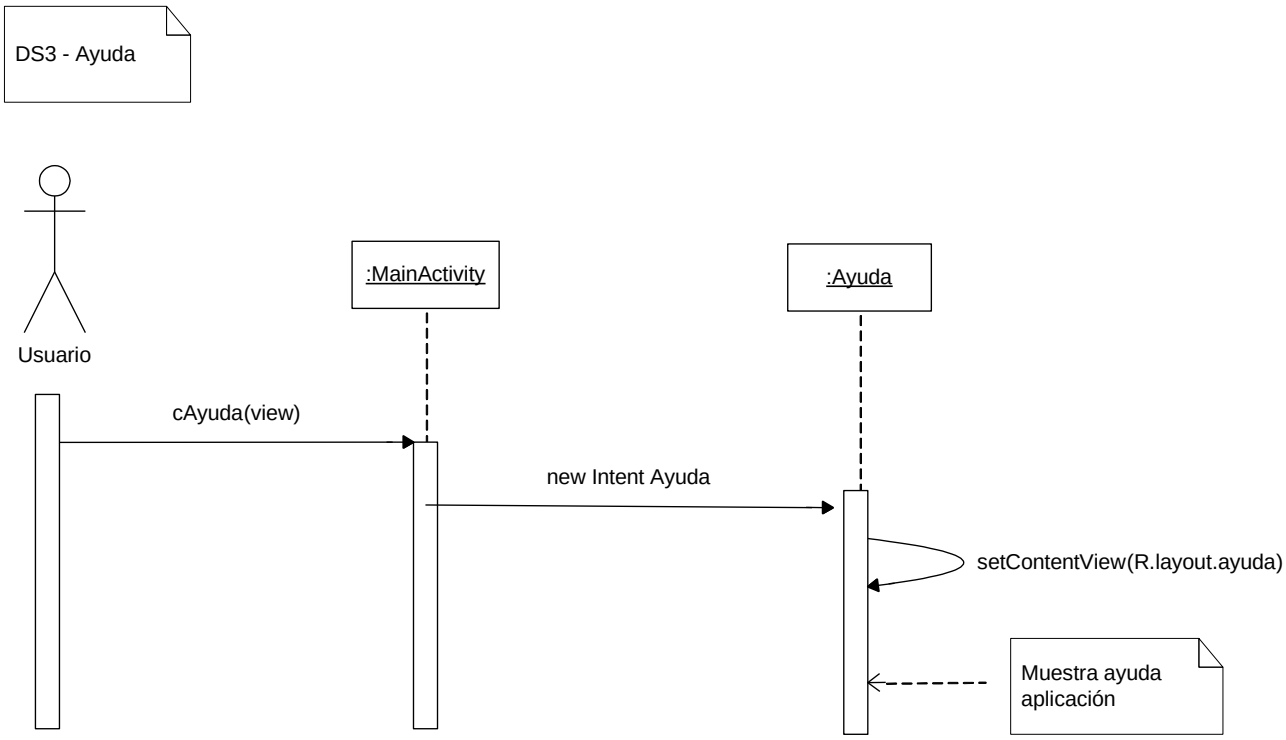


Figura 45: Diagrama secuencia Ayuda

4.3.14 Diagrama Secuencia: Información Aplicación

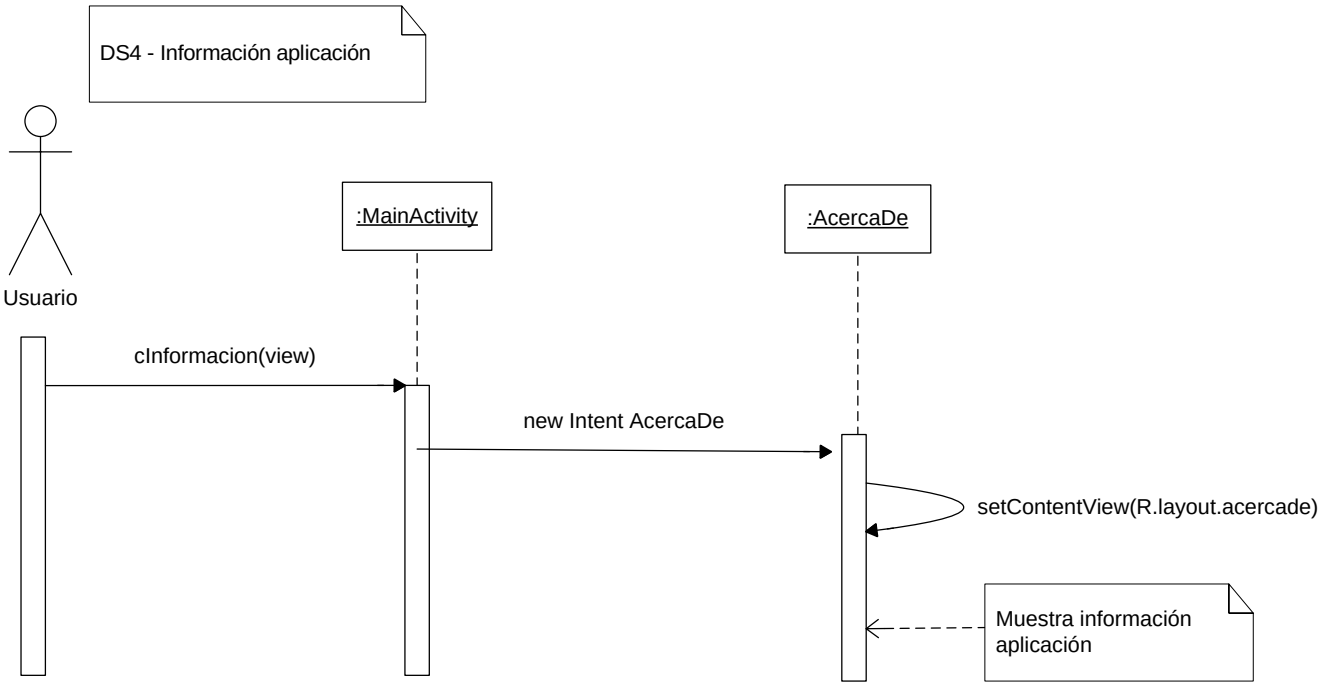


Figura 46: Diagrama secuencia Información aplicación

4.3.15 Diagrama de secuencia: Salir

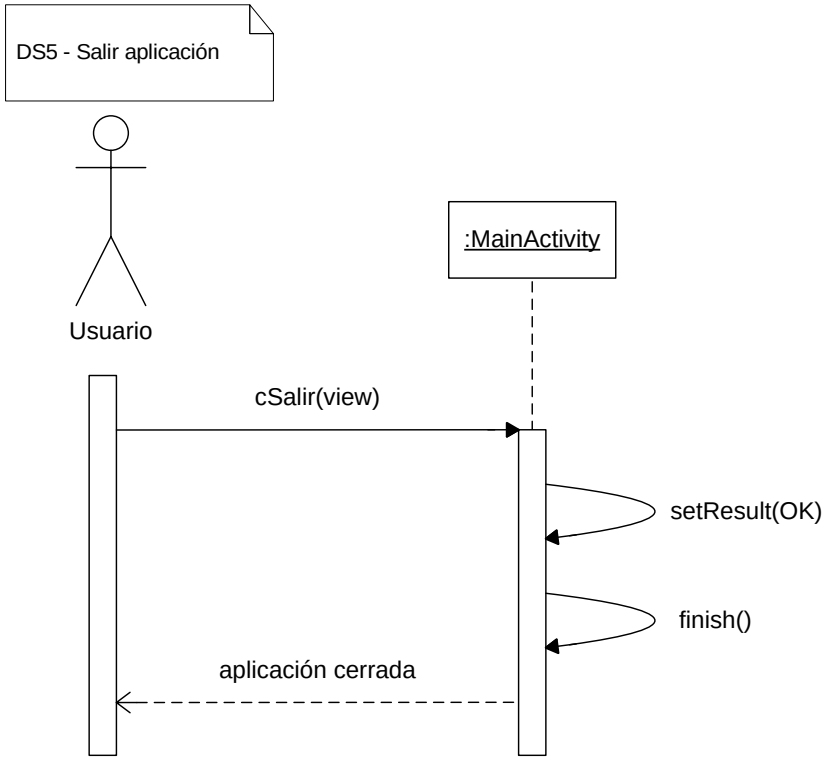


Figura 47: Diagrama secuencia Salir

5. Resultados

5.1 Matlab

Para realizar el entorno de pruebas y comprobar sus resultados, utilizamos la base de datos de imágenes MPEG-7 CE Shape 1.

Consiste en una colección de 1400 imágenes divididas en 70 clases de 20 imágenes consideradas similares entre ellas. Esta biblioteca de imágenes fue especialmente formada para permitir comparar la performance de los diferentes algoritmos basados en devolución por similaridad.

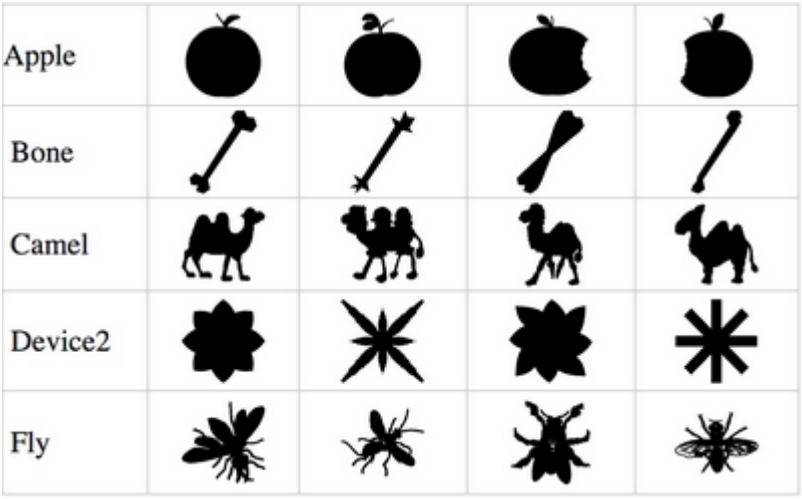


Figura 48: Algunas imágenes de la base de datos MPEG-7

Como hemos estado viendo con ejemplos en el apartado 3.3 y 3.4, el tratamiento de la imagen seleccionada por el usuario es correcto, es decir, conseguimos un contorno de la imagen casi perfecto.

A continuación vamos a mostrar dos de las pruebas realizadas con el programa, una con una imagen en color y otra con una imagen en blanco y negro (imagen extraída de la misma base de datos a comparar).

Vamos a explicar qué nos mostrará el sistema como resultado obtenido.
La pantalla muestra cinco imágenes. La primera, en la parte superior izquierda, es la imagen original extraída de la base de datos. La segunda imagen, en la parte superior derecha, corresponde a la imagen una vez hemos realizado el proceso de recortar la parte que deseamos comparar.

En la parte inferior, el sistema nos muestra las 3 imágenes que ha encontrado más parecidas en la base de datos. El orden es de más a menos y de izquierda a derecha.

La primera prueba consiste en realizar la comparación de imágenes escogiendo una imagen origen de la misma base de datos de las imágenes finales.
Para ello, seleccionamos al azar la imagen ‘bottle-02.gif’.
En la base de datos MPEG-7 existen 20 imágenes de botellas, unas orientadas a izquierda y otras orientadas a derecha.

Figura 49: Imagen de ‘bottle-02’

En esta ocasión no recortamos la imagen y la seleccionamos entera. El resultado obtenido es el siguiente

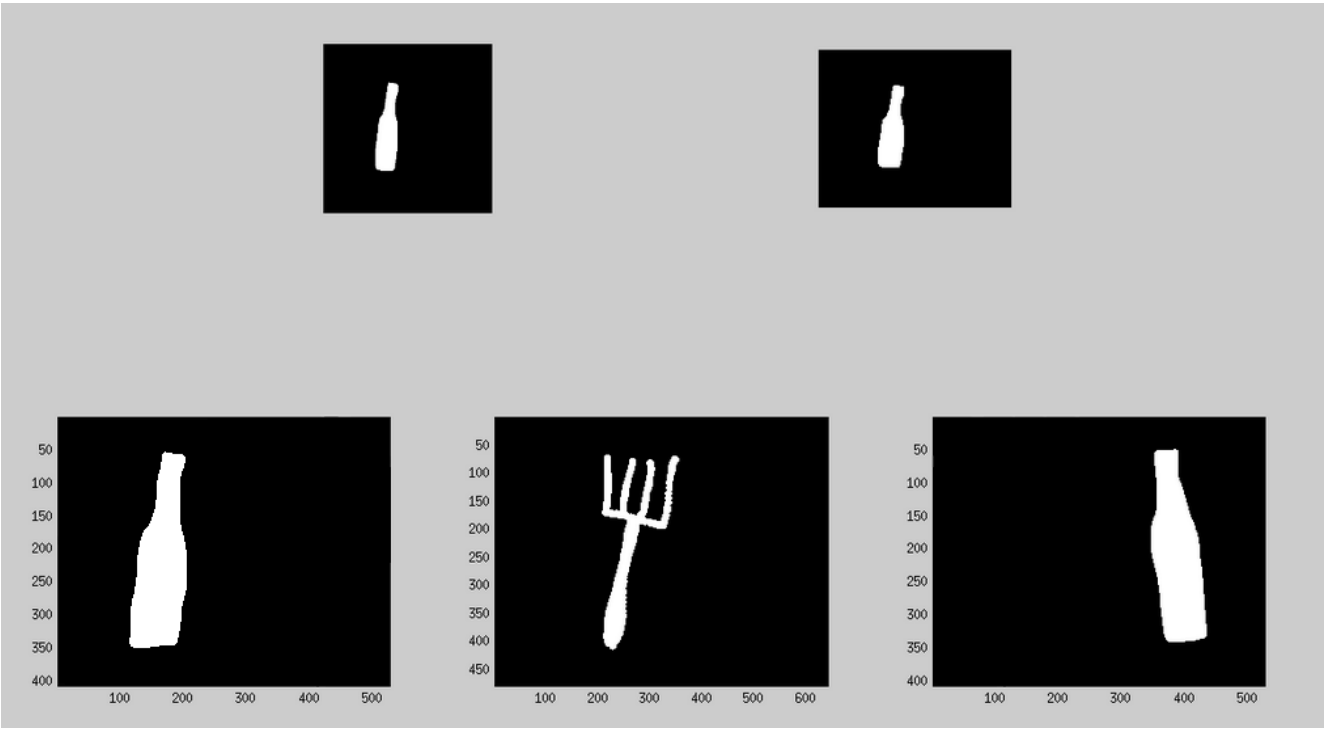


Figura 50: Imagen resultado de prueba con ‘bottle-02’

Podemos observar que la primera imagen encontrada es ella misma, la siguiente es un tenedor con una forma muy similar a la botella y la tercera es otra botella pero orientada hacia el lado contrario.

Podemos afirmar que en la primera imagen el programa realiza perfectamente su función.
Si paramos atención en la segunda imagen, realmente en forma y tamaño es muy similar a la botella y, además, ocupa un espacio en píxeles muy parecido.
También podemos observar que, cuando la parte de la botella disminuye su tamaño, en el cuello, el tenedor es cuando más espacio de píxeles ocupa y, por otra parte, cuando la botella ocupa más espacio (en el cuerpo), es cuando el tenedor se hace más delgado (en el mango).
La tercera imagen, realmente es una botella igual que la imagen seleccionada para hacer la prueba, aunque ésta está orientada hacia un lado que no corresponde.

La segunda prueba tiene la misma finalidad lo que la imagen seleccionada es una al azar que no corresponde a la base de datos de la aplicación.
Hemos utilizado la siguiente foto de un perro para la realización:

Figura 51: Imagen para realizar la prueba

Al tener la imagen mucho fondo blanco, recortamos la imagen para seleccionar únicamente la parte donde se encuentra el perro.

El resultado obtenido es el siguiente:

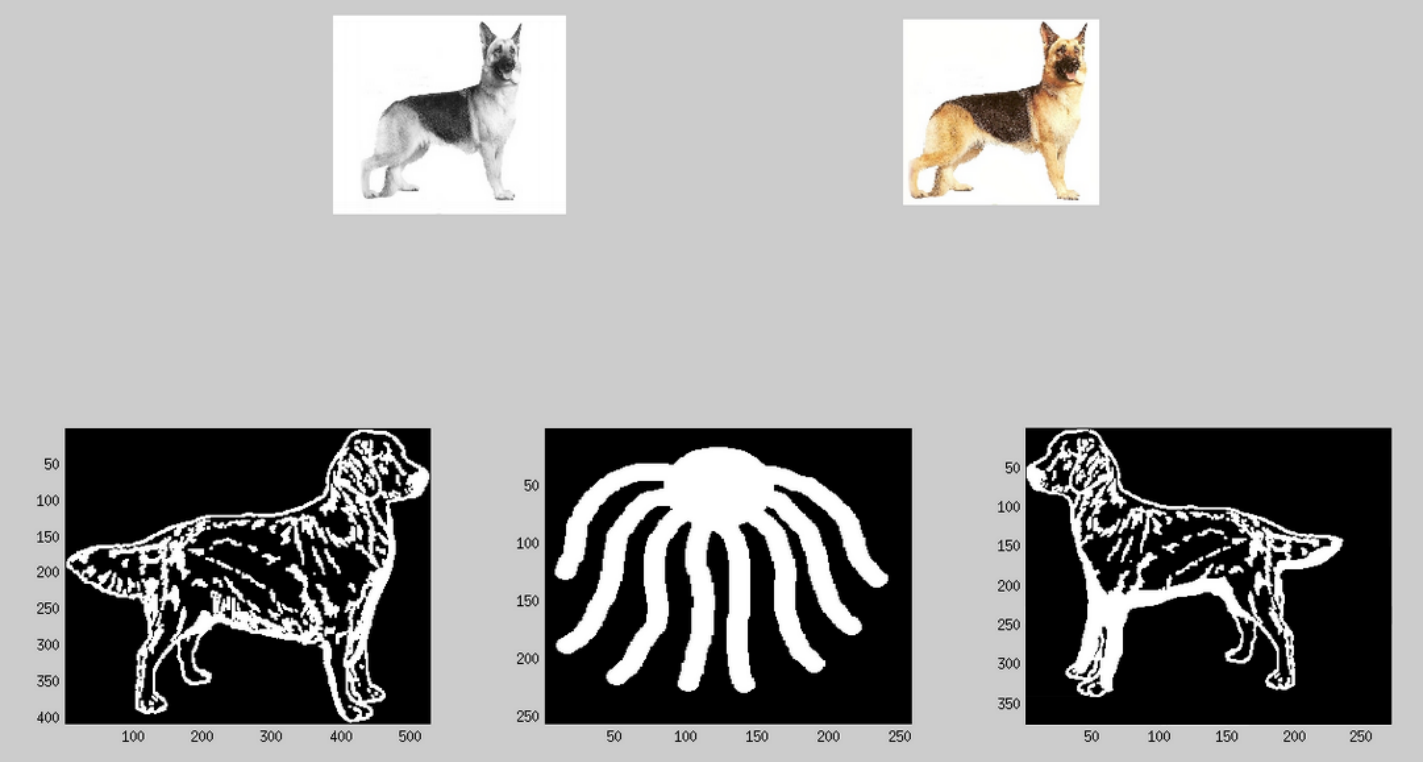


Figura 52: Imagen resultado de prueba con imagen de perro

En este caso el sistema nos muestra la imagen original en escala de grises para que podamos comparar más fácilmente los píxeles que se han utilizado en la comparación.

Como en el caso de la primera prueba, observamos que la primera imagen encontrada es un perro, es decir, la aplicación ha funcionado correctamente ya que, tanto la orientación como la forma son bastante similares y, a su vez, es la imagen más parecida a la original de todas las que existen en la base de datos.

La segunda imagen que nos muestra el sistema corresponde a un pulpo. Lógicamente un pulpo no se parece en nada a un perro. Como en la primera prueba, el único parecido es en extensión de espacio y distribución de píxeles, la cola del perro podría confundirse con brazos del pulpo y a su vez cada pata del perro podría ser un brazo del pulpo.
La tercera imagen corresponde a un perro de la base de datos pero orientado en lado contrario al de la foto original, igual que nos ocurría en la primera prueba.

5.2 Android

Las pruebas que hemos realizado para testear la aplicación han sido varias y en tres tipos de medios distintos.

La primera, la más sencilla, comprobaba que la interficie gráfica fuera correcta. Para ello utilizamos la maquina virtual de eclipse AVD.
Los AVD (Android Virtual Device) son los “dispositivos virtuales” que nos permiten probar versiones especificas de Android así como configuraciones de hardware.

Esto nos permitía visualizar en el emulador, cómo se verían las pantallas, el comportamiento de los botones, el paso entre actividades, etc.

No hubo ningún problema grave que no se pudiera solucionar in situ al realizar estas pruebas.

Las pruebas del funcionamiento de la aplicación, no se han podido realizar en la máquina virtual de Eclipse ya que se tenía que utilizar la cámara de fotos del dispositivo o acceder a la tarjeta SD para obtener la imagen de la galería.

Se han realizado pruebas con diferentes tamaños de archivo de los cálculos de la distancia euclídea proporcionados por Matlab (ver punto 4.3).
En la primera prueba, el tamaño total del mismo supera los 300 Megabytes. Esto hace que al ejecutar la aplicación, primero el tiempo que tarda en ejecutar todo el proceso sea muy alto y además, al final, no muestre ninguna fotografía en el dispositivo.

Por ello, realizamos una “mini” base de datos, con un número limitado de imágenes, generando así un archivo de datos de 500KB.
Al ejecutar la aplicación comprobamos que el tiempo de ejecución disminuye considerablemente y que nos muestra imágenes como resultado final.

Cargando la misma imagen origen, la seleccionada por el usuario y teniendo la misma base de datos (tanto de cálculos, como de nombres y de imágenes) el programa nos muestra diferentes imágenes en diferentes ejecuciones de la aplicación.

Las pruebas realizadas las veremos in situ el día de la presentación tanto en vídeo como en ejemplos en tiempo real.

6. Conclusiones

Si hablamos de la parte de prototipado Matlab, vistos los resultados, podemos afirmar que el programa funciona perfectamente con el primer resultado comparado. Tanto en la primera prueba, donde la imagen fue la misma, como en la segunda, donde la imagen mostrada era la más similar en la base de datos.

La segunda imagen que nos muestra el programa es donde encontramos más errores. Dicha imagen sería correcta si solo tuviésemos en la base de datos una imagen de cada ejemplo, es decir, una imagen de una botella o una sola imagen de un perro. Al no ser así y tener tanto 20 fotos de botellas y 20 fotos de perros el resultado obtenido no es correcto. Como hemos podido observar, únicamente es parecido en tamaño y extensión en la zona de la pantalla. También, el resultado obtenido en la primera prueba es, bastante más parecido a la imagen original que no el resultado de la segunda prueba.

La tercera imagen, aunque la imagen que nos muestre tenga diferente orientación a la origen, la podemos dar como buena.
Como hemos visto antes, al realizar el cálculo de la transformada de la distancia, la distancia entre píxeles nulos y píxeles no nulos es la misma tenga la orientación que tengan. Al no variar esto, el resultado del cálculo del mapa de distancias no varía y la similitud con la imagen cargada es muy alta.

En la segunda parte, la aplicación Android, el resultado obtenido es bastante incierto.
Para empezar, podemos confirmar que con toda la base de datos cargada en el teléfono, la aplicación no pasa el test de estrés ya que no muestra ninguna imagen como resultado final.
Esto es debido a que se producen tiempos muertos de espera en el dispositivo y que tenga que realizar tantos cálculos que se produzcan excepciones de memoria.

Cuando cargamos la base de datos reducida, los resultados que nos muestra la aplicación no tienen lógica, ya que teniendo la misma imagen origen nos muestra imágenes diferentes de la base de datos que, a veces coinciden (siendo la misma o del mismo rango) y a veces no tienen nada que ver.

Según hemos podido acotar el error, creemos que el problema reside en la matriz final que nos genera la clase CannyEdgeDetector que nos saca el contorno de la imagen origen.
Al realizar el cálculo (calculosdistancia()) de esta matriz con los datos que obtenemos del archivo datos_euclidea, se generan unos valores erróneos que hacen que el orden de las distancias no coincidan con los verdaderos.
Podemos creer que simplemente es “suerte” las veces que coinciden en el resultado final dichas imágenes con la proporcionada por el usuario

Llegamos a la conclusión de que el prototipado en Matlab funciona correctamente y que nuestra aplicación Android tiene fallos en lso cálculos finales y, por tanto, en el resultado final

6.1 Propuestas de mejora

Una de las propuestas de mejora es el tiempo que la aplicación MATLAB tarda en realizar todo el proceso de cálculo y tratamiento de la imagen.
El tiempo necesario varía dependiendo del tamaño o el número de píxeles que asignemos a las matrices de datos ya que cuanto mayor sea la matriz, mayor puntos tendrá y más numero de cálculos tendrá que realizar. Por eso realizamos el escalado de las imágenes antes de tratar y una vez tratadas.

Cada vez que el usuario carga una imagen para tratar, el sistema procesa el contorno de cada una de las imágenes que hay en la base de datos y, calcula la distancia euclídea y el mapa de distancias que forma junto con la imagen original seleccionada por el usuario.

Podríamos almacenar en un archivo (con extensión por ejemplo txt) la matriz que forma el contorno con el edge canny calculado de cada imagen de la base de datos.
Una vez cargada la imagen por el usuario, accederíamos al archivo con los datos de las imágenes de la base de datos y, seguidamente el sistema únicamente calcularía la distancia eculidea y el mapa de distancias en tiempo real.

Actualmente, en el programa, tenemos que distinguir, antes de cada inicialización del mismo, si la foto que el usuario nos va a cargar es en escala de grises o en color.
Esto es debido a que la función Edge Canny trata las imágenes siempre en escala de grises o bidimensional y, de esta forma, si seleccionásemos una imagen en color y no la transformásemos, el sistema mostraría un mensaje de error como el siguiente:

```
??? Error using ==> iptcheckinput
Function EDGE expected its first input, I, to be two-dimensional.

Error in ==> edge>parse_inputs at 541
iptcheckinput(I,{'numeric','logical'},{'nonsparse','2d'},mfilename,'I',1);

Error in ==> edge at 197
[a,method,thresh,sigma,thinning,H,kx,ky] = parse_inputs(varargin{:});
```

Error producido al tratar imágenes sin transformar a escala de grises

Este error simplemente nos está indicando que la imagen que se espera tiene que ser bidimensional (0s y 1s).

Para solucionar este problema, podríamos crear una condición que recorriera la matriz de la imagen origen y, si encontrara un valor diferente de 1 o 0, automáticamente tratara la imagen como color y la transformase en escala de grises.
Si al finalizar la condición no se hubiesen encontrado valores diferentes a 0 o 1, la imagen continuaría siendo tratada como bidimensional y no se realizaría la transformación.

Como es lógico, una de las propuestas de mejora para la aplicación Android es conseguir que el funcionamiento final sea el correcto y el marcado en los objetivos iniciales.

Otra de las propuestas de mejora es el tiempo que la aplicación tarda en ejecutar el reconocimiento. Vemos que cuanto más limitamos la base más rápido se ejecuta, igualmente, cuanto mejor es el dispositivo más rápido funciona (en las pruebas realizadas con la tablet ARCHOS el tiempo disminuía considerablemente que cuando era ejecutado en el dispositivio HTC Desire).

Otra de las propuestas que se están desarrollando es crear la aplicación como cliente-servidor.
Constaría de tener un servidor webservices (realizado en .NET) y que la aplicación interactuase con el servidor.

La aplicación Android únicamente realizaría el trabajo de enviar la foto que el usuario eligiese, mientras que el webservice recibiría la imagen, realizaría la conexión con Matlab y sería el propio programa Matlab en tiempo real que realizaría los cálculos
Al finalizar los cálculos el webservice buscaría en una base de datos almacenada en el propio servidor la imagen como resultado final y se la enviaría al dispositivo.
Con esto ganaríamos en tiempo y en trabajo realizado por el dispositivo, ya que solo enviaría y recibiría las imágenes.

También se podría desarrollar una parte en que el usuario pudiese ampliar la base de datos de imágenes y de esta manera ganar en número de imágenes para comparar.
Simplemente se habilitaría el acceso a la tarjeta SD del dispositivo y se guardaría la imagen que el usuario quisiese en la carpeta especificada.

7. Bibliografía

La mayoría de conocimientos a la hora de desarrollar en Matlab ha sido adquiridos en:

• http://www.mathworks.es → Documentación oficial de Matlab

• http://www.mat.ucm.es → Introducción a Matlab Universidad

• http://stackoverflow.com/ → Foro de múltiples temáticas

• http://oefa.blogspot.com.es/ → Blog especializado en Matlab

• Javier García de Jalón, José Ignacio Rodriguez, Jesús Vidal, aprenda Matlab 7.0 como si estuviera en primero, Madrid, diciembre 2005, Universidad Politécnica de Madrid

Por otra parte a la hora de desarrollar en Android hemos consultado:

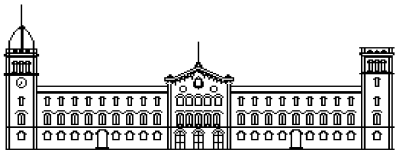
• http://developer.android.com/index.html → Documentación oficial de Android

• http://android-spa.com/ → Foro dedicado a Android

• http://stackoverflow.com/ → Foro de múltiples temáticas

• http://www.edu4java.com/ → Tutoriales muy útiles

• http://es.wikipedia.org/ → Información general



ENGINYERIA TÈCNICA INFORMÀTICA DE SISTEMES

UNIVERSITAT DE BARCELONA

Treball fi de carrera presentat el dia de de 20
a la Facultat de Matemàtiques de la Universitat de Barcelona,
amb el següent tribunal:

Dr. President

Dr. Vocal 1

Dr. Secretari

Amb la qualificació de: