Procesado de imágenes para plataformas móviles

Proyecto de Sistemas Informáticos

2012 / 2013

Facultad de Informática

Universidad Complutense de Madrid

Autores:

Joyce Daniela Aramayo Salgueiro

Alberto de la Piedad Gascueña

Eduardo Sánchez Muñoz

Directores:

Guillermo Botella Juan

José Antonio Martín Hernández

Autorización

Los abajo firmantes, matriculados en la asignatura Sistemas Informáticos de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado durante el curso académico 2012-2013.

Joyce Daniela Aramayo Salgueiro

Alberto de la Piedad Gascueña

Eduardo Sánchez Muñoz

Dedicatoria

A nuestras familias y amigos por su apoyo y cariño infinito.

Agradecimientos

Gracias a José Antonio Martín y Guillermo Botella, tutores de este proyecto, por su colaboración, apoyo y fe.

Gracias a Jorge Romero por la creación del logotipo de la aplicación.

Y un agradecimiento muy especial a todos los amigos que se han prestado pacientemente como modelos, para la creación de nuestra base de datos de imágenes.

Índice

Índice de ilustraciones

Resumen

W2B es una aplicación móvil, desarrollada para ser ejecutada bajo el sistema operativo de código abierto Android, que ofrece al usuario de una manera muy intuitiva la posibilidad de acceder a tiendas online donde comprar prendas de vestir extraídas de una imagen dada. Dicha imagen podrá ser tomada en el acto a través de la cámara o estar almacenada en el dispositivo (Galería). Es decir, a partir de una imagen de entrada que será segmentada y una breve descripción del producto, el usuario obtendrá las tres prendas de ropa que el sistema considere que menos difieren de la original y el enlace a su correspondientes tiendas online donde podrá efectuar la compra, entre las candidatas obtenidas desde Google o de la búsqueda directa en la tienda online de ropa.

El usuario podrá seleccionar entre tres niveles de precisión; alto, medio y bajo, con estos niveles el usuario definirá indirectamente el tiempo de procesamiento, dependiendo de los resultados que quiera obtener. También podrá decidir buscar las imágenes candidatas mediante la API de Google o directamente en la tienda online Zalando,

Palabras clave: Android, Segmentación, Matching, Ropa, Tratamiento de imágenes, Google, OpenCV, Compras

Abstract

W2B is a mobile application, developed under an open source code and to be executed with the Android OS. The W2B application offers the user, the possibility to access online stores websites in a very intuitive manner to buy clothes extracted from any given image. The image is taken on the go by the camera or could already be stored in memory (Gallery).

The user will click (select) one image, that will be segmented, as well as a brief description of the product and the system will provide three alternatives based on the lower grade of differing from the original image and link to its corresponding online website stores among the candidates obtained from Google or from the direct search of the clothes online store website itself.

The user will be able to select between three accuracy levels; low, medium and high, with these levels the user will define indirectly the processing time, depending on the results he/she wants to obtain. The user will also be able to decide between searching the images obtain from Google or directly on the online store website, such as Zalando.

Keywords: Android, Segmentation, Matching, Clothes, Image treatment, Google, OpenCV, Shopping

1. Introducción

1.1 Planteamiento del problema

Actualmente el mundo de la tecnología móvil se encuentra en pleno desarrollo, muchas son las empresas que luchan por ser el líder del sector y tener el mejor dispositivo. Existe una gran variedad de marcas (Samsung, Nokia, Apple, Sony…) y sistemas operativos (Android, IOS, Windows Phone…) por lo que la competitividad resulta evidente. En la mayoría de los casos la marca va ligada al sistema operativo y viceversa.

Este proyecto se centra en Android. Este sistema operativo, desarrollado por Google y de código abierto es el que, actualmente, abarca más dispositivos a nivel mundial. También es el que más aplicaciones pone a disposición del usuario.

Desde la aparición del primer móvil en 1983 hasta la actualidad el desarrollo y la mejora en éstos ha sido asombrosa. Actualmente el mercado pone a disposición de los usuarios terminales con cámaras de última generación, conexión de alta velocidad a Internet y una gran capacidad de procesamiento.

Todos estos avances son utilizados por las aplicaciones. El volumen de aplicaciones disponibles para los distintos sistemas operativos es enorme. Cada uno de estos sistemas pone las aplicaciones al servicio de los usuarios a través de un portal que han creado como si de un 'mercado' se tratara. IOS a través de Itunes Stores, Android con Play Store y Windows Phone Store de Windows Phone. Conteniendo aplicaciones gratuitas y de pago. Muchas de las gratuitas venden publicidad, dando la posibilidad al usuario de eliminarla mediante el pago de una cantidad o también para acceder a contenido exclusivo o extra.

Valiéndose de esta tecnología los portales de venta online han sido testigos del incremento de hasta un 200% en ventas en los últimos años, debido en gran parte a la comodidad que supone el uso de dispositivos móviles a la hora de acceder al gran abanico de productos ofrecidos en red, y aunque en principio la gran parte de adquisiciones pertenecían al campo de la electrónica la moda va ganando terreno en este tipo de transacciones.

A pesar de existir una gran cantidad de aplicaciones, cabe destacar el vacío en cuanto a aplicaciones de búsqueda de prendas de vestir a partir de imágenes. Existen aplicaciones capaces de buscar objetos a través de la cámara como Google Goggles pero no está dirigida a ropa. Es por esta razón que surge la necesidad de desarrollar la aplicación de este proyecto llenando el vacío anteriormente comentado y tendiendo puentes entre algo tan cotidiano como las prendas de ropa y las aplicaciones de última generación.

Las bibliotecas de visión artificial forman la base de este proyecto, puesto que brindan la posibilidad de realizar el análisis y clasificación de las imágenes como se verá más adelante. Una de las más potentes es OpenCV, que incluye soporte específico para Android.

A la hora de encontrar una base de datos de imágenes en Internet, automáticamente el primer pensamiento es Google. Además esta compañía pone a disposición de los desarrolladores de manera gratuita una API de fácil acceso y con soporte Android.

1.2 Objetivos y alcance del proyecto

Por todas las ideas y tecnologías expuestas anteriormente, se quiere desarrollar esta aplicación para llenar el vacío de unión entre moda y tecnología en cuanto a la búsqueda de imágenes. Se propone crear una aplicación que permita al usuario utilizar su dispositivo Android para localizar un punto de venta online que disponga de la misma prenda de vestir o similar a la que contenga la imagen dada a través de la cámara o galería de su Smartphone.

La aplicación será capaz de buscar una prenda por ejecución, pudiendo contener cada fotografía más de una prenda. Es decir, en una imagen que contenga una sudadera y unos pantalones se realizará una búsqueda para la sudadera y otra para los pantalones, las dos de manera independiente y con su propia descripción. Obteniendo tiendas online para cada prenda, no para el conjunto completo. El usuario podrá seleccionar el modo deseado entre tres posibilidades. La primera opción será la más rápida, asumiendo una probabilidad de pérdida de similitud en los resultados obtenidos. La tercera opción será la más precisa de todas, pero también la que mayor procesamiento y tiempo invierta. La opción intermedia es la más equilibrada en cuanto a precisión y tiempo. De esta manera se brinda la opción de abrir y cerrar el espectro de posibilidades teniendo en cuenta el objetivo que se quiera conseguir.

Las prendas de vestir para las que este proyecto está diseñado son camisetas y camisas tanto de manga corta como larga, pantalones largos y cortos, vestidos y chalecos. Para hombres, mujeres y niños. Quedan fuera del ámbito del proyecto zapatos, zapatillas y complementos así como ropa interior y ropa de baño de mujer. Pudiendo ser ejecutados en la aplicación pero con resultados inciertos.

Toda la funcionalidad de la aplicación es gratuita, y tampoco incluirá publicidad para evitar ensombrecer la interfaz gráfica. Ésta será muy cuidada e intuitiva para facilitar el trabajo al usuario, validando cada una de ellas para que la experiencia al usar la aplicación sea lo más gratificante posible.

1.3 Planteamiento de trabajo

Para llevar a cabo el proyecto se van a dividir las acciones en tres partes diferentes pero relacionadas. Segmentación, búsqueda de imágenes en la red y Matching.

En primer lugar se tomará una imagen de entrada. Por cámara o galería como ya hemos visto. Mediante una herramienta de selección táctil en forma rectangular o cuadrada se escogerá la prenda que se desee buscar y se procederá a su segmentación para aislar el objeto del fondo y conseguir los mejores resultados.

El usuario introducirá por pantalla una pequeña descripción del objeto, como el tipo de prenda, el color, si es para hombre o mujer. A través de estos datos se realizará una búsqueda

en la API de Google o en la web de Zalando (1) obteniendo las imágenes candidatas que posteriormente serán comparadas con la imagen de entrada ya segmentada, habiendo ésta satisfecho los mínimos establecidos de calidad en la segmentación.

Por último se procederá al Matching entre la imagen de entrada segmentada y las imágenes candidatas obtenidas. Cada imagen obtendrá un valor que representa el grado de error respecto a la imagen de entrada, cuanto menor sea la puntuación mayor la similitud encontrada por el sistema. El usuario obtendrá las tres imágenes más similares, pudiendo acceder a su página Web e incluso adquirir el producto.

2. Estado del arte

2.1 Sistemas Operativos para dispositivos móviles

Este proyecto se ha realizado bajo el sistema operativo Android. Éste se expondrá en profundidad más adelante, pero antes se va a hacer un pequeño recorrido de las alternativas actuales a este sistema operativo.

2.1.1 iOS

Pertenece a la empresa Apple Inc. Originalmente desarrollado para el iPhone (iPhone OS), siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV. Apple, Inc. no permite la instalación de iOS en hardware de terceros. En mayo de 2010 en los Estados Unidos, tenía el 59% de consumo de datos móviles (incluyendo el iPod Touch y el iPad).

La interfaz de usuario está basada en el concepto de manipulación directa, usando gestos multitáctiles. Los elementos de control consisten en deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee de una interfaz fluida. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan al sacudir el dispositivo (por ejemplo, para el comando deshacer) o rotarlo en tres dimensiones (un resultado común es cambiar de modo vertical al apaisado u horizontal).

iOS se deriva de Mac OS X, que a su vez está basado en Darwin BSD, y por lo tanto es un sistema operativo Unix. Cuenta con cuatro capas de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Principales", la capa de "Medios" y la capa de "Cocoa Touch". La versión actual del sistema operativo es iOS 6.1.3.

Para poder desarrollar en iOS, con ánimo de lucro, es necesario acogerse al programa de desarrolladores con un coste anual de noventa y nueve dólares. Ofreciendo documentación técnica, herramientas de desarrollo, instalar aplicaciones en nuestros dispositivos registrándolos previamente en la Web de Apple y publicar en la AppStore aplicaciones propias y comercializarlas con unos beneficios del 70% sobre el precio de venta establecido.

Figura 1: Ejemplo interfaz iPhone

2.1.2 Windows Phone

Desarrollado por Microsoft, como sucesor de la plataforma Windows Mobile. A diferencia de su predecesor, está enfocado en el mercado de consumo generalista en lugar del mercado empresarial por lo que carece de muchas funcionalidades que proporcionaba la versión anterior. Microsoft ha decidido no hacer compatible Windows Phone con Windows Mobile por lo que las aplicaciones existentes no funcionan en Windows Phone haciendo necesario desarrollar nuevas aplicaciones. Con Windows Phone, Microsoft ofrece una nueva interfaz de usuario que integra varios servicios en el sistema operativo. Microsoft planeaba un estricto control del hardware que implementaría el sistema operativo, para evitar la fragmentación con la evolución del sistema, pero han reducido los requisitos de hardware de tal forma que puede que eso no sea posible.

Con la llegada de Windows Phone 8 (solo para nuevos dispositivos) se cambió por completo el kernel, lo que lo hace incompatible con dispositivos basados en la versión anterior. Esta versión incluye nuevas funciones que de acuerdo a Microsoft lo harán competitivo con sistemas operativos como iOS de Apple o Android de Google. Con esta versión comienza la fragmentación de Windows Phone ya que los dispositivos basados en Windows Phone 7 no pueden actualizarse a Windows Phone 8.

Cuenta con una nueva interfaz de usuario, llamada Metro. La pantalla de inicio, llamada "Start Screen", se compone de "Live Tiles", mosaicos dinámicos que son enlaces a aplicaciones, características, funciones y objetos individuales (como contactos, páginas web, o archivos multimedia) y que muestran información útil y personalizada para el usuario. Estos mosaicos se actualizan frecuentemente manteniendo informado de cualquier cambio al

usuario. La información que se muestra en los mosaicos dinámicos puede ser desde llamadas, mensajes recibidos, correos electrónicos pendientes, citas previstas, juegos o enlaces rápidos a aplicaciones. La pantalla de inicio y la posición del los mosaicos dinámicos se puede personalizar pulsando y arrastrando los mosaicos a la posición que se desee. La interfaz por defecto tiene un estilo visual negro que prolonga la batería en pantallas OLED ya que los píxeles negros no emiten tanta luz como otros, por lo tanto no consumen tanta energía.

Figura 2: Ejemplo interfaz Windows Phone

2.1.1 Android

Es el sistema operativo de Google (2) para dispositivos móviles, llego al mercado en 2008 y está basado en una versión modificada del kernel de Linux 2.6. Éste es utilizado para controlar servicios del núcleo del sistema como pueden ser la seguridad, gestión de memoria o gestión de procesos.

Es una plataforma de código abierto distribuida bajo la licencia Apache 2.0 por lo que su distribución es libre y posibilita el acceso y modificación de su código fuente. Inicialmente fue desarrollado por Google, para más tarde unirse a la Open Handset Alliance (de la cual, Google también forma parte) que está integrada por T-Mobile, Intel, Samsung, HTC o Nvidia entre otros. Sin embargo, Google ha sido la compañía que ha publicado la mayor parte del código fuente.

Para el desarrollo software en esta plataforma se dispone de manera gratuita de un SDK y un plugin que se integra dentro del entorno de desarrollo de Eclipse, donde se incluyen todas las APIs necesarias además de un potente emulador integrado para facilitar las pruebas de la aplicación. Google intenta con Android aunar todos los elementos necesarios para que los desarrolladores tengan acceso a todas y cada una de las funciones que ofrece un dispositivo

móvil de manera sencilla, es decir, se quiere estandarizar el desarrollo de aplicaciones para dispositivos móviles.

Figura 3: Ejemplo interfaz Android

A Historia

En octubre de 2008 se presentaba el teléfono G1 con la versión 1.0 del sistema operativo. Esta versión presentaba algunos errores, sobre todo, en comparación con su principal rival el iPhone de Apple. No obstante la plataforma comenzó a crecer a buen ritmo. Muchos fabricantes presentaron sus nuevos terminales con Android, llegando a convertirse en la opción principal de los usuarios sobre todo con las últimas versiones. A continuación se muestra de modo gráfico la evolución del sistema operativo junto con las versiones.

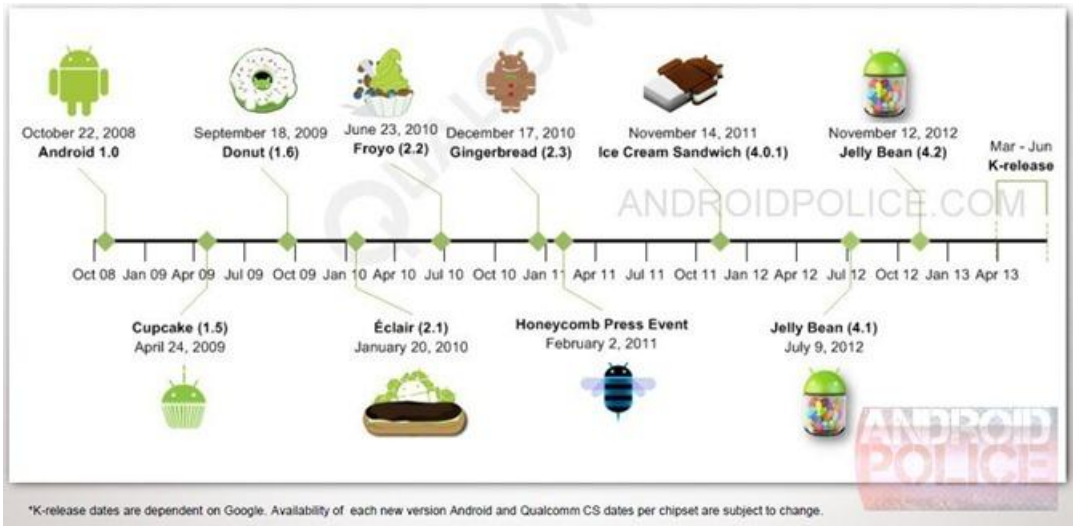


Figura 4: Línea del tiempo Android. Fuente: Android Police (3)

B Arquitectura

Android es un sistema diseñado por capas, utiliza el kernel de Linux 2.6, como ya se vio anteriormente, que le da acceso a la parte hardware de los dispositivos- Esta arquitectura se puede contemplar en la siguiente figura.

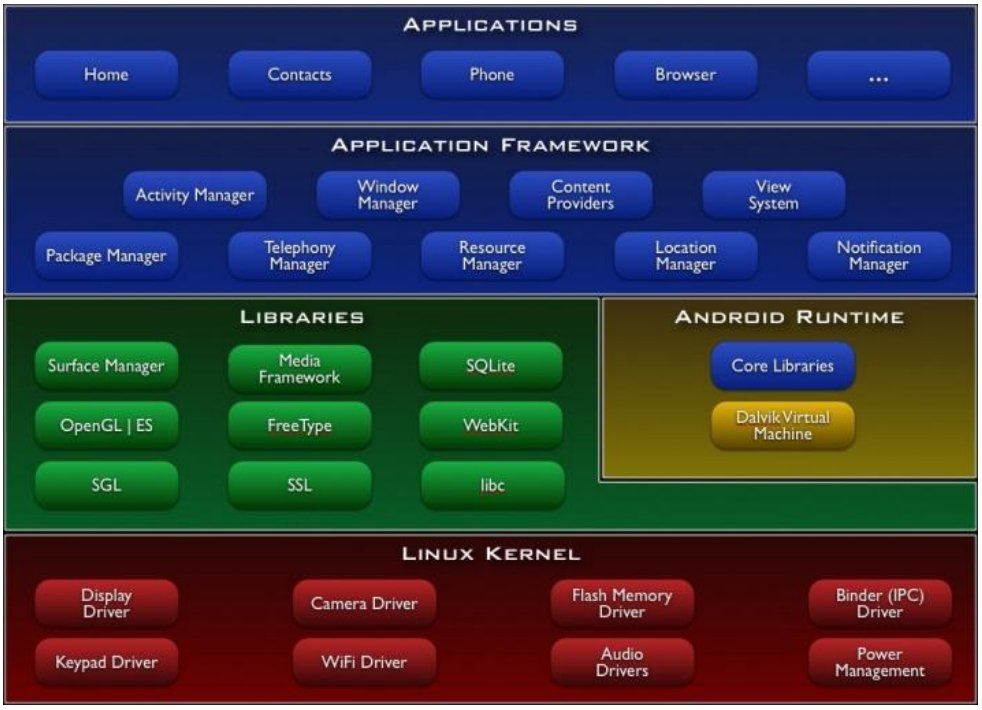


Figura 5: Arquitectura Android

C Aplicaciones

Están desarrolladas en Java y pueden estar formadas por los siguientes bloques: Activity, Intent, Broadcast, Services y Content Providers. No tienen porque aparecer todos en todas las aplicaciones.

D Framework

El marco de aplicaciones da acceso completo a las APIs. La arquitectura está diseñada para que la reutilización de componentes sea sencilla, ya que cualquier aplicación puede dar un perfil público a sus datos o capacidades para que otra aplicación haga uso de esas cualidades.

E Bibliotecas

El sistema incluye un conjunto de bibliotecas en C y C++ que proporcionan la mayor parte de las funcionalidades presentes y que son utilizadas por varios de los componentes del sistema Android. Estas capacidades son expuestas a los desarrolladores a través del framework descrito anteriormente para que puedan ser utilizadas.

F Android runtime

Al mismo nivel que las bibliotecas de Android, se sitúa el Android runtime o entorno de ejecución, compuesto por bibliotecas que proporcionan la mayor parte de las funcionalidades de Java además de incluir la máquina virtual. Cada aplicación, al ejecutarse crea su propio proceso con su propia instancia de la máquina virtual Dalvik, que ha sido escrita de manera que un mismo dispositivo pueda lanzar múltiples instancias de dicha máquina virtual de forma eficiente.

G Kernel de Linux

La capa más cercana al hardware corresponde al núcleo de Android que está basado en el kernel de Linux 2.6. Android utiliza este kernel como una abstracción del hardware disponible en cada terminal conteniendo los drivers necesarios para utilizar cualquier parte de este hardware a través de las llamadas correspondientes. Android también depende de este kernel para gestionar los servicios base del sistema como pueden ser la seguridad, gestión de memoria, de procesos, la pila de red y el modelo de driver.

H Ciclo de vida de una aplicación

Una aplicación Android se ejecuta en su propio proceso Linux. Este proceso es creado cuando parte del código necesita ser ejecutado, y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación. Una característica importante y poco usual de Android es que el tiempo de vida de un proceso no es controlado directamente por la aplicación, sino que es el sistema quien determina el tiempo de vida basado en el conocimiento que tiene de las partes de la aplicación, la importancia de la misma y cuánta memoria hay disponible. Por lo tanto, el usuario no tiene conocimiento sobre estas áreas ya que según la naturaleza de Android son tareas del sistema operativo.

I Ciclo vida de un activity

Android maneja las actividades como una pila, cuando una nueva actividad es creada, se coloca en lo más alto de la pila y relega a la actividad anterior a permanecer justo debajo de ella en la propia pila. En la siguiente tabla se muestran en detalle los posibles estados de una actividad junto con su descripción.

Estado	Descripción	Abortable
onCreate	Se ejecuta cuando se crea el activity por primera vez. La inicialización de la aplicación.	No
onRestart	Se ejecuta cuando la aplicación se ha cerrado y se va a ejecutar nuevamente.	No
onStart	Se ejecuta cuando la aplicación aparece visible para el usuario.	No
onResume	Se ejecuta cuando el usuario interactúa con la aplicación.	No
onPause	Se ejecuta cuando el sistema va a continuar con un activity anterior.	Si
onStop	Se ejecuta cuando el activity deja de ser visible al usuario.	Si
onDestroy	Se ejecuta antes de destruir el activity.	Si

Figura 6: Tabla estados de una actividad

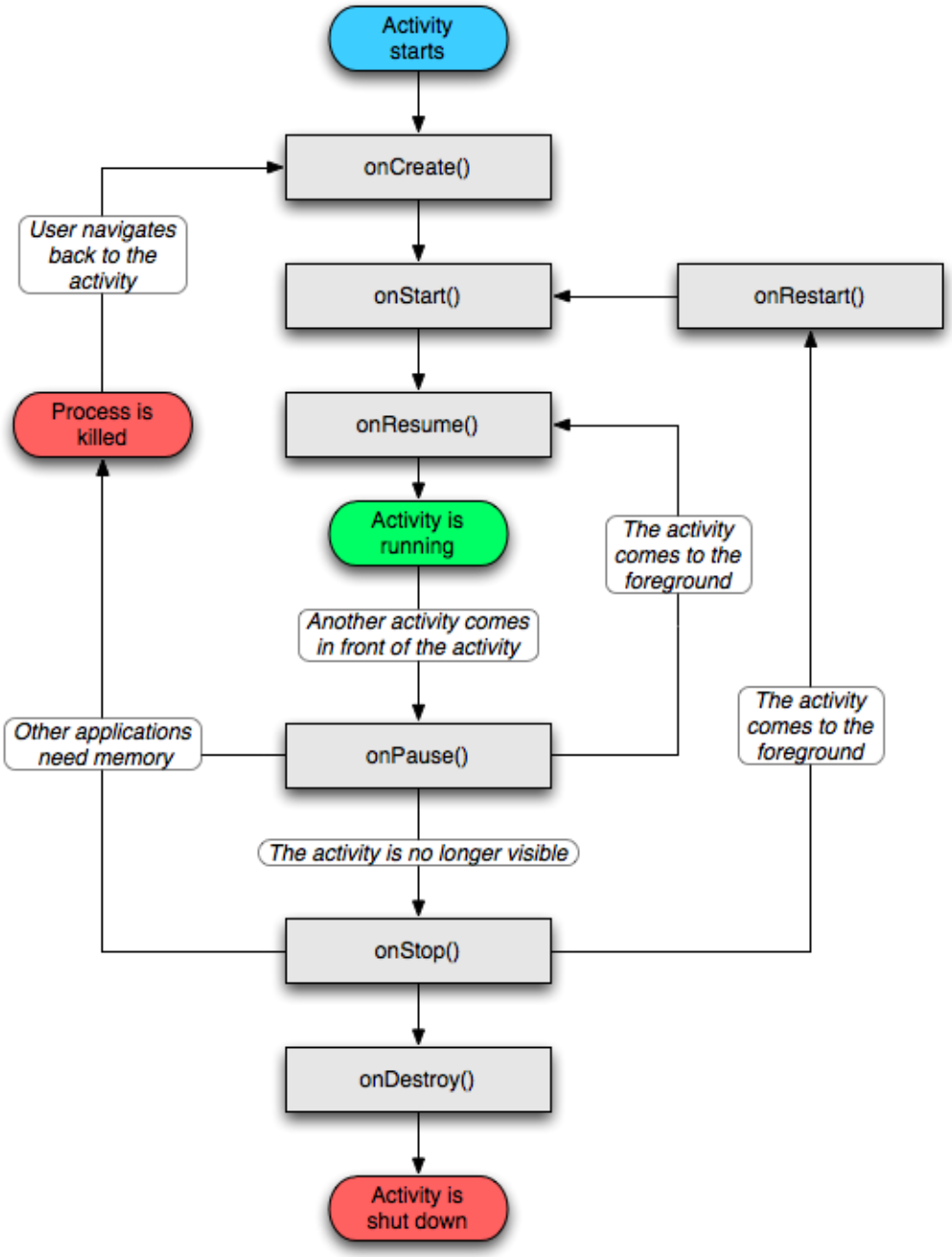


Figura 7: Relación estados en Android

En la figura anterior se puede apreciar la relación entre los estados a la hora de implementar la aplicación.

2.2 Aplicaciones similares

2.2.1 Google Goggles

Es una aplicación (4) de reconocimiento de imágenes para dispositivos móviles creada por Google, disponible para Android, IOS, BlackBerry y Nokia. Es bastante conocida ya que cuenta con más de diez millones de descargas en el Play Store de Google.

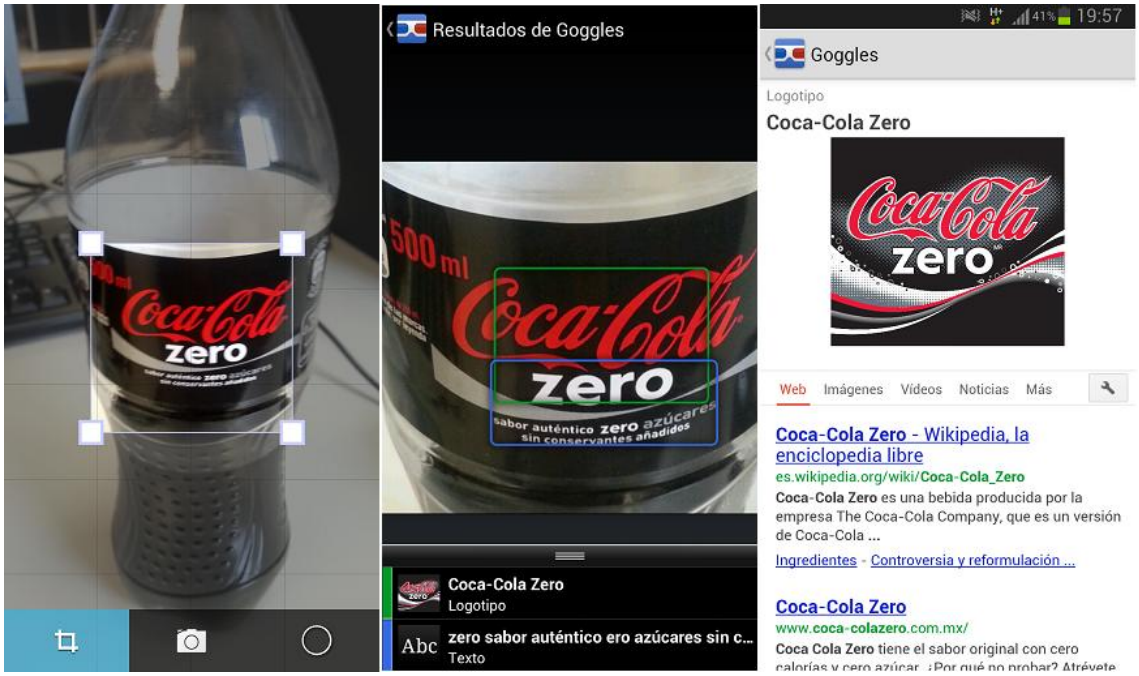


Figura 8: Captura aplicación Google Goggles

Sus principales funcionalidades son reconocer cuadros, lugares famosos, códigos de barras, códigos QR, un logotipo o una imagen popular. En caso de encontrar la imagen en su base de datos ofrece información útil sobre ella buscándolo en el propio buscador de Google. Otras funciones que ofrece la aplicación es traducir texto a otro idioma previamente capturado con la cámara o resolver sudokus.

En la figura 8 se observa el funcionamiento de esta aplicación al tomar una foto a una botella de Coca-Cola Zero en la que reconoce el logotipo de dicha marca y el texto de la parte inferior y nos ofrece la información del buscador de Google.

La aplicación obtiene unos buenos resultados bastante precisos para los casos anteriormente descritos, pero tiene poca utilidad para encontrar objetos en los que no se vea ningún tipo de marca o logotipo.

2.2.2 Kooaba

Es una plataforma (5) de reconocimiento de imágenes diseñada para integración en aplicaciones desarrollada en ETH Zurich, uno de los centros más innovadores en visión por

computador. Da la funcionalidad de realizar las búsquedas en imágenes propias o en su base de datos. Una vez encontradas devuelve información adicional sobre ellas como por ejemplo, páginas web relacionadas, descripción, tags, etc.

Ofrece diferentes servicios de precios según los requerimientos concretos de cada aplicación, desde la versión gratuita con peticiones limitadas hasta la versión más cara de 1500 euros que ofrece poder subir hasta 100.000 imágenes.

Una de sus principales ventajas es su gran escalabilidad debido a que se ejecuta en los servicios web de Amazon. Su plataforma de reconocimiento visual se utiliza tanto como para mejorar sistemas de visión por computador de grandes empresas como de desarrolladores individuales. Una selección de aplicaciones interesantes que hacen uso de la plataforma de Kooaba es ShortCut Reader:

ShortCut Reader es un acceso directo entre la vida real e internet. Esta aplicación permite al usuario tomar una foto de lo que está leyendo en periódicos y revistas de varios países e inmediatamente conectarlo con su versión digital. También funciona para anuncios y vallas publicitarias.

Figura 9: Captura de ShortCut Reader

En la figura 9 se observa el funcionamiento de la aplicación fotografiando un artículo de un periódico y como nos muestra su versión online.

2.2.3 Vivino

Es una aplicación móvil para iOS y Android que utiliza las tecnología de visión por computador de Kooaba para reconocimiento de botellas de vino escaneando su etiqueta. En cierto modo es una red social de vinos donde cada usuario que hace una foto a su botella puede dar su puntuación y su opinión sobre el vino pudiendo ver que han dicho los demás usuarios también.

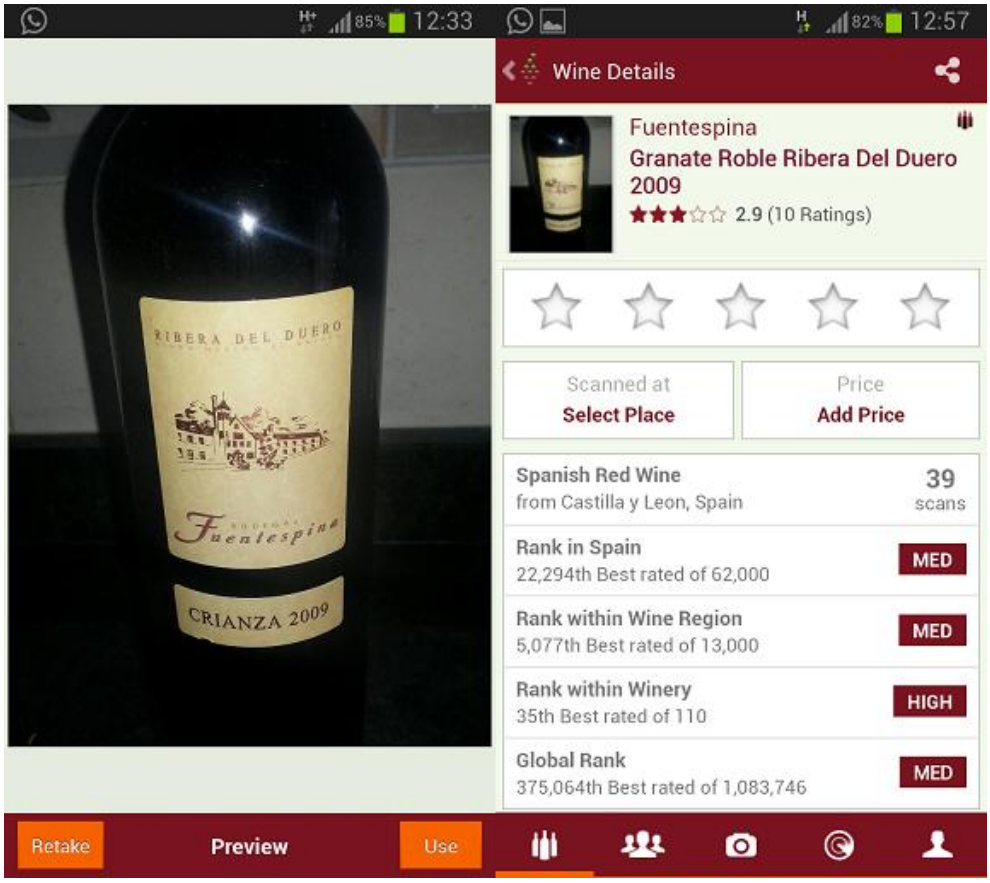


Figura 10: Captura Vivino

En el ejemplo de uso de la aplicación en la figura 10 se toma una foto de una botella y una vez analizada la aplicación muestra su nombre completo con la puntuación de los demás usuarios y también el ranking que ocupa esta botella entre todas las demás que tiene catalogadas.

2.2.4 Clipping Magic

ClippingMagic.com es un sitio web dedicado a quitar el fondo de imágenes. Actualmente en versión Alfa esta herramienta de editado de fotos intenta conseguir lo que generalmente es un proceso complicado (aislar el sujeto de una imagen) en uno simple, lo hace permitiendo al usuario alternar dos marcadores sobre la imagen, uno verde y otro rojo, que darán información al sistema acerca de cuáles son las áreas consideradas como fondo y cuáles no. Generalmente las áreas a marcar no necesitan ser demasiado grandes para conseguir un buen

resultado, sin embargo esto depende de la imagen a tratar, ya que el algoritmo utilizado por la herramienta necesita bordes visibles a nivel de pixel, es decir que bordes poco definidos(como pieles) o transparencias llevarán a un mal resultado.

El uso de la herramienta es sencillo, el usuario carga la imagen a tratar arrastrándola o eligiéndola desde el navegador de archivos, a continuación marcará las zonas que desee aislar (marcador verde) o desechar (marcador rojo), no es necesario ser demasiado preciso en este marcado.

Figura 11: Método Clipping Magic

Haciendo uso de estas líneas guía la herramienta aislará el objeto deseado y enseñará los resultados en tiempo real gracias a una pantalla dividida, el usuario puede añadir o quitar marcas hasta quedar satisfecho con el resultado.
Por último la herramienta brinda al usuario la opción de descargar la imagen final con formato PNG, al igual que ofrece un link para su compartición.

A continuación se muestran algunos ejemplos de la ejecución de la herramienta.

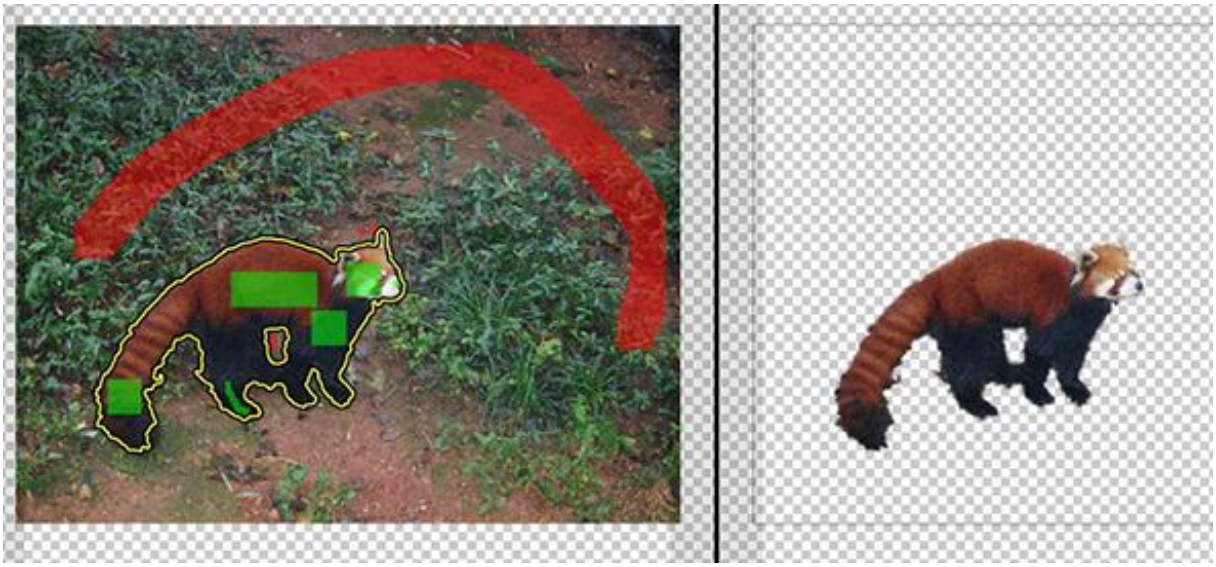


Figura 12: Ejemplo bordes Clipping Magic

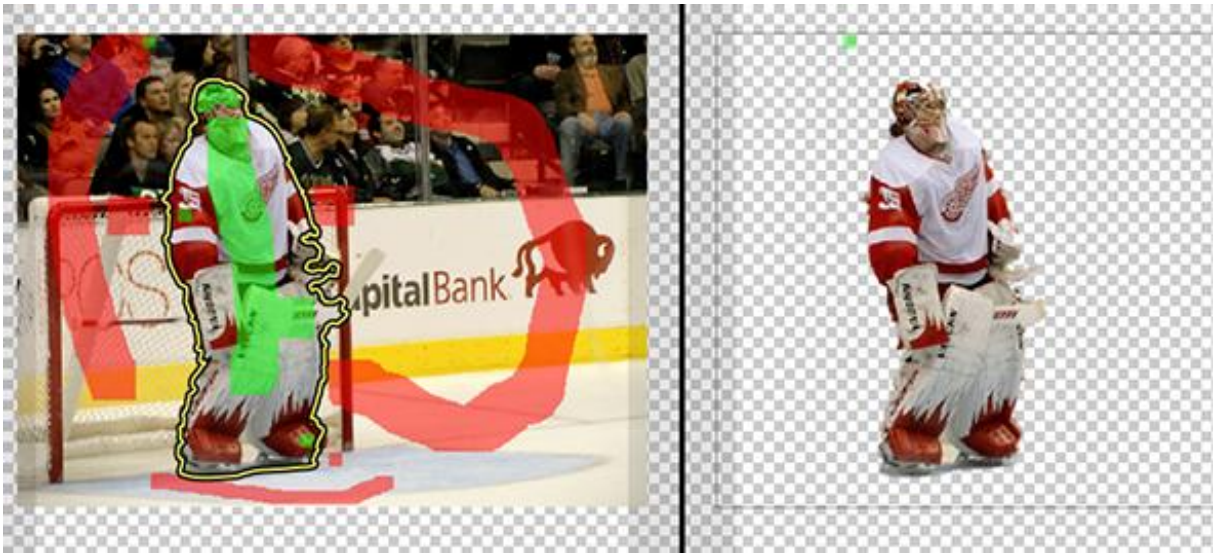


Figura 13: Ejemplo sujeto Clipping Magic

2.2.5 Tineye

Se trata de un motor de búsqueda (6) inverso, al que se le puede enviar una imagen y devolverá de dónde viene esa imagen, cómo se usa, si existen versiones modificadas de la imagen y también para encontrar versiones con mayor resolución.

Es el único motor de búsqueda que utiliza tecnologías de identificación de imágenes en lugar de palabras clave, metadatos o marcas de agua.

Tineye posee una amplia base datos con un total de 2,636,427,301 imágenes donde realiza las búsquedas y comparaciones. Esta aplicación es libre de usarse para cualquier búsqueda no

comercial, pero ofrece servicios para usarse como apoyo para otra aplicación en forma de API mediante pagos por búsqueda.

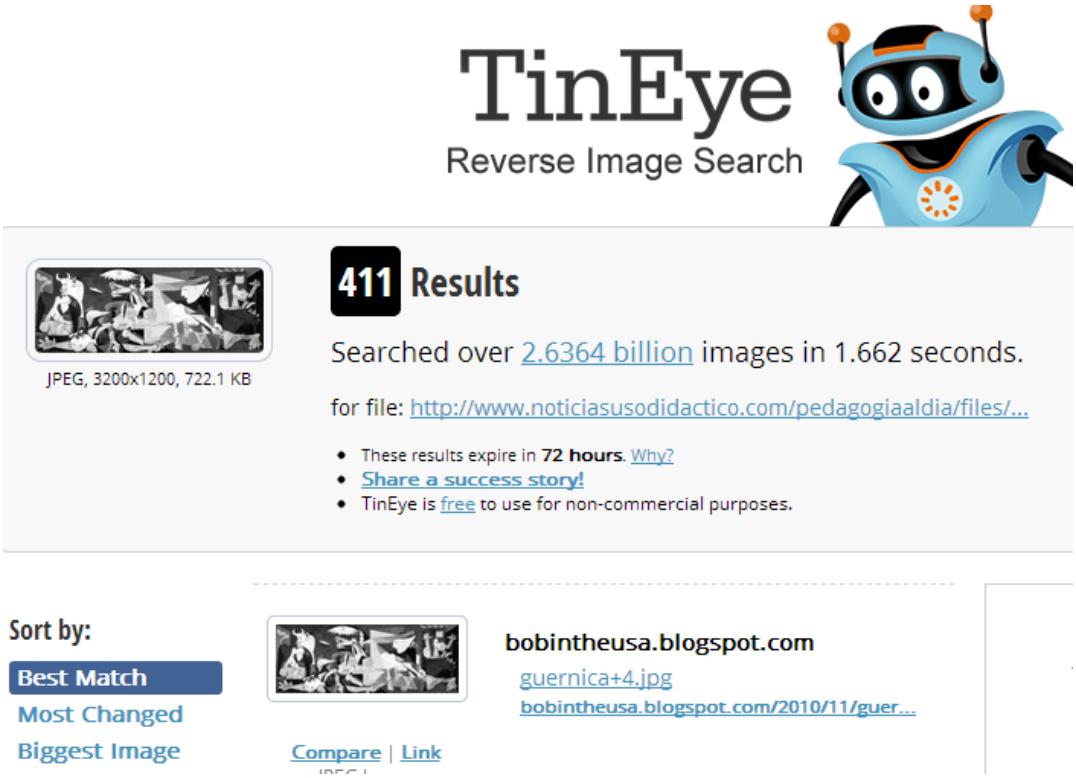


Figura 14: Ejemplo de cuadro en TinEye

En la figura 14 se puede observar el resultado de buscar el cuadro El Guernica en el buscador TinEye.

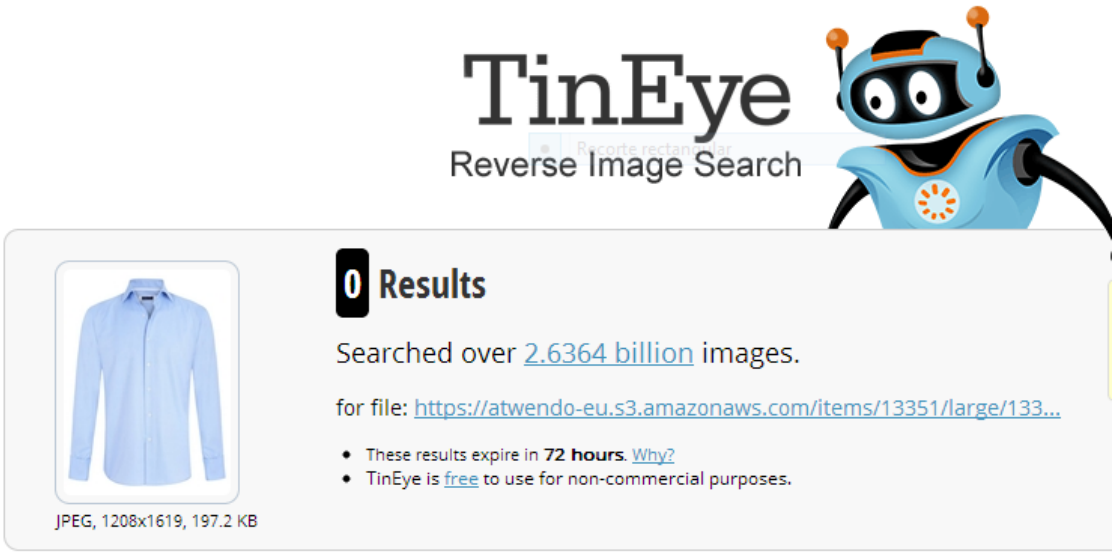


Figura 15: Ejemplo de camisa en TinEye

En la figura 15 se puede apreciar el resultado de buscar una camisa en el buscador TinEye.

En la práctica este buscador sólo obtiene buenos resultados en el caso de introducir alguna imagen bastante conocida, en caso contrario es muy difícil que devuelva algo. Este hecho se debe a que no permite al usuario introducir algún tipo de metadatos para poder filtrar la búsqueda.

Como por ejemplo en los casos mostrados anteriormente en los que para una imagen mundialmente conocida como el cuadro de El Guernica en la figura 14 la aplicación ofrece un total de 411 resultados que básicamente consisten en diferentes imágenes del cuadro con resoluciones distintas. Por otro lado en la figura 15 tenemos la búsqueda de una camisa en la que este buscador no consigue encontrar ningún resultado.

2.3 Bibliotecas para el procesamiento de imágenes

2.3.1 OpenCV

Es una biblioteca (7) libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

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

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

2.3.2 JavaCV

Es una interfaz para usar OpenCV bajo Java. De hecho comparte la misma documentación que OpenCV. Ha quedado relegada a un segundo plano con la salida de la última versión de OpenCV, la 2.4.4, que soporta Java.

2.3.3 Jon's Java Imaging Library (JJIL)

Es una biblioteca para el procesamiento de imágenes en Java. Incluye una arquitectura de procesamiento de imágenes y más de 60 rutinas para diversas tareas de procesamiento de imágenes. Está especialmente dirigida a aplicaciones móviles. Es eficiente tanto espacio como en tiempo sobre teléfonos móviles. Incluye interfaces para que las imágenes se puedan

convertir hacia y desde formatos nativos para J2ME, Android, y J2SE. La biblioteca está disponible como software libre bajo licencia GPL.

La biblioteca elegida para realizar este proyecto ha sido OpenCV, por ser código abierto, contar con funciones que cubren las necesidades del alcance del problema, una documentación muy amplia y un desarrollo con la plataforma Android muy completo.

2.4 Lenguajes de programación y entornos de trabajo

2.4.1 Android-iOS

Para el desarrollo de una aplicación móvil existen dos grandes sistemas operativos entre los que elegir: Android OS e iPhone OS. El primero utiliza principalmente como lenguaje de programación Java, al ser este uno de los lenguajes más utilizados, el desarrollo de aplicaciones para esta plataforma es más natural para gran parte de programadores. En contraposición iPhone OS utiliza como lenguaje de desarrollo Objective-C, que generalmente puede ser desentrañado por desarrolladores que estén familiarizados con C y C++.

En cuanto a plataformas de desarrollo; Android ofrece a los desarrolladores la posibilidad de utilizar plataformas abiertas y la libertad para utilizar herramientas adyacentes para el desarrollo. Esto permite a los desarrolladores incorporar distintas características a la aplicación, además de más funcionalidades. Por otra parte Apple es bastante restrictiva con sus pautas de desarrollo. El desarrollador en este caso obtiene un conjunto fijo de herramientas para el desarrollo y no puede hacer uso de ninguna otra fuera de este.

Otro dato a tener en consideración es el número de terminales que utilizan uno u otro sistema operativo, la gráfica siguiente, que recoge datos de la Corporación Internacional de Datos, refleja una clara tendencia de los consumidores a adquirir terminales Android sobre iOS.

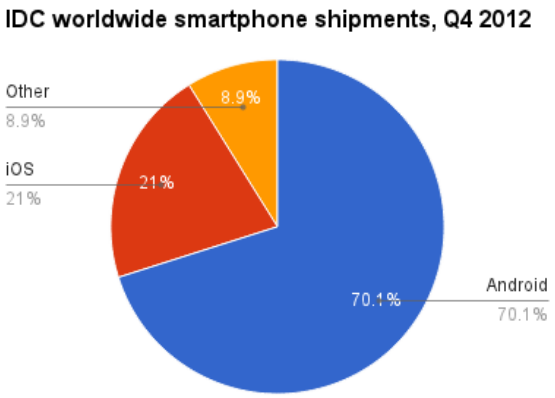


Figura 16: Distribución de Smartphones a nivel mundial

iOS es una plataforma madura, tiene origen en el sistema NeXTStep de los años 90 (de ahí que muchas clases de su API tengan como prefijo 'NS'). Esta plataforma evolucionó más tarde dando lugar al Mac Os, de la cual iOS es una versión minimizada.

Con cada nuevo lanzamiento iOS ofrece nuevas características, y el paso a nuevas versiones es generalmente muy sencillo, ocasionalmente el comportamiento de APIs entre distintas versiones cambia.

La primera versión de Android fue lanzada al mercado en 2007. La apariencia y sensación de esta ha cambiado de forma notable. Los primeros lanzamientos alfa fueron parcialmente inspirados por BlackBerry, aunque lanzamientos posteriores se asemejan más a iOS. Android 3.0 parece haberse inspirado en Windows Metro, por sus bordes afilados, además ha suprimido cuatro botones hardware de navegación y los ha remplazado con una barra de navegación y una barra de acción en la pantalla táctil. Otras características de Android se van viendo mejoradas en versiones nuevas, esto añade carga a los desarrolladores, ya que si desean ofrecer la mejor experiencia de uso para versiones de Android recientes y anteriores han de escribir a menudo códigos separados para cada lanzamiento importante.

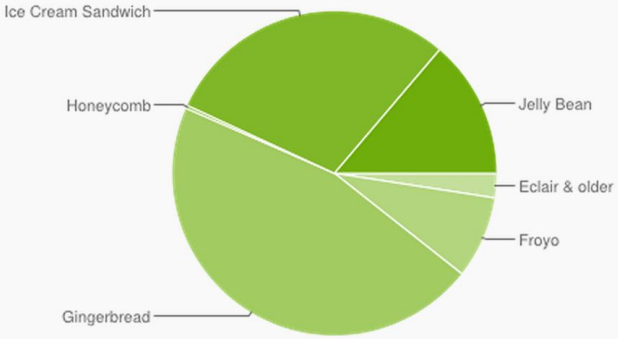
Para desarrollar aplicaciones en ambos sistemas operativos son suficientes un IDE (Integrated Development Enviroment) y un SDK (Software Development Kit).

En el caso de iOS es necesario un Mac para el desarrollo, el IDE por asignado por defecto es Xcode de Apple. Parte de la distribución de Xcode es el SDK de iOS y su simulador. Durante mucho tiempo este fue el único IDE disponible para este sistema operativo, desde Abril de 2011 tiene como competidor AppCode de JetBrains.

El desarrollo en Android puede hacerse tanto en Windows, Mac OS o Linux. La forma más fácil de empezar con el desarrollo es con un ADT (Android Development Tools) que recoge en un solo paquete un IDE(Eclipse), un Plugin de ADT para Eclipse y los componentes esenciales de Android SDK. Si se quiere usar otro Java IDE, como IntelliJ, es necesario solo el paquete de herramientas de Android SDK y utilizarlo para la obtención de paquetes adicionales.

Google publica la información sobre la distribución de las distintas versiones de Android, la gráfica siguiente muestra la distribución de estas versiones del 22 de Enero al 4 de Febrero de 2013.

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.2%
2.2	Froyo	8	8.1%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	45.4%
3.1	Honeycomb	12	0.3%
3.2		13	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.0%
4.1	Jelly Bean	16	12.2%
4.2		17	1.4%



Data collected during a 14-day period ending on February 4, 2013

Figura 17: Distribución versiones Android

Durante este período, más de la mitad de los usuarios de Android aún eran usuarios de una versión 2.x. del sistema operativo, esto se debe a que cada fabricante del dispositivo ha de lanzar su propia actualización de Android, lo cual a veces no ocurre o se da demasiado tarde.

Para desarrolladores de Android esto significa tener que dar soporte a un amplio rango de versiones (además de la gran variedad de dispositivos).

Apple, en contraposición, no publica ninguna información sobre la distribución de las versiones de iOS. Terceras partes sin embargo publican dicha información basándose en diversos datos a su disposición como iOS Version Stats, se presume que alrededor del 80% de los usuarios de dispositivos iOS utilizan alguna de las dos últimas versiones de este sistema operativo.

A diferencia de los desarrolladores Android un desarrollador de iOS cubre cerca del 80% de potenciales usuarios al crear una aplicación para una de las últimas versiones del sistema.

En primer lugar cabe destacar ANDROID NDK (Native Development Kit) un conjunto de herramientas que permite implementar partes de una aplicación usando lenguajes en código nativo como C y C++. Puede ser de ayuda para ciertos tipos de aplicaciones ya que se puede reutilizar códigos de bibliotecas escritas en estos lenguajes, pero la mayor parte de las aplicaciones no se ven beneficiadas por su uso, ya que generalmente no resulta en una mejora notable de rendimiento pero siempre incrementa su complejidad. En general debería usarse sólo si es esencial para la aplicación, nunca porque simplemente se prefiera programar en C/C++.

Típicamente los candidatos buenos para el uso de NDK son aplicaciones que hagan uso intenso de operaciones en CPU que no reserven mucha memoria, como procesamiento de

señales, simulación física, y otros. En vista de sus características y a pesar de que la biblioteca de la que hace uso esta aplicación está desarrollada en C, no se considera pertinente el uso de Android NDK en este proyecto.

Otra opción es hacer uso de SL4A (Scripting Layer for Android), biblioteca lanzada por Android en 2009, que permite la creación y ejecución de varios lenguajes scripting directamente en dispositivos Android.

Estos scripts tienen acceso a muchas de las APIs disponibles para aplicaciones Java Android normales, pero con una interfaz simplificada. Los scripts puedes ejecutarse interactivamente en un terminal, o en segundo plano utilizando la arquitectura de servicios de Android. Actualmente los lenguajes soportados son:

- Python using CPython
- Perl
- Ruby using JRuby
- Lua
- BeanShell
- JavaScript
- Tcl

El lenguaje escogido por la mayor parte de los desarrolladores de aplicaciones Android es Java, por varios motivos. En primer lugar, como se ha mencionado en el apartado anterior, es un lenguaje muy extendido entre los programadores, se calcula que existen cerca de 9 millones de programadores de este lenguaje, lo cual implica su fácil acceso al desarrollo de aplicaciones móviles valiéndose de conocimientos ya adquiridos. Por otra parte las aplicaciones desarrolladas con este lenguaje corren sobre una máquina virtual, descartando la necesidad de recompilar el código para cada terminal y facilitando su seguridad. Existe además un gran número de herramientas de desarrollo para java que ofrecen una mayor libertad a los programadores a la hora de crear nuevas aplicaciones.

2.5 Definiciones de Segmentación

2.5.1 Algoritmo de k-means o k-medias

El objetivo del algoritmo k-means (8) es dividir una población N-dimensional en *k* clusters previamente determinados, donde cada elemento de la población pertenecerá a una de las *k* medias.

Es un algoritmo iterativo que consiste en asignar *k* centroides, uno para cada cluster. Se puede determinar la posición de los centroides o asignarlos de manera aleatoria. OpenCV también da la posibilidad de hacerlo mediante la inicialización de Arthur y Vassilvitskii.

Una vez fijados los centroides $\mu_{i,}\ \forall = 1..k,$ se asocia cada uno de los elementos de la población a uno de ellos en base a la distancia mínima mediante la distancia euclídea,

formándose así k clusters. Una vez fijados, se calcula la media de cada cluster generando k nuevos centroides que se obtienen minimizando el objetivo:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

Donde μ_i es el centroide de cada elemento de la población $x_j \in S_i$

De manera iterativa se repite este proceso hasta que se consigue que ningún centroide se mueva. Esto produce la separación de los elementos en k grupos donde la distancia será mínima.

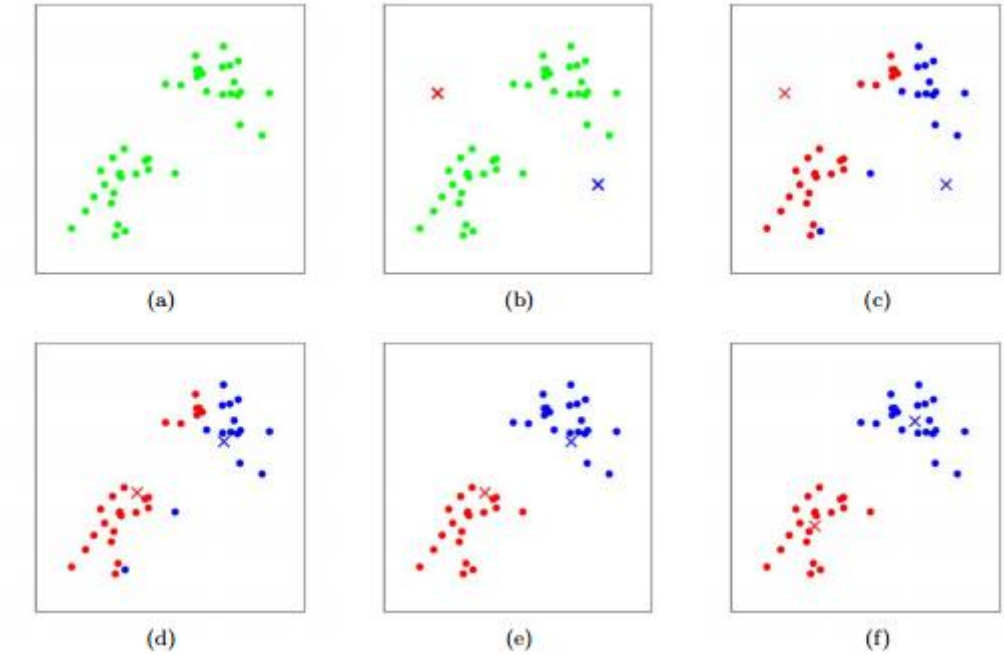


Figura 18: Funcionamiento K-means

En la figura 18 se puede apreciar el funcionamiento del algoritmo de k-means. (a) Conjunto de datos originales. (b) Inicialización aleatoria de 2 centroides. (c-f) Dos iteraciones de la ejecución del algoritmo.

En el ejemplo de la figura 18 tenemos un conjunto inicial de datos que queremos dividir en dos, para ellos se generan dos centroides aleatorios y para cada elemento del conjunto según su cercanía se le asigna uno de los centroides para a continuación calcular la nueva posición de estos y volver a asignarle a cada elemento un centroide. Repitiendo este proceso una vez más queda como se ve al final de la figura.

Este algoritmo es fácil de programar y computacionalmente económico por lo que es viable para aplicaciones con un gran número de muestras y es idóneo para realizar agrupamientos por similitud.

2.5.2 Suavizado de Gauss

Las operaciones de suavizado tienen por objeto reducir el ruido y/o efectos espurios que pueden presentarse en una imagen a consecuencia del proceso de captura, digitalización y transmisión. Existen distintos tipos de algoritmos que permiten la reducción del ruido. En este caso se aplicarán filtros lineales (convolución de una imagen con una máscara predefinida) como suavizado gaussiano. En dicho suavizado los pesos toman la forma de una campana de Gauss (media ponderada). Se aplica en dos dimensiones, se construye la máscara que depende de la distancia al píxel central. En la siguiente imagen se puede observar cómo se consigue el resultado de promediar con distintos pesos los valores vecinos a ambos lados del punto a tratar.

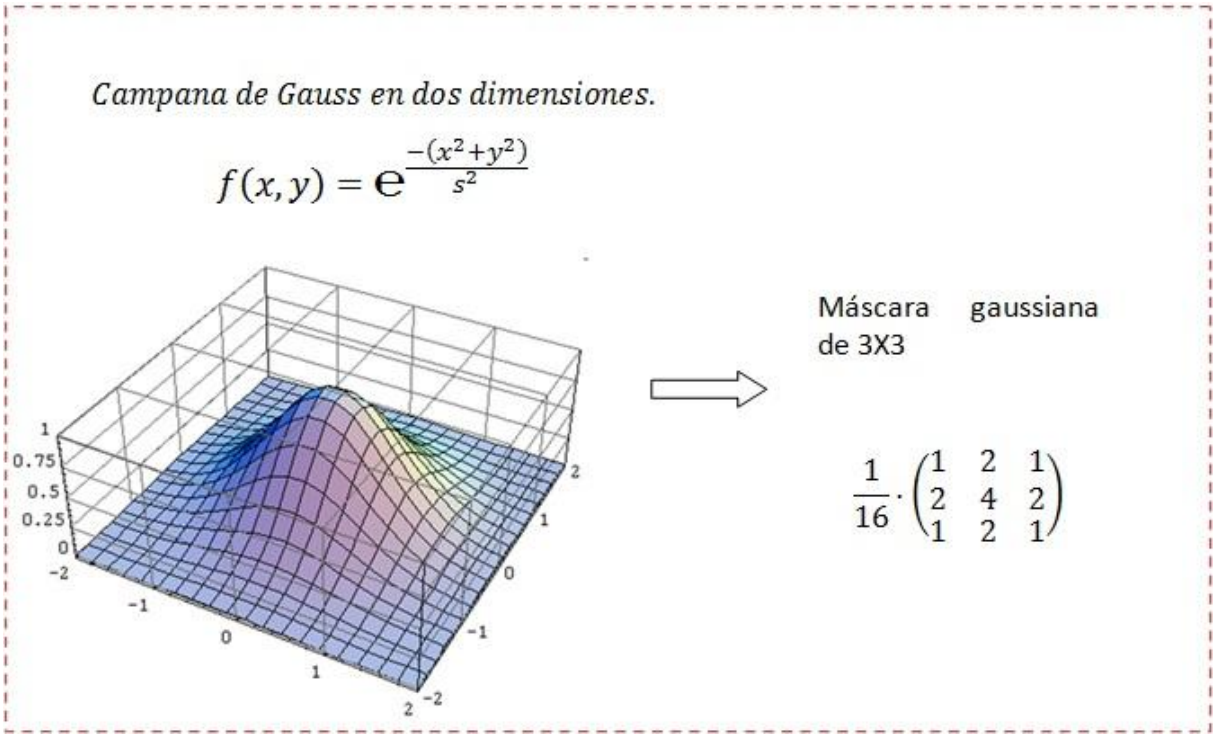


Figura 19: Campana y máscara Gauss

La varianza s^2 indica el nivel de suavizado. Si la varianza es grande, la campana es más ancha y el suavizado mayor. Al contrario que con la varianza más pequeña, puesto que la campana es más estrecha y el suavizado menor.

En la figura 20 se puede apreciar el resultado de la imagen al aplicar un filtro de Gauss.

Figura 20: Ejemplo suavizado Gauss

2.5.3 Componentes RGB

RGB (9) (Red, Green, Blue) es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul, por lo que los mismos valores RGB pueden mostrar colores notablemente diferentes en diferentes dispositivos que usen este modelo de color. Aunque utilicen un mismo modelo de color, sus espacios de color pueden variar considerablemente.

Se puede contemplar el modelo aditivo de colores rojo, verde y azul en la siguiente figura.

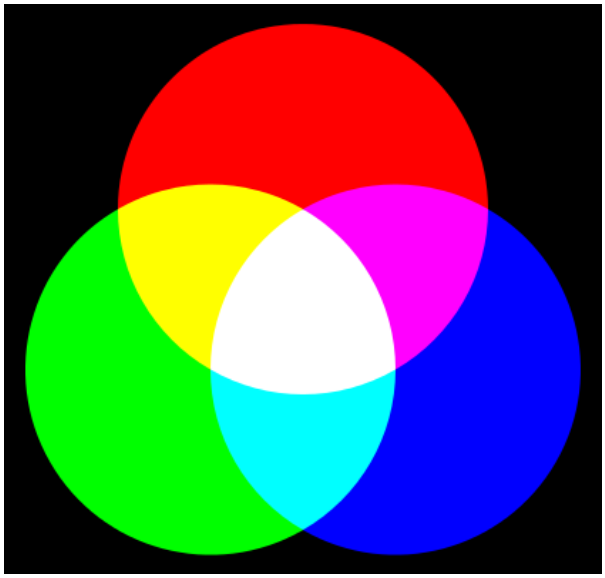


Figura 21: Modelo RGB

Los ojos humanos tienen dos tipos de células sensibles a la luz (fotorreceptores): los bastones y los conos. Estos últimos son los encargados de aportar la información de color.

Para saber cómo es percibido un color, hay que tener en cuenta que existen tres tipos de conos con respuestas de frecuencia diferentes, y que tienen máxima sensibilidad a los colores que forman la terna RGB. Aunque que los conos, que reciben información del verde y el rojo, tienen una curva de sensibilidad similar, la respuesta al color azul es una veinteava (1/20) parte de la respuesta a los otros dos colores. Este hecho lo aprovechan algunos sistemas de codificación de imagen y vídeo, como el JPEG o el MPEG, "perdiendo" de manera consciente más información de la componente azul, ya que el ser humano no percibe esta pérdida.

La sensación de color se puede definir como la respuesta de cada una de las curvas de sensibilidad al espectro radiado por el objeto observado. De esta manera, obtenemos tres respuestas diferentes, una por cada color.

El hecho de que la sensación de color se obtenga de este modo, hace que dos objetos observados, radiando un espectro diferente, puedan producir la misma sensación. Y en esta limitación de la visión humana se basa el modelo de síntesis del color, mediante el cual podemos obtener a partir de estímulos visuales estudiados y con una mezcla de los tres colores primarios, el color de un objeto con un espectro determinado.

2.5.4 Distancia Euclídea

En matemáticas, (10) la distancia euclídea o euclidiana es la distancia "ordinaria" (que se mediría con una regla de acero) entre dos puntos de un espacio euclídeo, la cual se deduce a partir del teorema de Pitágoras.

Por ejemplo, en un espacio bidimensional, la distancia euclidiana entre dos puntos P1 y P2, de coordenadas (x1, y1) y (x2, y2) respectivamente, es:

$$d_E(P_1, P_2) = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

2.5.5 Filtro de Gabor

El filtro de Gabor (11) es un filtro lineal cuya respuesta de impulso es una función sinusoidal multiplicada por una función gaussiana. Son funciones casi paso banda.

La principal ventaja que se obtiene al introducir la envolvente gaussiana es que las funciones de Gabor están localizadas tanto en el dominio espacial como en el de la frecuencia, a diferencia de lo que ocurre con las funciones sinusoidales, que están perfectamente localizadas en el dominio frecuencial y completamente deslocalizadas en el espacial (las funciones sinusoidales cubren todo el espacio). Por tanto, son funciones más adecuadas para representar una señal conjuntamente en ambos dominios. Ésta es la base que llevó a Gabor a introducirlas en 1946.

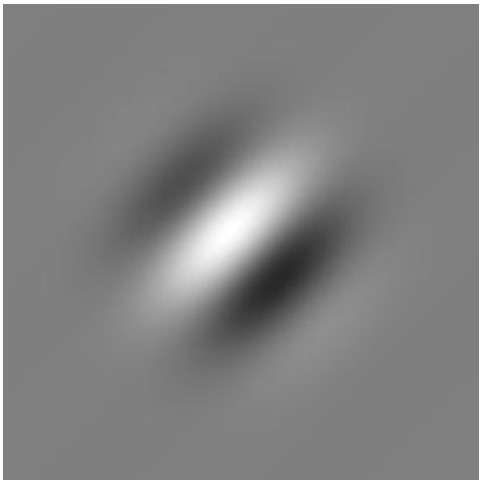


Figura 22: Ejemplo Gabor

En la figura 22 se aprecia parte real de la respuesta de impulso de un filtro de Gabor bidimensional.

La transformada de Fourier de un filtro de Gabor son gaussianas centradas en la frecuencia de la función sinusoidal (siendo estas gaussianas la transformada de Fourier de la gaussiana temporal o espacial). Se puede llegar a este resultado empleando la propiedad de convolución de la Transformada de Fourier, que transforma los productos en convoluciones. Así, la transformada de la respuesta de impulso de Gabor es la convolución de la transformada de la función sinusoidal y de la transformada de la función gaussiana.

Los filtros de Gabor están directamente relacionados con los wavelets de Gabor, dado que son funciones aproximadamente pasabanda que pueden diseñarse como un banco de filtros con diferentes dilataciones y rotaciones. No obstante, uno de los requisitos de los wavelets, que los filtros sean ortogonales, presenta complicaciones en este caso, requiriendo el uso de wavelets biortogonales. Una opción alternativa consiste en trabajar con representaciones sobrecompletas y considerar que son una buena aproximación al caso ortogonal. Por ejemplo, se pueden descomponer imágenes en bancos de Gabor sobrecompletos y volver a reconstruir la imagen original simplemente sumando los diferentes canales. Esto sólo se podría hacer en modo estricto si los canales de Gabor fuesen ortogonales, pero los errores que se introducen muchas veces no son perceptibles bajo inspección visual.

El filtrado de una imagen con funciones de Gabor está relacionado con los procesos en la corteza visual. Concretamente, son un buen modelo para los campos receptivos de las células simples de la corteza cerebral si se supone que éstas poseen un comportamiento lineal. Además, los filtros de Gabor se han empleado en el procesamiento digital de imágenes, donde se han mostrado eficientes a la hora de realizar diferentes tareas, tales como segmentación de texturas, compresión, etc. El primer hecho sirvió de inspiración para el desarrollo de aplicaciones en el campo del tratamiento de imágenes, mientras que los éxitos en este último campo impulsaron la aparición de teorías acerca de lo que sucede en la corteza visual.

2.5.6 GrabCut

Es un método (12) de segmentación de imágenes basado en cortes de grafos. El corte de grafos es un método de segmentación de imágenes basado en regiones que puede ser utilizado para resolver de manera eficiente una amplia variedad de problemas de bajo nivel en la visión por computadora (visión artificial) como suavizar imágenes, el problema de correspondencia estereoscópica, y muchos otros que pueden ser formulados en términos de minimización de la energía.

Se basa en la teoría de grafos donde el problema de minimización de energía se puede reducir en términos del problema de flujo máximo de un grafo.

En el procedimiento de aplicación del método la imagen se modela como una red de flujo donde un píxel o un grupo de píxeles se asocian con los nodos y los pesos de los arcos definen la similitud entre los píxeles vecinos. Esta red de flujo tiene dos vértices diferenciados S (Fuente) y T (Pozo), que se corresponden con las dos semillas de la inicialización, y las capacidades de cada arco.

Un flujo es una función que a cada arco le asigna un valor entre 0 y su capacidad, representando la ley de conservación (para cada vórtice, excepto la Fuente y el Pozo, el flujo que entra es igual al que sale). El valor de un flujo es el que entra en el Pozo y se busca un flujo de valor máximo.

- Un corte en la red es una partición de los vértices en dos subconjuntos disyuntos, C_1 y C_2 de tal manera que S ∈ C_1 y T ∈ C_2.
- Los arcos del corte son los que van de C_1 a C_2.
- El valor del corte es la suma de las capacidades de sus arcos. Buscamos un corte de valor mínimo.
- El flujo de la red a través del corte es la suma de todas las capacidades alrededor des de S a T menos la suma de todas las capacidades alrededor des de T a S.
- La capacidad del corte es la suma de todas la capacidades alrededor des de S a T.
- Un corte mínimo es un corte donde su capacidad es la mínima de todos los cortes del grafo.

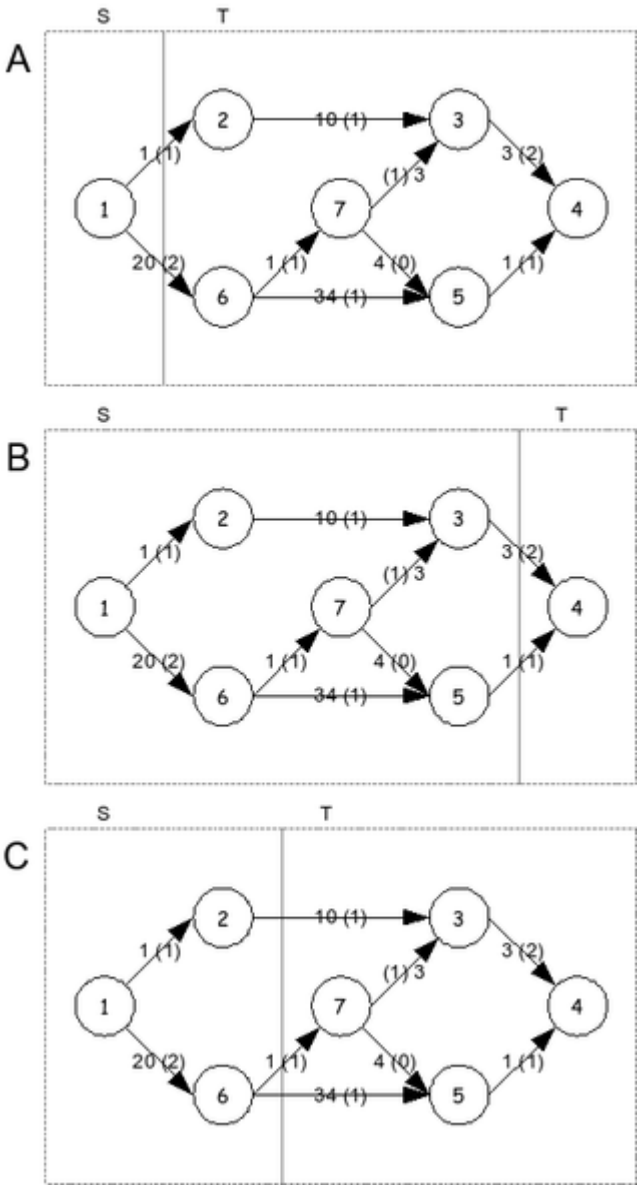


Figura 23: Ejemplos de cortes red de flujo

2.6 Definiciones de Matching

2.6.1 Detectores de características (features detectors)

En la visión artificial o por computador y el procesamiento de imágenes el concepto de detección de características **¡Error! No se encuentra el origen de la referencia.** (13) se efiere a los métodos que tienen como objetivo calcular abstracciones de la información de la imagen y la toma de decisiones locales en cada punto de la imagen, para determinar si una característica de un tipo determinado pertenece a ese punto o no. Las características resultantes serán subconjuntos del dominio de la imagen, a menudo en forma de puntos aislados, curvas continuas o regiones conectadas.

No existe una definición universal o exacta de lo que constituye una característica, y la definición exacta depende a menudo del problema o el tipo de aplicación. En general, una característica se define como una parte "interesante" de una imagen, y estas características se utilizan como punto de partida para muchos algoritmos de visión por computador. Dado que las características se utilizan como punto de partida para los algoritmos, la mayoría de las veces, éstos serán tan buenos como su detector de características sea. Por consiguiente, la propiedad deseable para un detector de característica es la repetición: si la misma característica será detectado o no en dos o más imágenes diferentes de la misma escena.

La detección de características es una operación de bajo nivel de procesamiento sobre las imágenes. Es decir, se realiza generalmente como la primera operación en una imagen, y examina cada píxel para ver si hay una característica que está presente en ese píxel.

Ocasionalmente, cuando la detección de características es computacionalmente cara y hay limitaciones de tiempo, un algoritmo de nivel superior puede ser usado para guiar la fase de detección de características, de modo que sólo se buscan características en ciertas partes de la imagen.

Muchos algoritmos de visión por computador utilizan la detección de características como el paso inicial, por lo que, como resultado, un número muy grande de detectores de características se han desarrollado. Éstos varían mucho en función de las características detectadas, la complejidad computacional y la repetibilidad. Existen muchos detectores, los utilizados en este proyecto son:

A Detector de características ORB

Definido más adelante en descriptores, por aunar la detección y descripción de características.

B Detector de características Harris

El detector de esquinas de Harris (14), es uno de los más usados debido a su elevada invariancia ante escala, rotación, cambios de iluminación y ruido en la imagen. Este detector está basado en la siguiente matriz C(x, y) que se calcula sobre una subventana pxp para cada punto de interés en la posición (x,y) (15).

$$C(x,y) = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix}$$

donde I_x,I_y son los gradientes de la imagen en dirección horizontal y vertical respectivamente. Sean λ_1 y λ_2 los valores propios de la matriz C(x, y), la función de autocorrelación R tendrá un pico si ambos valores propios son altos. Esto significa que desplazamientos en cualquier dirección producirán un incremento significativo, indicando que se trata de una esquina.

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)$$

Algunos autores han obtenido buenos resultados experimentales con k = 0.04. Este es el caso de Davidson y otros (16), que utiliza un detector de esquinas Harris con una subventana de 15x15.

La idea es encontrar puntos en los que el desplazamiento de la ventana en cualquier dirección proporcione un gran cambio en la intensidad de los píxeles, como se puede apreciar en la figura 24.

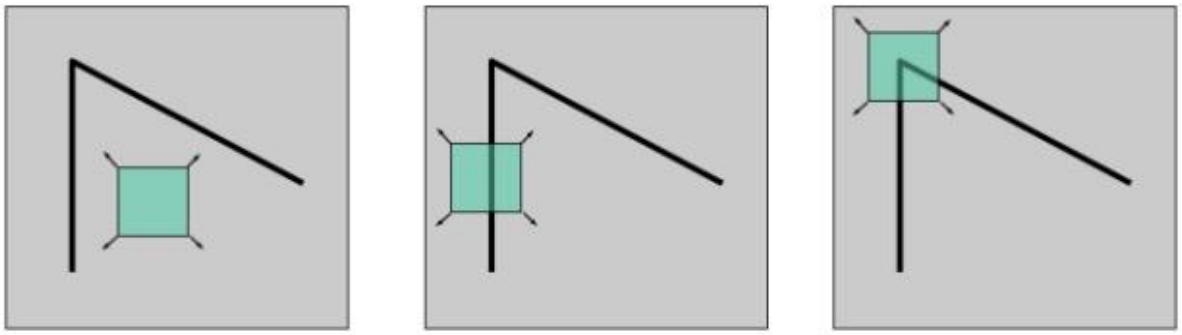


Figura 24: Detector Harris

C Detector de características GFTT

Llamado Good Features to Track (GFTT) (17) está basado en el detector de esquinas de Harris. Permite encontrar contornos reseñables en las imágenes y ajustar el número de contornos a encontrar, y para cada uno de ellos, obtiene las coordenadas X e Y . Esta selección de contornos, se hace mediante la diferencia de intensidad lumínica (diferencias en el gradiente de la imagen) entre los posibles objetos a detectar en una imagen. Uno de los inconvenientes del algoritmo es su tiempo de proceso, algo que al tratarse de dispositivos móviles plantea un mayor problema.

2.6.2 Descriptores o extractores de características (features descriptors)

Una vez que se han detectado las características, una parcela local de la imagen en torno a la característica puede ser extraída. Estas parcelas permanecen inalterables a la escala, rotación, puntos de vista e iluminación. Esta extracción puede implicar cantidades bastante considerables de procesamiento de imágenes. El resultado se conoce como un descriptor de características o vector de características.

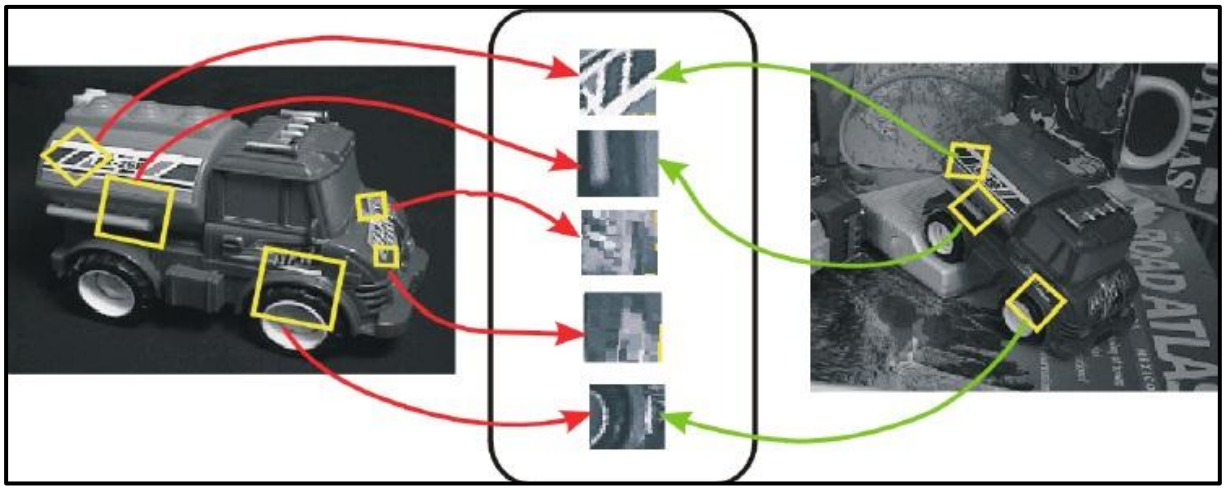


Figura 25: Descriptores comunes en distintas imágenes del mismo objeto

A Detector y Descriptor de características ORB

Los descriptores ORB (Oriented FAST and Rotated BRIEF) (18) se han estudiado por ser una alternativa reciente a otros descriptores de uso más extendido. Como su propio nombre indica, los descriptores ORB están basados en otros métodos anteriores, como el detector de puntos de interés FAST (19) y los descriptores BRIEF (20). La idea de estos descriptores es combinar la rapidez del detector de puntos de interés FAST con las propiedades que aporta el recientemente desarrollado descriptor BRIEF, consiguiendo así un descriptor rápido de calcular y que presenta un buen rendimiento. Además, el tamaño del descriptor es reducido, lo que ayuda a mitigar el problema de memoria que arrastran los descriptores locales. El problema que presenta este descriptor, así como todos los basados en cálculos de gradiente, es que aplicado al problema del reconocimiento de objetos, solo tiene utilidad con objetos que presentan una gran cantidad de cambios del gradiente en sí mismos, los llamados objetos con textura, ya que en caso contrario el algoritmo no detecta puntos clave o de interés donde calcular el descriptor.

B Descriptor de características BRIEF

BRIEF (Binary Robust Independent Elementary Features) es propuesto por Calonder et al (20), y es uno de los métodos más simples. Se utiliza un patrón de muestreo que consiste en 128, 256, o 512 comparaciones (que equivale a 128,256, o 512 bits), con los puntos de muestreo seleccionados al azar de una distribución isotrópica Gaussiana centrada en la ubicación de la característica. Calonder et al. (20) utiliza el detector SURF negando su ganancia computacional, pero BRIEF puede combinarse con cualquier otro detector. Debido a su construcción sencilla y almacenamiento compacto, BRIEF tiene el más bajo requerimiento de cómputo y almacenamiento.

2.6.3 Búsqueda por fuerza bruta

También conocida como búsqueda combinatoria (21), búsqueda exhaustiva o simplemente fuerza bruta, es una técnica trivial pero a menudo usada, que consiste en

enumerar sistemáticamente todos los posibles candidatos para la solución de un problema, con el fin de chequear si dicho candidato satisface la solución al mismo.

Por ejemplo, un algoritmo de fuerza bruta para encontrar el divisor de un número natural *n* consistiría en enumerar todos los enteros desde 1 hasta *n*, chequeando si cada uno de ellos divide *n* sin generar resto. Otro ejemplo de búsqueda por fuerza bruta, en este caso para solucionar el problema de las ocho reinas (posicionar ocho reinas en el tablero de ajedrez de forma que ninguna de ellas ataque al resto), consistiría en examinar todas las combinaciones de posición para las 8 reinas (en total 64! /56! = 178.462.987.637.760 posiciones diferentes), comprobando en cada una de ellas si las reinas se atacan mutuamente.

La búsqueda por fuerza bruta es sencilla de implementar y, siempre que exista, encuentra una solución. Sin embargo, su coste de ejecución es proporcional al número de soluciones candidatas, el cual es exponencialmente proporcional al tamaño del problema.

2.6.4 Distancia de Hamming

Es la efectividad de los códigos de bloque y depende de la diferencia entre una palabra de código válida y otra (22). Cuanto mayor sea esta diferencia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores. A esta diferencia se le llama distancia de Hamming, y se define como el número de bits que tienen que cambiarse para transformar una palabra de código válida en otra palabra de código válida. Si dos palabras de código difieren en una distancia d, se necesitan d errores para convertir una en la otra

Por ejemplo:

- La distancia Hamming entre 1011101 y 1001001 es 2.
- La distancia Hamming entre 2143896 y 2233796 es 3.
- La distancia Hamming entre "tener" y "reses" es 3.

2.6.5 Histograma de color

El histograma de color, (23) representa la frecuencia de aparición de cada una de las intensidades de color presentes en la imagen, a través de la contabilidad de los píxeles que comparten dichos valores de intensidad de color. Está compuesto por diferentes rangos que representan un valor de intensidad de color. Anterior a la etapa de contabilización de cada uno de los valores de los píxeles, existe una etapa de cuantificación de los intervalos o contenedores que se refiere al proceso de reducción del número de intervalos agrupando colores cuyos valores están próximos entre sí en el mismo contenedor. Esta etapa es importante en cuanto a que la cuantificación de los intervalos reduce la información representada por el descriptor sobre la imagen al mismo tiempo que reduce el tiempo de cálculo. Obviamente, cuanto mayor sea el número de intervalos, mayor poder discriminativo tendrá el descriptor. Sin embargo, un gran número de intervalos no sólo incrementará el coste

computacional asociado al descriptor, sino que también resultará inapropiado e ineficiente en cuanto a las comparaciones. El espacio de color se define como un modelo de representación del color con respecto a los valores de intensidad. El rango del espacio de color puede estar comprendida entre una hasta cuatro dimensiones, siendo los espacios más representativos y utilizados los formados por tres componentes o canales de color.

Los espacios de color más conocidos son RGB (Red, Green y Blue) y HSV (Hue (Matiz), Saturation (Saturación), Value (Valor)).

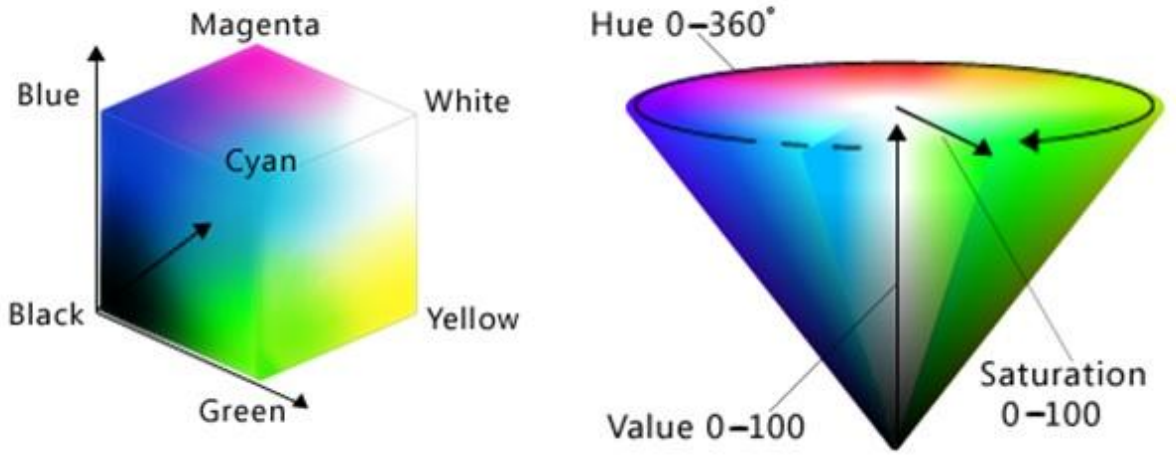


Figura 26: Sistema de color RGB y HSV

El sistema RGB está formado por los colores primarios Rojo, Verde y Azul con valores entre [0,1], y cuya mezcla proporcionada resulta en el color deseado. En HSV, Hue varía entre 0 y 360 grados, ambos rojos. Saturation tiene un rango entre 0 y 100, donde 100 es totalmente saturado y 0 es el color gris. Value, lleva el mismo rango que Saturation donde 100 es lo más brillante posible y 0 lo más oscuro.

3. Metodología

La manera en la que se han utilizado en la aplicación las tecnologías y definiciones propuestas anteriormente será descrita a continuación.

3.1 Evolución de la Metodología

El proyecto se ha estructurado desde su concepción, en tres etapas, segmentación, búsqueda y matching.

En primera instancia se pensó en realizar la fase de búsqueda de imágenes candidatas valiéndose de la búsqueda por imagen que ofrece Google en su web, sin embargo esta opción quedó desechada al no tener su API esta funcionalidad.

Buscando una solución a este problema, se optó por seguir haciendo uso de la API de Google, esta vez para obtener las imágenes candidatas a través de una búsqueda por texto introducida por el usuario. Este cambio abrió nuevas posibilidades ya que al ser los resultados de estas búsquedas muy aleatorios se pensó la posibilidad de realizar esta búsqueda por texto en sitios web específicos de compra, como eBay o Amazon.

La API de eBay permite a un programa comunicarse directamente con su base de datos en formato XML. Mediante el uso de esta API el usuario puede crear programas con las siguientes características:

- Listar artículos nuevos en eBay
- Obtener la lista actual de categorías
- Ver información sobre artículos listados
- Obtener información sobre la puja más alta de artículos propios en venta
- Obtener listas de artículos en venta de un usuario en particular
- Obtener listas de artículos sobre los que haya pujado un usuario en particular
- Mostrar listas de eBay en otros sitios web
- Dejar comentarios sobre otros usuarios al finalizar una transacción comercial

Ya que la API no depende de la interfaz de usuario de eBay, permite crear interfaces estables y de funcionalidad personalizable a la medida del negocio.

Por ejemplo, el usuario puede automatizar el proceso de listado y monitorizado de pujas. No hay necesidad de parsear páginas de eBay que cambian frecuentemente. Un vendedor tiene acceso a más herramientas que puede usar para vender mejor y más rápido. La API también permite extraer información de usuario y automatizar el proceso de fin de una puja además de la entrega entre vendedores y compradores.

Existe un sitio web protegido por contraseña exclusivo para desarrolladores, es la principal fuente de documentación técnica y especificaciones, archivos DTD que definen los XML de

entrada y salida para la API, herramientas para la creación y certificación de la aplicación y un conjunto completo de recursos técnicos adicionales.

Después de la implementación de un primer prototipo que hacía uso de esta API, se buscaron alternativas debido a la mala calidad de las imágenes encontradas, al ser la mayor parte de estas realizadas por usuarios, sin un patrón específico (como un fondo blanco).

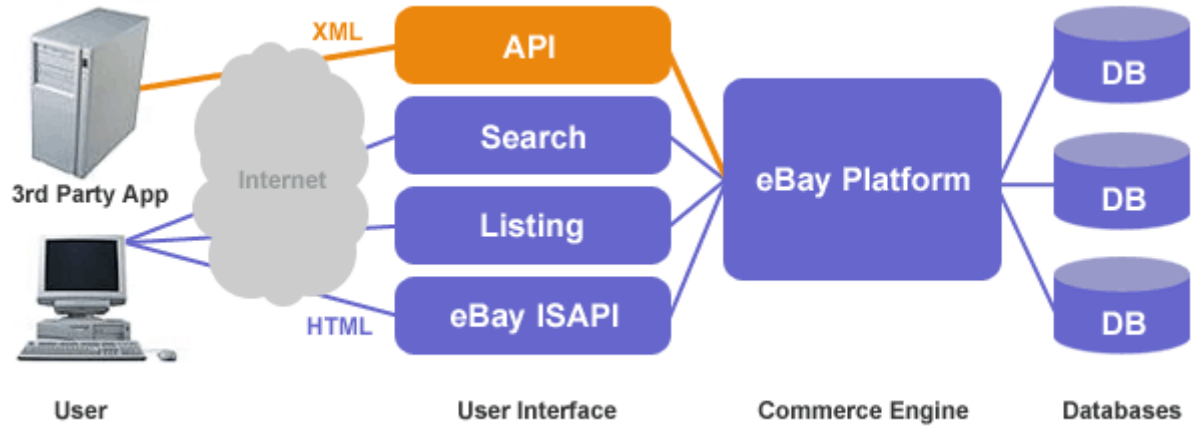


Figura 27: Funcionamiento Api ebay

Por otra parte se consideró la posibilidad de realizar la búsqueda de imágenes candidatas en Amazon a través de su API de publicidad de productos, provee acceso programático a la selección de productos de Amazon y da a los usuarios la posibilidad de ganar dinero con su sitio web. Esta API ayuda al usuario a publicitar productos de Amazon haciendo uso de la capacidad de búsqueda de productos, información de estos y características como comentarios de cliente, productos similares, listas de deseos y listados nuevos y usados. El usuario puede obtener dinero publicitando productos de Amazon junto al programa de asociados de Amazon. Esta opción se desechó ya que Amazon establece en el punto 4e del acuerdo de uso de su API que no se permite al usuario hacer uso de esta, sin consentimiento previo por su parte, en aplicaciones diseñadas y orientadas a dispositivos portátiles.

Para entrenar los algoritmos de matching de la aplicación, se optó por la creación de una base de datos propia en el sitio web dropbox.

Finalmente la opción elegida fue hacer uso de JSoup, para el parseo de las URL's de imágenes de distintas tiendas online. Las candidatas para este propósito fueron la web de El Corte Inglés y Zalando. Ambas nos permiten encontrar prendas estableciendo distintos criterios de búsqueda como sexo, marca, color, precio o tipo de prenda, sin embargo se descartó la búsqueda en la web de El Corte Inglés debido a que dentro de la categoría de pantalones, estos se muestran siempre con un modelo, lo que empeora los resultados devueltos por el matching. Al basarse este proceso de búsqueda en el parseo de URL's abre la posibilidad de realizarlo en cualquier otra web de similares características.

3.2 Api de Google

Una de las principales funcionalidades de esta aplicación consiste en buscar imágenes por la red. Para ello hemos hecho uso de la API de Google llamada Google Custom Search.

Google Custom Search permite aprovechar el poder del buscador web de Google en cualquier sitio web o aplicación móvil programáticamente, adaptándose a las necesidades e intereses de cada uno. Usando esta API podemos realizar tanto búsquedas web como búsquedas de imágenes obteniendo resultados en formato JSON o Atom.

JavaScript Object Notation (JSON) es un estándar abierto basado en texto diseñado para intercambio de datos. Deriva del lenguaje de scripting JavaScript para representar estructuras de datos simples, llamados objetos. Este formato es comúnmente utilizado para serializar datos y trasmitirlos entre un servidor y una aplicación web.

JSON es fácil para los desarrolladores de leer y escribir y fácil para las máquinas de analizar y generar. Es completamente independiente de los lenguajes de programación pero resulta más familiar para los programadores de los lenguajes de la familia C incluyendo C, C++, C#, Java, JavaScript, Perl, Python y otros. Estas características hacen de JSON un lenguaje ideal para el intercambio de datos.

Su motor de búsqueda personalizado da la posibilidad de restringir la búsqueda si se quiere a determinados sitios web o usar todas las páginas indexadas por Google. Una de sus principales ventajas es poder filtrar la búsqueda mediante una gran cantidad de características.

Google ofrece su API mediante tarifa gratuita y otra de pago. Usando la cuota gratuita sólo nos permite realizar 100 búsquedas ya que está pensado para que la usen los desarrolladores para las pruebas, aunque aun así puede resultar escasa. Si se elige la opción de pago, habrá que abonar cinco dólares por cada cinco mil búsquedas, hasta un máximo de diez mil al día.

Para poder hacer uso de Google Custom Search API, en primer lugar hay que tener una cuenta de Google, a continuación necesitamos obtener una clave para que identifique nuestras búsquedas y crear el propio motor de búsqueda del que también obtendremos su identificador, todo esto se puede hacer en los menús de esta API.

Una vez tenemos todos los datos neceárseos podremos crear la url que tendrá el siguiente aspecto.

https://www.googleapis.com/customsearch/v1?key=INSERT-YOUR-KEY&cx=017576662512468239146:omuauf_lfve&q=lectures.

Desglosando esta url:

- https://www.googleapis.com/customsearch/v1
- **key**=INSERT-YOUR-KEY
- **cx**=017576662512468239146:omuauf_lfve
- **q**=camiseta

El parámetro "key" hace referencia a la clave que hemos tenido que crear para identificar las búsquedas y que estén asociadas a la cuenta de Google. El parámetro "cx" sirve para identificar nuestro motor de búsqueda y el parámetro "q" es la búsqueda que se realizara en el buscador de Google.

Esta url que hemos mostrado es la más simple posible pero esta API nos ofrece multitud de opción con las que podremos filtrar y afinar mas las búsquedas para así obtener unos resultados lo más precisos posibles.

Podemos destacar las siguientes opciones de búsqueda:

- cr: Nos permite restringir la búsqueda por país.
- dateRestrict: Especifica la búsqueda para un periodo de tiempo determinado.
- exactTerms: Identifica una frase o palabra que todos los documentos debe de tener.
- excludeTerms: Identifica una frase o palabra que no debe aparecer en ninguno de los resultados.
- fileType: Devuelve imágenes del tipo especificado como png, jpg,svg,pdf…
- imgColorType: Devuelve imágenes en blanco y negro, escala de grises o en color.
- imgDominantColor: Especifica el color predominante de que tiene que tener las imágenes que nos devuelva, permite los colores, amarillo, verde, verde azulado, azul, morado, rosa blanco y marron.
- imgSize: Determina el tamaño de las imágenes devueltas admitiendo los siguientes: icono, pequeña, mediana, grande, extra grande, extra extra grande y enorme.
- imgType: Devuelve imágenes de un tipo especificado que pueden ser, prediseñadas, cara, arte línea, noticias o foto.
- lr: Restringe las búsquedas por el lenguaje.
- num: Numero de resultados devueltos.
- rights: Filtra el basándose en la licencia que tengan los resultados.
- safe: Determina el nivel de seguridad de búsqueda.

- searchType: Mediante este atributo se determina que las búsquedas serán de imágenes y no web.
- siteSearch: Especifica que todos los resultados deben de ser de páginas de un sitio dado.
- start: Indice del primer resultado a devolver.

Para nuestro concreto caso hemos usado

- dateRestrict para obtener unos resultados que no sean demasiado antiguos.
- imgType para determinar que las imágenes obtenidas sean prediseñadas ya que estas imágenes siempre pertenecen a sitios web de venta de ropa y evitamos que por ejemplo buscando "camiseta negra" nos devuelva resultados de gente llevado camisetas así, utilizado esta restricción solo obtenemos la prenda suelta y en perfectas condiciones de trabajar con ella.
- num: especificamos el número de resultados a 30 ya que mas allá de este número de búsquedas las imágenes ya tienen poco que ver con lo que hemos buscado.
- searchType: determinamos que la búsqueda sea sólo de imágenes.

De esta manera se obtiene una url así

https://www.googleapis.com/customsearch/v1?key=AIzaSyCKj94zF4oPDHF-AzcfYWqa8-bX7mCp6F4&cx=011729499595405224404:haf3odx65_m&q=comprar+camiseta+zara&num=10&start=10&searchType=image&imgType=clipart&https://www.googleapis.com/customsearch/v1?key=AIzaSyCKj94zF4oPDHF-AzcfYWqa8-bX7mCp6F4&cx=011729499595405224404:haf3odx65_m&q=comprar+camiseta+negra+zara&num=10&start=10&searchType=image&imgType=clipart&dateRestrict=y1

Una vez tenemos la url que queremos realizamos una petición GET con la que obtenemos un string de objetos JSON con todos los resultados de la búsqueda especificada del cual lo que nos interesa son los ítems de cada imagen:


```
{
   "kind": "customsearch#result",
   "title": "CAMISETA CUELLO PICO - Camisetas - TRF - ZARA España",
   "htmlTitle": "\u003cb\u003eCAMISETA\u003c/b\u003e CUELLO PICO -
\u003cb\u003eCamisetas\u003c/b\u003e - TRF -
\u003cb\u003eZARA\u003c/b\u003e España",
   "link":
"http://static.zara.net/photos//2013/V/0/1/p/1919/881/800/2/1919881800_1_1_
3.jpg?timestamp=1366812595305",
   "displayLink": "www.zara.com",
   "snippet": "Imagen 1 de CAMISETA CUELLO",
   "htmlSnippet": "Imagen 1 de \u003cb\u003eCAMISETA\u003c/b\u003e CUELLO",
   "mime": "image/jpeg",
   "image": {
```

```
    "contextLink":
"http://www.zara.com/webapp/wcs/stores/servlet/product/es/es/zara-
S2013/358034/1048478/CAMISETA+CUELLO+PICO",
    "height": 640,
    "width": 400,
    "byteSize": 28398,
    "thumbnailLink": "https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcRFFvhqCfw-
ou91PtViWNg3UWetKOsZoso0U038Ig1iZnMVYkUONsJgkVS6",
    "thumbnailHeight": 137,
    "thumbnailWidth": 86
   }
```


Este ítem nos ofrece toda la información que la API tiene sobre una imagen en concreto. Mediante un proceso de deserializacion de string JSON, de entre toda esta información nos quedaremos con las siguientes:

- title: título que aparecerá en los resultados de las imágenes de nuestra aplicación.
- link: enlace a la imagen en tamaño original.
- contextLink: enlace a la página concreta en la que esta imagen esta y donde el usuario obtendrá toda la información de la imagen.
- thumbnailLink: enlace que nos devuelve la imagen en un tamaño reducido que será con la que trabajemos.

3.3 Biblioteca OpenCV

OpenCV tiene una estructura modular, lo que significa que el paquete incluye varias bibliotecas estáticas o compartidas. Están disponibles los siguientes módulos:

core - Módulo compacto que define estructuras básicas, incluyendo el denso array multi-dimensional Mat y funciones básicas utilizadas por el resto de módulos.

imgproc - Módulo de procesamiento de imágenes que incluye filtros lineares y no lineares de imágenes, transformaciones geométricas (redimensionamiento, deformación afín y de perspectiva, mapeo genérico basado en tablas) conversión de espacios de colores, histogramas y más.

video - Un módulos de video análisis que incluye estimación de movimiento, sustracción de fondo, y algoritmos de rastreo de objetos.

calib3d - Algoritmos de geometría multivista básica, calibración simple y estéreo de cámara, estimación de posición de objetos, algoritmos de correspondencia estéreo, y reconstrucción de elementos en 3D.

features2d - Detectores de características salientes, descriptores y emparejadores de descriptores.

objdetect - Detección de objetos e instancias de las clases predefinidas (por ejemplo, caras, ojos, tazas, gente, coches y más)

highgui - Interfaz de fácil uso para la captura de vídeo, imágenes y video codecs, a al igual que simples capacidades de UI.

gpu - Algoritmos de distintos módulos de OpenCV acelerados mediante GPU.

Esta aplicación se basa principalmente en el uso de las funciones de segmentación y matching de la biblioteca, para ello recurrimos a funciones de varios de los módulos principales, Core, Imgproc, Highgui y Feature2D.

CORE

En este módulo se definen los bloques básicos de la biblioteca, entre ellos encontramos uno de los tipos más importantes para la aplicación, Mat.

MAT-Contenedor básico de imágenes

Existen múltiples formas de adquirir imágenes digitales del mundo real: cámaras digitales, scanners, tomografía computarizada o imágenes de resonancia magnética para nombrar unos pocos. En todos los casos lo que nosotros (humanos) vemos son imágenes. Sin embargo, al transformar esta información a nuestros dispositivos digitales lo que grabamos son valores numéricos para cada punto de una imagen.

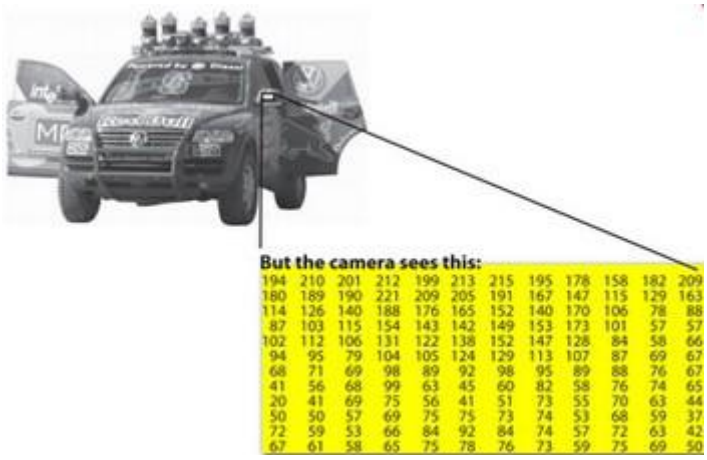


Figura 28: Matriz pixels

Por ejemplo en la imagen superior se puede ver que el retrovisor no es más que una matriz que contiene todos los valores de intensidad de los pixels. Cómo conseguimos y almacenamos los valores de los pixels puede variar según nuestras necesidades, todas las imágenes dentro de un ordenador pueden reducirse a matrices numéricas y alguna otra información describiendo la propia matriz. OpenCV es una biblioteca de visión por computador cuyo objetivo principal es procesar y manipular esta información para extrapolar otra. En consecuencia, lo primero que se necesita entender es cómo OpenCV almacena y manipula imágenes.

En las primeras versiones de OpenCV, desarrolladas en C, las imágenes se almacenaban en una estructura llamada IplImage, que traía consigo problemas como el de la gestión manual(reserva y liberación de memoria), con el tipo Mat esta gestión es automática(utilizamos en todo momento solo el espacio que necesitamos), si bien en caso de problemas pequeños la gestión no supone un problema, lo es cuánto más grande y denso es el código con el que trabajamos, como es nuestro caso, permitiéndonos esta característica centrarnos en el objetivo de la aplicación antes que en la gestión de memoria.

La clase Mat representa un array numérico en 2D que hace las veces de matriz, imagen, mapa de flujo óptico, etc. Es muy similar al tipo CvMat de versiones anteriores de OpenCV, y al igual que este, la matriz puede ser multicanal, pero soporta además mecanismos ROI ("Region of Interest") como hace IplImage. Durante la creación podemos especificar un tamaño y tipos iniciales de cada elemento de la matriz. Este tipo especifica, por ejemplo, valores de pixels de imagen de 1 byte con signo (CV-8U), o tres canales para una imagen en color (CV-oUC3) o incluso números en punto flotante 32it/64bit (CV-32F).

Acerca del almacenamiento de los valores de los pixeles podemos elegir el espacio de color y el tipo de datos a utilizar. El espacio de color se refiere al modo en el que combinamos los componentes de color para codificar un color dado. La forma más simple es la escala de grises, como en la imagen del ejemplo. Para imágenes en color tenemos muchos métodos entre los que elegir, aunque todos ellos se basan en tres o cuatro componentes básicos y en la combinación de estos para la obtención del resto de colores. La forma más popular es RGB, principalmente porque es también cómo nuestros ojos forman los colores. Estos colores base son rojo, verde y azul. Para codificar la transparencia de un color a veces se añade un cuarto elemento: alpha(A). Sin embargo, existen muchos sistemas de color, cada uno con sus ventajas:

• RGB es el más común al utilizar nuestros ojos un sistema similar, al igual que nuestros dispositivos de reproducción de imágenes.

• HSC y HLS descomponen los colores según sus valores de matices, saturación y luminosidad, que es una forma más natural para nosotros de describir colores. Al hacer uso de este sistema se puede ignorar la última componente, haciendo el algoritmo menos sensible a condiciones de iluminación de la imagen de entrada.

• YCvCb es utilizado por el popular formato JPEG.

• CIE L*a*b* es perceptualmente un espacio de color uniforme, lo cual es útil si necesitamos medir la distancia entre dos colores dados.

La función más importante dentro de este módulo para la aplicación es **KMeans**, ya que es la base para la segmentación de las imágenes, entramos en detalle más adelante, junto a esta encontramos funciones para la aplicación de filtros de Gabor y otros.

3.4 JSoup

JSoup es un proyecto lanzado en su primera versión beta en 2010, de código abierto se distribuye bajo licencia MIT, es una biblioteca para trabajar con HTML. Provee una API muy conveniente para la extracción y manipulación de datos, haciendo uso de lo mejor de DOM (Document Object Model), CSS(Cascading Style Sheets), y métodos tipo jquery.

JSoup implementa la especificación de HTML5 WHATWG, y parsea(analiza gramaticalmente) HTML de la misma forma que el modelo de objetos de documentos que los navegadores modernos:

- Obteniendo y analizando gramaticalmente HTML de una URL, archive o cadena de texto
- Encontrando y extrayendo información, usando el recorrido de DOM o selectores CSS
- Manipulando los elementos HTML, atributos y texto
- Limpiando contenido enviado por los usuarios frente una lista segura, para prevenir ataques XSS(Cross Site Scripting)
- Devolviendo un HTML pulcro

JSoup está diseñado para lidiar con todas las variedades de HTML encontradas, desde uno limpio y validado hasta una mezcla de etiquetas invalidas, jsoup creará un árbol de análisis sensato.

Esta biblioteca permite al usuario encontrar y manipular elementos haciendo uso de selectores sintácticos tipo jquery o CSS, mediante los métodos *Element.select(String selector*) y *Elements.select(String selector)* como se muestra a continuación:

```
File input = new File("/tmp/input.html");
Document doc = Jsoup.parse(input, "UTF-8", "http://example.com/");

Elements links = doc.select("a[href]"); // a with href
Elements pngs = doc.select("img[src$=.png]");
  // img with src ending .png

Element masthead = doc.select("div.masthead").first();
  // div with class=masthead

Elements resultLinks = doc.select("h3.r > a"); // direct a after h3
```

Los elementos jsoup soportan selectores sintácticos de tipo CSS o jquery para encontrar y emparejar elementos, esto permite búsquedas muy potentes y robustas.

El método *select* está disponible en los tipos *Document*, *Element* y *Elements*. Es contextual, de tal forma que el usuario pueda filtrar seleccionando a partir de un elemento específico o cambiando las llamadas de selección.

Select devuelve una lista de elementos (como *Elements*), que proveen un abanico de métodos para extraer y manipular los resultados.

Dentro de los selectores encontramos:

- Tagname: encuentra elementos por etiqueta, por ejemplo: a
- Ns|tag: encuentra elementos por etiqueta dentro de un espacio de nombres, por ejemplo: fb|name encuentra elementos del tipo<fb;name>
- #id: encuentra elementos por ID, por ejemplo: #logo
- .class: encuentra elementos por nombre de clase, por ejemplo: .masthead
- [attribute]: encuentra elementos con atributo, por ejemplo: [href]
- [^attr]: encuentra elementos con un prefijo en el nombre del atributo, por ejemplo: [^data-] encuentra elementos con atributos HTML5 de tipo dataset
- [attr=value]: encuentra elementos cuyo atributo tenga el valor dado, por ejemplo [width=500]
- [attr^=value],[attr$=value], [attr*=value]: encuentra elementos con atributos que empiezan, terminan o contienen el valor dado, por ejemplo [href*=/path/]
- [attr~=regex]: encuentra elementos con valores de atributos que encajen en la expresión regular, por ejemplo: img[src~=(?i)\.(png|jpe?g)]
- *: encuentra todos los elementos

En caso de querer descargar una imagen, una vez obtenida su URL, el siguiente código es aconsejado:

```
        //Open a URL Stream
        Response resultImageResponse =
Jsoup.connect(imageLocation).cookies(cookies).ignoreContentType(true).e
xecute();

        // output here
        FileOutputStream out = (new FileOutputStream(new
java.io.File(outputFolder + name)));
        out.write(resultImageResponse.bodyAsBytes());           //
resultImageResponse.body() is where the image's contents are.

        out.close();
```

3.5 Segmentación

La segmentación de imágenes es una parte del campo de visión por computador que consiste en dividir una imagen digital en varias regiones o grupos de pixeles de tal manera que se cree

una nueva imagen más fácil de analizar. La segmentación se usa tanto para localizar objetos como para encontrar los límites de estos dentro de la imagen. Se puede definir como un proceso por el cual se le asigna una etiqueta a cada pixel de tal manera que los pixeles que compartan la misma etiqueta tendrán ciertas características visuales similares.

El objetivo concreto para el desarrollo de la aplicación, es segmentar la imagen que a procesar en dos grupos, el objeto a analizar y el resto de la imagen que no aporta ninguna información útil.

Uno de los posibles problemas que puede aparecer a la hora de trabajar con una imagen es el ruido en la imagen, que consiste en una variación aleatoria no correspondiente con la realidad del brillo o del color producido al tomar una imagen digital. Esto puede causar que algunos pixeles no tengan su información correcta.

La forma en que se trata este problema es modelando el ruido mediante un suavizado gaussiano de la imagen. Aplicamos un núcleo gaussiano de 9x9 de tal manera que cada pixel de la imagen se ve afectado por todos los pixeles a 9 unidades de distancia pero no todos se afectan de la misma manera, utilizando un núcleo Gaussiano tienen más fuerza los pixeles más cercanos. De tal manera, se obtiene así un promedio de los valores de los pixeles vecinos consiguiendo así dos objetivos:

- Eliminar el ruido.
- Difuminar los bordes pequeños que no son útiles para delimitar el objeto que se quiere segmentar con el fondo.

Una vez eliminado el ruido y los bordes insignificantes se lleva a cabo el proceso de segmentación.

Para realizar la segmentación de la imagen, el algoritmo se basa en la segmentación por características, es decir, a cada pixel se le asigna una serie de atributos discriminantes con respecto al resto, lo que permitirá poder agruparlos según sus similitudes y diferencias. Al final de la segmentación cada uno de los pixeles de una región guardará una gran similitud con el resto.

Figura 29: Ejemplo segmentación

Como puede observarse en los dos ejemplos de la figura 29 segmentando las imágenes originales de la izquierda se consigue aislar la prenda de ropa después de haber eliminado los brazos, el cuello, el pantalón y el resto del fondo de la imagen.

La forma utilizada para llevar a cabo este proceso es con el algoritmo de clustering k-means cuyo propósito es la partición de un conjunto de *n* elementos en *k* grupos o clusters, donde en este caso *n* son el conjunto de pixeles de la imagen con sus propiedades determinadas y *k* es 2, un cluster será el objeto segmentado y el otro será el resto de la imagen.

Para determinar a que cluster pertenece cada uno de los pixeles de la imagen se han introducido las siguientes características discriminantes a cada uno de ellos:

- Coordenada X.
- Coordenada Y.
- Componente R.
- Componente G.
- Componente B.

- Resultado algoritmo Grabcut.
- Textura tras aplicar el filtro de Gabor.
- Distancia Euclídea con respecto al centro.

A cada una de estas características se le asigna un peso específico en el algoritmo para así determinar su importancia dentro del mismo.

Se decidió bajar el peso de las coordenadas dentro del algoritmo al 0,5 de su valor original debido al amplio rango de valores que pueden tomar, desde (0,0) hasta el tamaño máximo de la imagen por lo que si tuvieran un mayor peso lo único que conseguiríamos seria que agrupase los pixeles por sus coordenadas.

En el caso de los componentes de color RGB, debido a que es la principal característica de cada pixel ya que la mayoría de las prendas de ropa están compuestas por colores uniformes, es fundamental para poder delimitar la imagen con respecto al fondo. Por este motivo se introduce el doble de su valor.

Otro de los atributos discriminantes de una imagen consiste en las diferentes texturas de los pixeles. Se aplica a la imagen un filtro Gabor resultando así una nueva imagen en escala de grises que discriminará cada uno de los píxeles según el tipo de textura que tengan.

El resultado del algoritmo de segmentación grabcut se convierte en una característica discriminante más que se le asigna un valor del 0.8 del su valor original.

Para llevar a cabo este proceso se crean una serie de núcleos definidos matemáticamente como:

$$G(x,y) = \exp\left(\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

Donde:

- $x' = x\cos\theta + y\sin\theta$
- $y' = x\sin\theta + y\cos\theta$
- λ: longitud de onda del factor coseno medido en pixeles (mínimo 2). Su valor debe de ser menor que un quinto del tamaño de la imagen.
- θ: orientación medida en grados con un rango de 0º a 360º.
- ψ: desfase especificado en grados con un rango de valores reales de -180 a 180.
- γ: ratio aspect, determina la elepticidad del núcleo.
- σ: varianza de la función, depende de la longitud de onda siendo 0.59* λ

Se han creado con un tamaño de ventana de 9x9, ocho núcleos de Gabor, uno para cada orientación: 0, 45º,90º, 135º, 180º, 225º, 270º y 315º.

El resto de los parámetros son comunes para todos, siendo la longitud de onda un décimo del tamaño de la imagen, el desfase se ha establecido a 90º, la elepticidad γ =1 creándose así un núcleo circular.

De esta manera se consigue el siguiente resultado de la figura 30:

Figura 30: Resultado aplicar filtro Gabor

Al resultado de aplicar este filtro a la imagen original se introduce al algoritmo k-means dividido por dos.

En cuanto a la distancia Euclídea respecto al centro, aun tratándose también de las posiciones de cada pixel, nos interesa más debido a que el objeto que se quiere segmentar siempre va a tender a estar cerca del centro, por lo que esta característica ayudará a mantener los pixeles centrales en el mismo grupo y a discriminar los bordes y esquinas de las imágenes que siempre serán parte del fondo de la imagen. De tal manera se introduce al algoritmo su valor original.

En la figura 31 se puede apreciar que para cada pixel se van a representar mediante la X y la Y las coordenadas, con R G B, la cantidad de color Red, Green y Blue, D especifica la distancia euclídea al centro, G la pertenencia a Gabor y V la diferencia con respecto a sus vecinos.

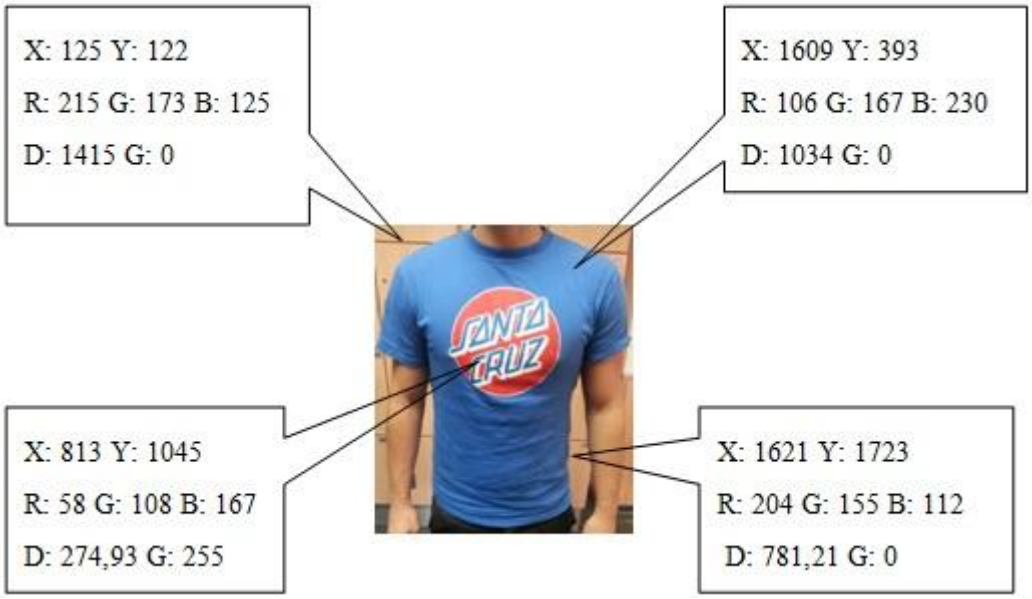


Figura 31: Ejemplo características discriminantes de los pixels

Tras realizar todo el proceso de obtención de las características discriminantes de todos los píxeles de la imagen y añadidas a un vector se procede a iniciar el algoritmo de kmeans que se ejecuta con mediante diferentes inicializaciones, con los centroides colocados de manera aleatoria o en el centro. Al ser un algoritmo iterativo se establece un criterio de terminación de tal manera que parará cuando ocurra una de las dos siguientes condiciones: que los centroides no se muevan más de un épsilon igual a 1 o que se llegue a las 10 iteraciones, devolviendo el mejor resultado al terminar todas las iteraciones.

El algoritmo de k-means devuelve una matriz en la que todos los píxeles están etiquetados en dos clases, por ejemplo; fondo o figura. Esto resulta en un problema ya que ahora hay que averiguar cuál de las dos etiquetas representa al objeto y cual al resto de la imagen (fondo).

Debido a que en la aplicación móvil se permite al usuario poder recortar la imagen para que los resultados sean más precisos; suponemos que la mayor parte de la imagen estará formada por el objeto, por lo tanto, contamos cuantos pixeles pertenecen a cada conjunto de pixeles y el grupo con mayor número será el que forma el objeto. Durante este proceso también se guardan los pixeles más externos del objeto para a continuación poder recortar la imagen de tal manera que al final del proceso casi toda la imagen final será el objeto deseado.

Otro de los problemas que surgen al segmentar una prenda suelen ser los dibujos, letras, símbolos y marcas que muchas tienen y que el algoritmo puede llegar a confundirse y asignar a estas partes de la prenda como que no forman parte del objeto.

Una vez pasado el algoritmo de k-means, y averiguado el color de pixel que ha resultado ser figura y no fondo, se determina el contorno que forma la prenda de vestir y se rellena dicho contorno. Gracias a esto se consigue obviar los logos de marca o dibujos que contienen las prendas y que estropearían el resultado. Como solo se dispone de dos cluster, lo que no forma

parte de la figura es fondo, y para evitar que los logos u objetos queden en la parte del fondo, se realiza esta operación y se consigue montar una máscara uniforme que garantiza extraer la figura del fondo de una manera eficaz.

Tras crear la máscara, que está formada por pixeles blancos y negros, se aplica sobre la imagen original creándose una nueva imagen que será el resultado final de tal manera que todos los pixeles que componían el fondo se quedan en blanco, para que así tenga una mayor similitud con las imágenes prediseñadas que se encuentran en la red y más adelante se comparan con ellas.

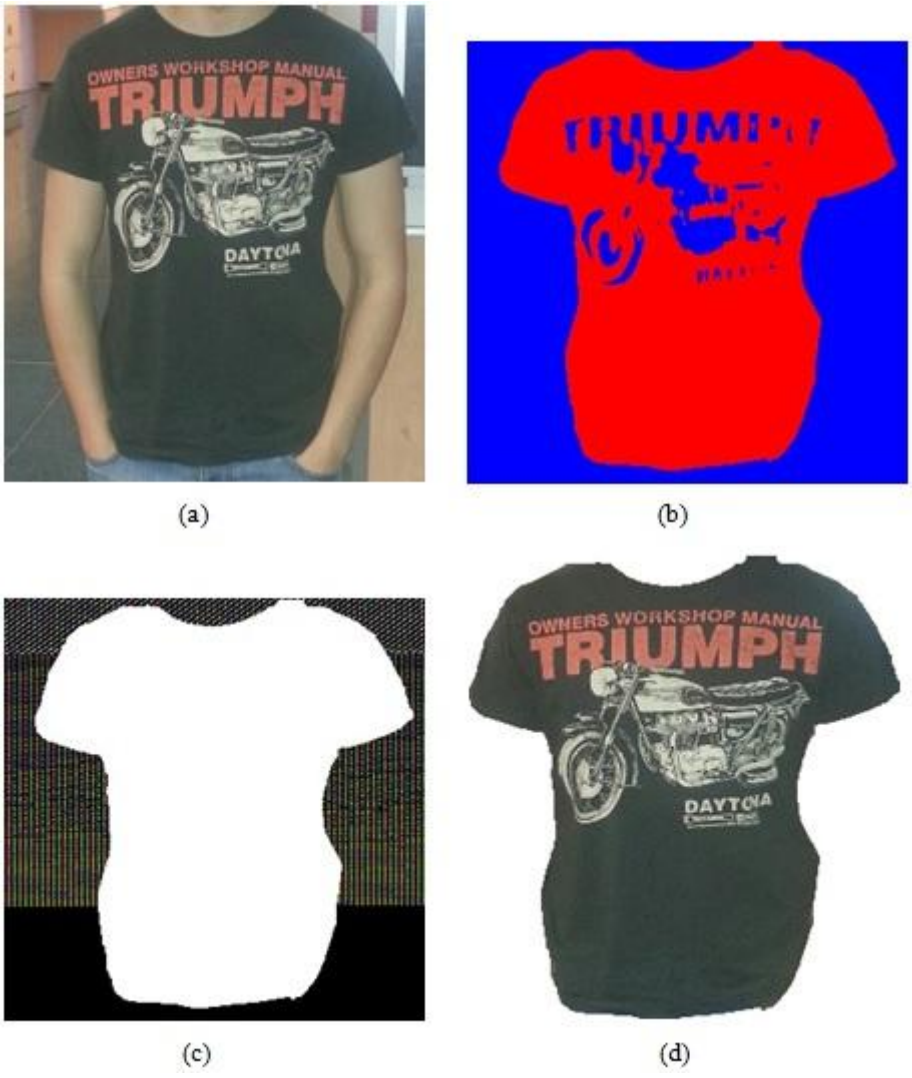


Figura 32: Ejemplo completo segmentación

(a) Imagen tomada con la cámara del dispositivo móvil. (b) Imagen reconstruida después del algoritmo k-means. (c) Máscara con el objeto localizado. (d) Resultado final.

3.5.1 Threshold para la segmentación

El éxito del proceso de segmentación depende en cierta medida de la calidad de la imagen tomada. Ciertos parámetros como la iluminación excesiva como insuficiente, que la imagen esté desenfocada, que el propio flash de la cámara haya dejado un foco de iluminación en una parte de la foto o que el fondo de la imagen sea del mismo color que la prenda, pueden hacer que la segmentación no se realice correctamente.

Para obtener información sobre qué tan bien el algoritmo de k-means ha realizado el clustering nos basamos en la desviación media que devuelve:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

Cuanto mayor sea esta desviación, peor se habrá realizado la segmentación.

En el caso de que la segmentación se haya realizado de forma incorrecta es inútil intentar realizar el proceso de matching, por tanto, para no hacer perder tiempo al usuario, la aplicación tiene un umbral (threshold) que hace de filtro; las imágenes cuya desviación (*V*) sobrepasen un cierto valor de umbral son rechazadas por la aplicación, pidiéndole al usuario usar una foto con una calidad mayor.

Para determinar este threshold se ha usado toda la información obtenida de las pruebas de la aplicación, que se observan en el apéndice B y en la gráfica con la distribución de las pruebas según su puntuación ---que se encuentra en la sección de resultados.

3.6 Matching

Se va a comparar la imagen de entrada, ya sea por la galería o la cámara de fotos, contra la base de datos de imágenes que nos ofrece Google o Zalando. Al final de esta comparación se ofrecerá al usuario los tres mejores resultados, es decir las tres imágenes que mejor puntuación (menor error) obtengan al final de este proceso.

Cuando se habla de Matching entre dos imágenes, lo que se persigue es calcular un valor que represente el grado de similitud entre ellas (error). A continuación, se verá en detalle las distintas formas de calcular dicho valor.

Dependiendo del modo de precisión elegido por el usuario, se someterán a unos algoritmos u otros todas las imágenes. En la siguiente tabla se puede observar los algoritmos en cada posible modo.

Modo	Feature detector	Descriptor extractor	Histograma de color
1	ORB	ORB	Correlación
2	ORB GFTT	ORB BRIEF	Correlación Chi-cuadrado
3	ORB HARRIS ORB GFTT	ORB BRIEF BRIEF BRIEF	Correlación, Chi-cuadrado Intersección Bhattacharyya

Figura 33: Tabla modos aplicación

En cualquiera de los modos, las imágenes pasarán como mínimo por un detector de características, un descriptor y un histograma. Y como máximo por cuatro detectores de características, cuatro descriptores y cuatro histogramas. Todos los descriptores serán comparados por el mismo método, fuerza bruta sobre distancia de Hamming. Todos estos términos fueron definidos anteriormente y serán completados a continuación.

Cabe destacar que utilizaremos dos tipos de descriptores de imágenes. Por un lado SURF y BRIEF que son descriptores locales y por otro el histograma de color que es un descriptor global.

Se va a diferenciar entre descriptores locales y globales para representar las imágenes.

3.6.1 Descriptores Locales

Actúan sobre regiones de interés, previamente identificadas, construyendo un vector de características de esa región adyacente al mismo. Estas regiones conocidas como keypoints, se almacenan en vectores formando el descriptor. Mediante los detectores de características (ORB, HARRIS y GFTT) serán localizadas estas regiones.

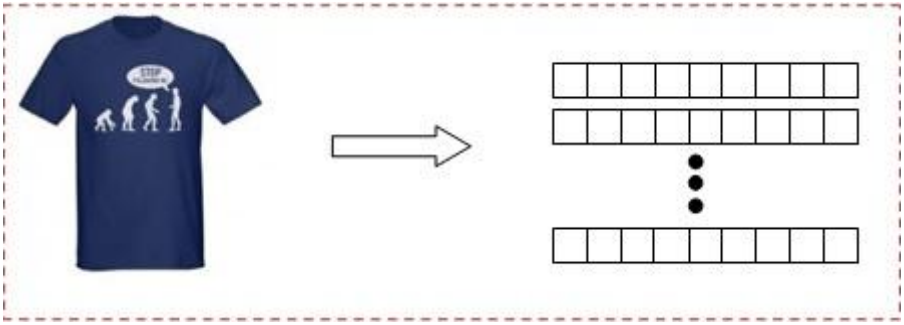


Figura 34: Representación en varios vectores de los descriptores locales

A Método de los keypoints, detectores y descriptores

Tanto para la imagen de entrada como para las imágenes candidatas, lo primero de todo es obtener a través de los detectores de características, los keypoints, es decir los puntos más significativos de cada imagen. A partir de estos keypoints propios de cada imagen, se van a calcular sus descriptores. Siguiendo la figura 35 se aplicará el algoritmo correspondiente en cada función necesaria para hallar tanto los keypoints como los descriptores.

Figura 35: Keypoints extraídos mediante ORB (1), HARRIS (2), GFTT (3)

B Comparando descriptores

Para cada par imagen de entrada-imagen candidata, se procede:

Recorriendo todo el espacio muestral (búsqueda exhaustiva) que nos ofrecen los descriptores con las características de las imágenes, y utilizando la Distancia de Hamming, quedarán solo los descriptores que coincidan en ambas imágenes (matches). También se calculará la distancia entre dichos descriptores coincidentes. Esta distancia representa la separación entre el lugar donde se encuentra esa característica en la imagen original y la candidata.

En la figura 36 se puede apreciar las correspondencias entre los descriptores de las camisetas. En la primera imagen con detector y descriptor ORB. En la segunda con detector GFTT y descriptor BRIEF.

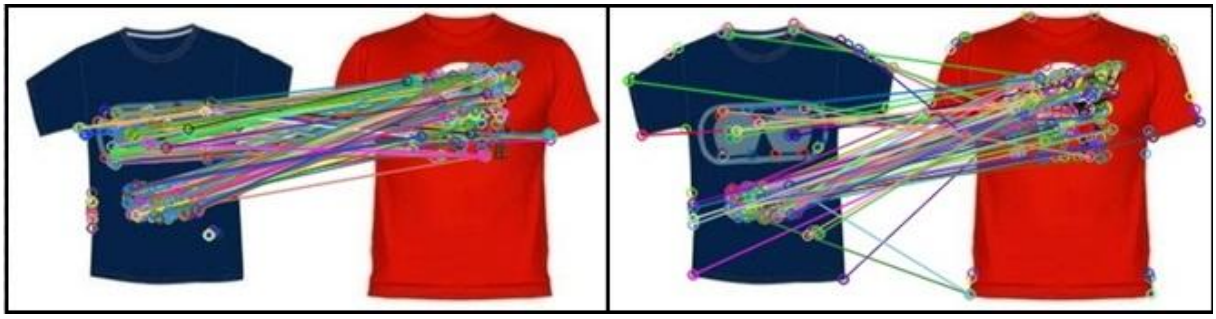


Figura 36: Correspondencias entre descriptores

C El vecino más próximo

Una vez obtenidos los matches, y sus distancias correspondientes, se calcula la distancia mínima. Y solo se mantendrán los matches cuya distancia sea menor al doble de la distancia mínima. Realizando este filtro de los matches se asegura una mayor similitud entre las imágenes, puesto que si tienen las mismas características pero en zonas distintas a la original no interesa dicha característica y quedará descartada.

Una vez realizado este proceso de selección de matches, se tendrá cada imagen candidata con un valor asociado. Se elegirá la imagen cuyo valor sea mayor puesto que será la que tenga el mayor número de coincidencias con respecto a la original.

De una manera más esquemática quedaría como sigue:

1. Aplicar detectores de características sobre todas las imágenes, obteniendo keypoints.

2. Utilizar los keypoints de cada imagen para sacar los descriptores de la misma.

3. Comparar descriptores de la imagen de entrada con cada imagen candidata obteniendo los matches.

4. Descartar matches no válidos, filtrando solo los buenos matches.

5. Finalmente, se obtiene para cada imagen candidata un valor de similitud (error).

El peso de los descriptores locales que intervengan en una ejecución lo harán en el mismo porcentaje. Es decir, para el modo dos que intervienen los descriptores ORB y BRIEF, ambos los harán con un peso equitativo, esto es, del cincuenta por ciento cada uno. Y así con todos los descriptores locales.

En la figura 37, se puede analizar los pasos del Matching de descriptores locales entre la imagen de entrada y las candidatas. Se extraen los descriptores a través de los keypoints de la imagen de entrada (1) y lo mismo con imágenes candidatas (2). Proceso de Matching entre descriptores de la imagen de entrada y cada una de las candidatas (3), sumando todos los matching encontrados. Por último se obtiene el vector puntuación local con los valores finales (4).

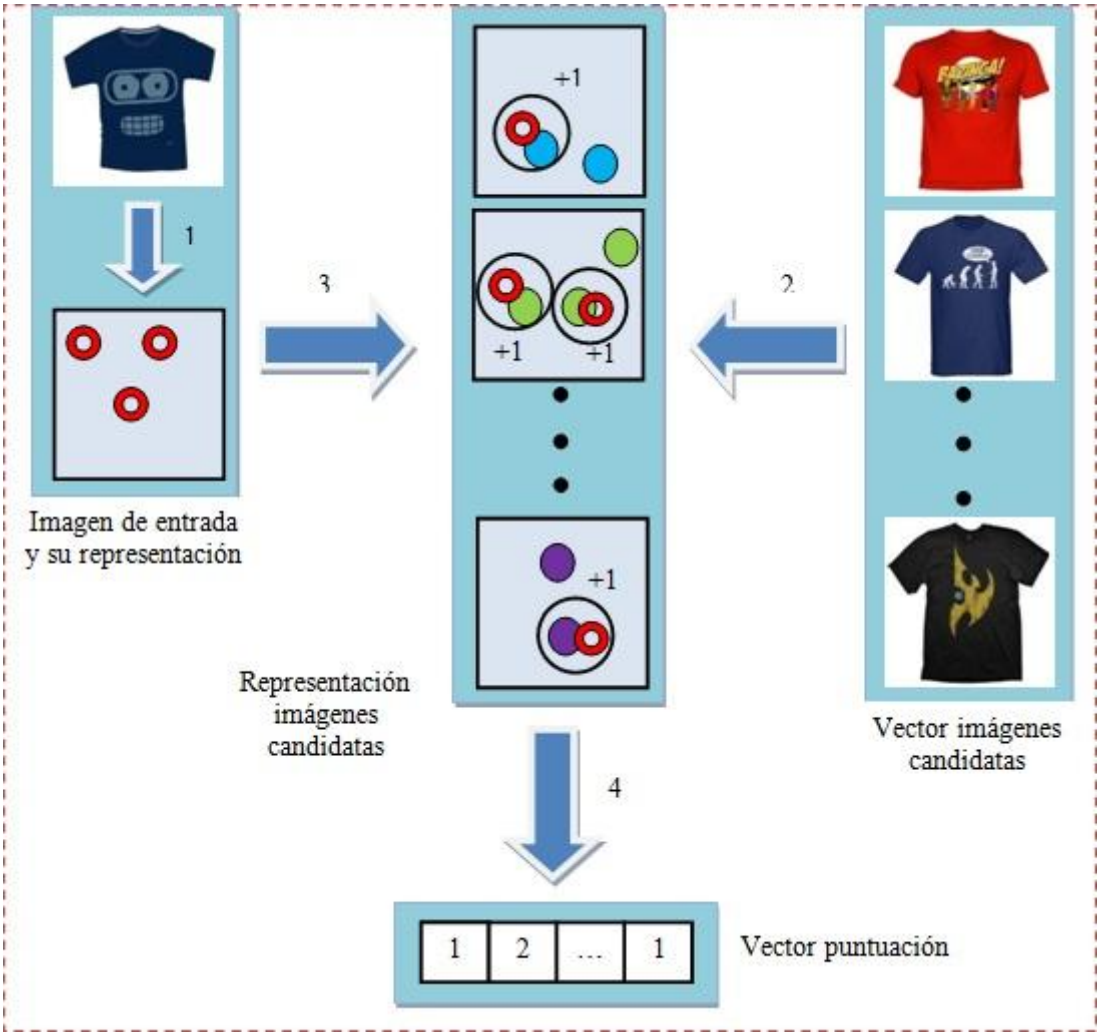


Figura 37: Matching entre imagen de entrada y candidatas con descriptores locales

3.6.2 Descriptores globales

A diferencia de los descriptores locales, sintetizan todas las características en un único vector. Contienen gran cantidad de información en un espacio reducido, por lo que su coste computacional es muy bajo. Son la alternativa a los descriptores anteriores cuando los objetos no tienen textura y su utilidad se reduce.

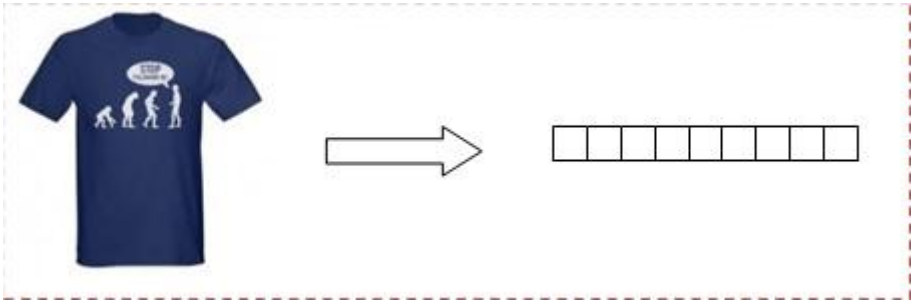


Figura 38: Representación en un único vector de los descriptores globales

A Histograma de color

De una manera aproximada a la anterior, el histograma de color de todas las imágenes candidatas serán comparadas con el histograma de color de la imagen de entrada.

La forma tradicional de representar el histograma de color de un determinado píxel es el sistema RGB. Como ya se vio anteriormente, representa la intensidad de los colores primarios en dicho píxel. Este sistema es muy sensible a cambios de escena por lo que no resulta de mucha utilidad en esta aplicación. Se debe buscar otro sistema. Para las necesidades de este proyecto se adapta mucho mejor el sistema HSV (Hue (Matiz), Saturation (Saturación), Value (Valor)). La Saturación y el Valor son más sensibles y propensos a cambios con la iluminación o la escena. El campo Hue representa el tono o matiz de un color y es más estable frente a cambios de escena, por lo que será el elegido para representar las imágenes. Se ha acotado el número de componentes del histograma a 25, para evitar que ligeras variaciones de matiz degraden las medidas de similitud. La columna más alta, nos indicará el matiz predominante en la imagen.

En la figura 39 se puede observar el histograma de color HSV para las distintas camisetas. En la primera se puede ver que el color predominante es el azul, mientras que en la segunda es el rojo.

Figura 39: Ejemplo histograma de color HSV

A la hora de la comparación, se necesita un valor que represente la similitud entre dos imágenes. Para ello se hace uso de las siguientes métricas:

Correlación: $d(H_1, H_2) = \frac{\sum_I (H_1(\mathrm{I}) - \mathrm{H}'_1)(H_2(I) - H'_2)}{\sqrt{\sum_I (H_1(I) - H'_1)^2 \sum_I (H_2(I) H'_2)^2}}$

Donde $H'_k = \frac{1}{N} \sum_J H_K(J)$

Y N es el número total de intervalos del histograma.

Chi-cuadrado: $d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$

Intersección: $d = (H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$

Bhattacharyya: $d = (H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{H'_1 H'_2 N^2}} \sum_I \sqrt{H_1(I) H_2(I)}}$

De igual modo que con los descriptores locales, el peso de los descriptores globales que intervengan en una ejecución lo harán en el mismo porcentaje. Es decir, para el modo dos que intervienen los descriptores con métrica Correlación y Chi-cuadrado, ambos los harán con un peso equitativo, esto es, del cincuenta por ciento cada uno. Y así con todos los descriptores globales, siempre de manera ecuánime entre ellos.

Dependiendo del modo elegido en la aplicación, según la figura 33 se aplicarán unas u otras métricas. De cualquier modo, todas las imágenes candidatas obtendrán un valor de similitud (error) con la imagen de entrada. De igual manera que sucedía con los descriptores locales, se logrará un vector puntuación. Con estos vectores puntuación se calculará el vector solución como sigue:

3.6.3 Vector solución

Tanto el vector puntuación de los descriptores locales como el vector puntuación de los descriptores globales serán utilizados para calcular la puntuación total de similitud entre las imágenes candidatas y la imagen de entrada. La fuerza del primer vector será del cuarenta por ciento mientras que la del segundo será del sesenta por ciento. Puesto que de esta manera se han obtenido los mejores resultados, ya que para los productos de este proyecto, la ropa, el color es un factor determinante. De ninguna manera es absoluto, ya que dos imágenes completamente distintas pueden ser iguales para el histograma. De ahí, el cuarenta por ciento de peso para los descriptores locales. De una manera similar, no todo el peso lo pueden llevar los descriptores locales puesto que para una imagen sin textura como ya se aproximó anteriormente, dichos descriptores pierden eficacia y gracias a los descriptores globales consiguen puntuar la imagen. Una vez calculado el vector con las puntuaciones totales, se mostrará al usuario los tres mejores resultados.

En la figura 40 se puede examinar lo anteriormente comentado, destacando el cálculo final entre los descriptores locales y globales con el porcentaje también visto anteriormente. Este proceso viene representado en la figura con un asterisco (*). A partir del vector solución ordenado, se mostrarán al usuario las imágenes que ocupen las tres primeras posiciones de la estructura, la solución a la imagen de entrada.

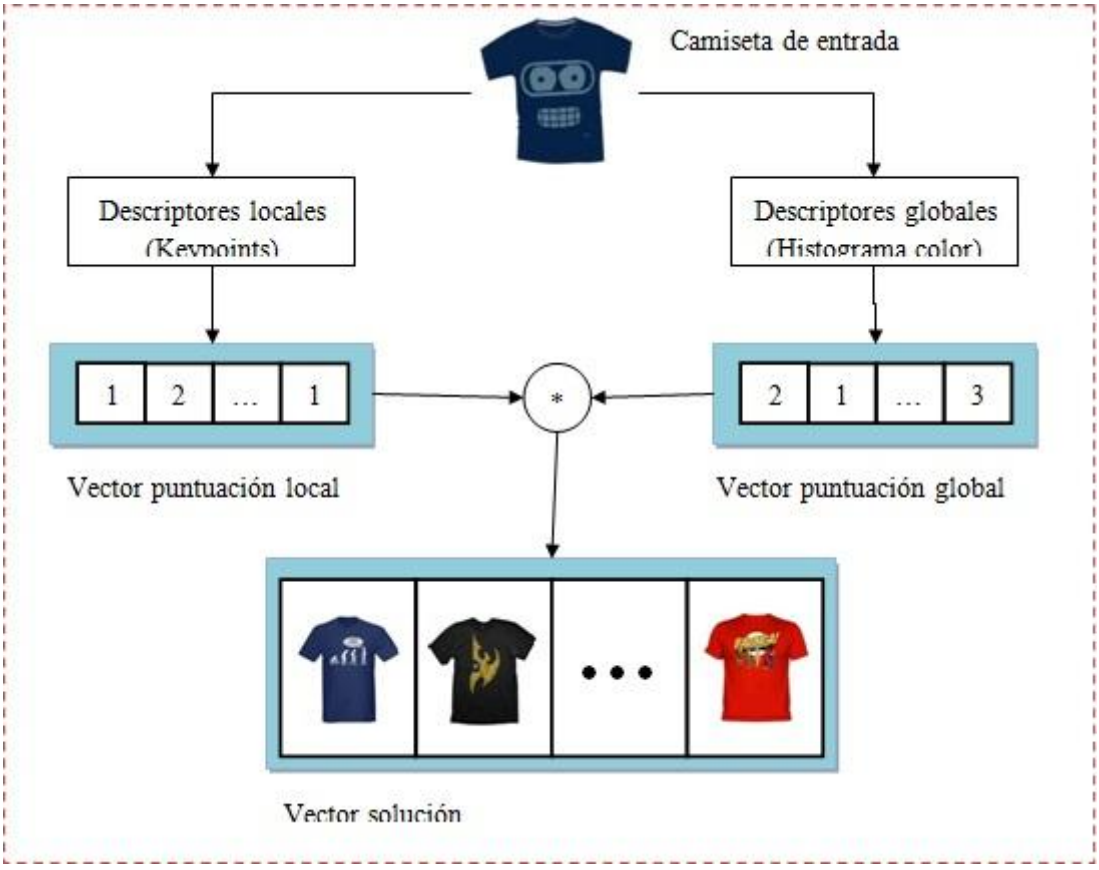


Figura 40: Esquema Matching

En cada uno de los modos de la aplicación, el peso de los descriptores locales que intervengan en una ejecución lo harán en el mismo porcentaje. Es decir, para el modo dos que intervienen los descriptores ORB y BRIEF, ambos los harán con un peso equitativo, esto es, del cincuenta por ciento cada uno. De igual manera para los descriptores globales. Una vez calculados los valores de los descriptores locales y globales, para realizar el cálculo final el peso de los descriptores globales será del sesenta por ciento, frente a un cuarenta por ciento de los descriptores locales.

3.6.4 Threshold del Matching

Para establecer el umbral se han realizado numerosas pruebas parecidas a la adjuntada en la sección 5. Cada nivel de la aplicación tendrá un threshold distinto, basado en la experiencia obtenida a través de los resultados. El motivo de establecer el threshold en un determinado lugar y no en otro, radica en el valor del error medio. A partir de cierto valor de error medio las imágenes resultantes, visualmente son peores que las que devuelven un error medio menor. Ser peor visualmente, significa que las tres mejores imágenes obtenidas para una imagen de entrada no se parecen demasiado y tienen un error medio elevado. Por lo tanto el threshold se establecerá en el valor que marca la frontera de los resultados visualmente peores y los mejores. El valor del threshold se conocerá en la prueba de la sección 5, donde mediante gráficas se podrá observar visualmente.

Una vez establecido el threshold, el sistema funcionará de la siguiente manera. Para una búsqueda cualquiera, la imagen original será comparada con la primera página de resultados, es decir 30 imágenes para Google y 100 para Zalando. Si el error medio devuelto por el matching de las tres mejores imágenes es mayor al threshold establecido, la aplicación buscará en la siguiente página para intentar conseguir un error por debajo del threshold. La aplicación seguirá buscando en las siguientes páginas, realizando el mismo proceso hasta que se superen las cuatro páginas de Google (donde los resultados devueltos son peores) o se acaben las páginas en Zalando. En ese caso, los resultados devueltos serán los mejores obtenidos hasta el momento.

3.7 Aplicación auxiliar de escritorio

Con la salida de la versión 2.4.4 de OpenCV, se puso por primera vez al alcance de los usuarios la versión para escritorio de Java portada desde Android. Aprovechando dicha biblioteca se trasladó el código desarrollado hasta el momento para el dispositivo móvil al ordenador de sobremesa para aprovechar su mayor capacidad de cómputo para entrenar los algoritmos de Segmentación y Matching vistos anteriormente. Una vez entrenados fueron portados de nuevo a la plataforma Android donde se realizaron las últimas pruebas.

La utilización de la biblioteca es similar en Android y en el escritorio, de hecho comparten la misma documentación lo que hizo tremendamente atractivo llevar a cabo los entrenamientos en el ordenador de una manera más eficaz y rápida sin tener que cambiar grandes cosas en el código.

Se han desarrollado dos aplicaciones se escritorio. La primera contiene la parte de la segmentación y la segunda la parte de búsqueda y matching.

La aplicación de segmentación de imágenes, está desarrollada para leer de una carpeta multitud de imágenes, hacerlas pasar por los algoritmos de segmentación y obtener los resultados directamente exportados en un documento de texto con su correspondiente desviación normalizada del k-means. La imagen original viene acompañada de una imagen que representa el k-means, de otra que representa la máscara a aplicar y por supuesto el resultado final. Todo esto es visible en los Apéndices de este documento, y que será ampliamente tratado en la sección 5. Con esta manera de proceder, se pueden entrenar a los algoritmos cambiando los parámetros de la segmentación y viendo los resultados en un espectro de imágenes suficientemente grande como para sacar resultados definitivos. Ver si mejora o empeora el resultado en la mayoría de las imágenes y poder concretar un cambio objetivamente.

En la aplicación de búsqueda y matching, la forma de proceder es similar a la anterior. Partiendo de la lectura de las imágenes ya segmentadas, y gracias a una descripción de cada prenda de vestir, se realiza la búsqueda en Google o Zalando. Se descargan las imágenes y se procede al matching entre la original y sus candidatas. El resultado es exportado directamente a un documento de texto que está formado por la imagen de entrada, es decir la imagen segmentada, el pódium obtenido, y el error medio. El pódium consta de las tres mejores

imágenes, las que tienen menor error. Y el error medio mostrado es la media de los errores de las imágenes del pódium. Al igual que en la aplicación de la segmentación, de esta manera podemos tomar decisiones de una manera objetiva y contrastada, ya que mejora la mayoría de las imágenes.

4. Uso de la aplicación

Uso de la aplicación

En este diagrama se puede observar el completo funcionamiento de la aplicación móvil. Empieza mostrando como se ve en la figura 0 del diagrama, el logotipo de la aplicación para pasar inmediatamente al primer menú de la aplicación en la figura 1 del diagrama, donde el usuario puede elegir si quiere tomar una foto con la cámara del dispositivo o seleccionar una imagen suya de la galería de imágenes (figuras 2 y 3 del diagrama). Una vez elegida, el usuario deberá recortar la imagen de tal manera que la prenda que quiera buscar ocupe la mayor parte posible, tal y como se muestra en la figura 4 del diagrama. En el mismo tiempo en el que el propio dispositivo recorta y guarda la imagen se procede a hacer un primer análisis de la misma pasándola por el algoritmo de k-means, que ofrece a la aplicación la desviación media obtenida; en el caso de que ésta sea excesivamente elevada y sobrepase el threshold establecido se rechazará la imagen ya que no será de utilidad para realizar un matching correcto, informando y pidiendo al usuario que intente realizar una imagen más precisa como se muestra en la figura 6 de diagrama. En el caso de que no sobrepase este umbral aparece una nueva pantalla donde se verá la imagen ya recortada y debajo de ésta una serie de campos donde elegir las características que pueda ayudar a delimitar la búsqueda.

Una vez especificadas todas las características de la imagen, se pulsa en el botón buscar y comienza el proceso de segmentación, búsqueda y matching que se ve en las figuras 8 9 y 10:

- Segmentación: se termina el proceso de segmentación iniciado en el punto 5.
- Búsqueda: se realiza una conexión a la web para descargar las imágenes de Google o Zalando.
- Comparación: una vez terminada la descarga se van comparando una a una con la imagen original.

Una vez terminado estos tres procesos se muestran los tres mejores resultados obtenidos con la imagen de la prenda y una pequeña descripción. Pulsando en las imágenes aparece una ventana pop-up donde se puede observar la imagen con un mayor tamaño. Para terminar pulsando en los botones con forma de etiqueta y la palabra BUY se lleva al usuario a la versión web donde se encuentra la prenda para su compra.

La descripción concreta de cada una de las diferentes etapas de la aplicación es la siguiente:

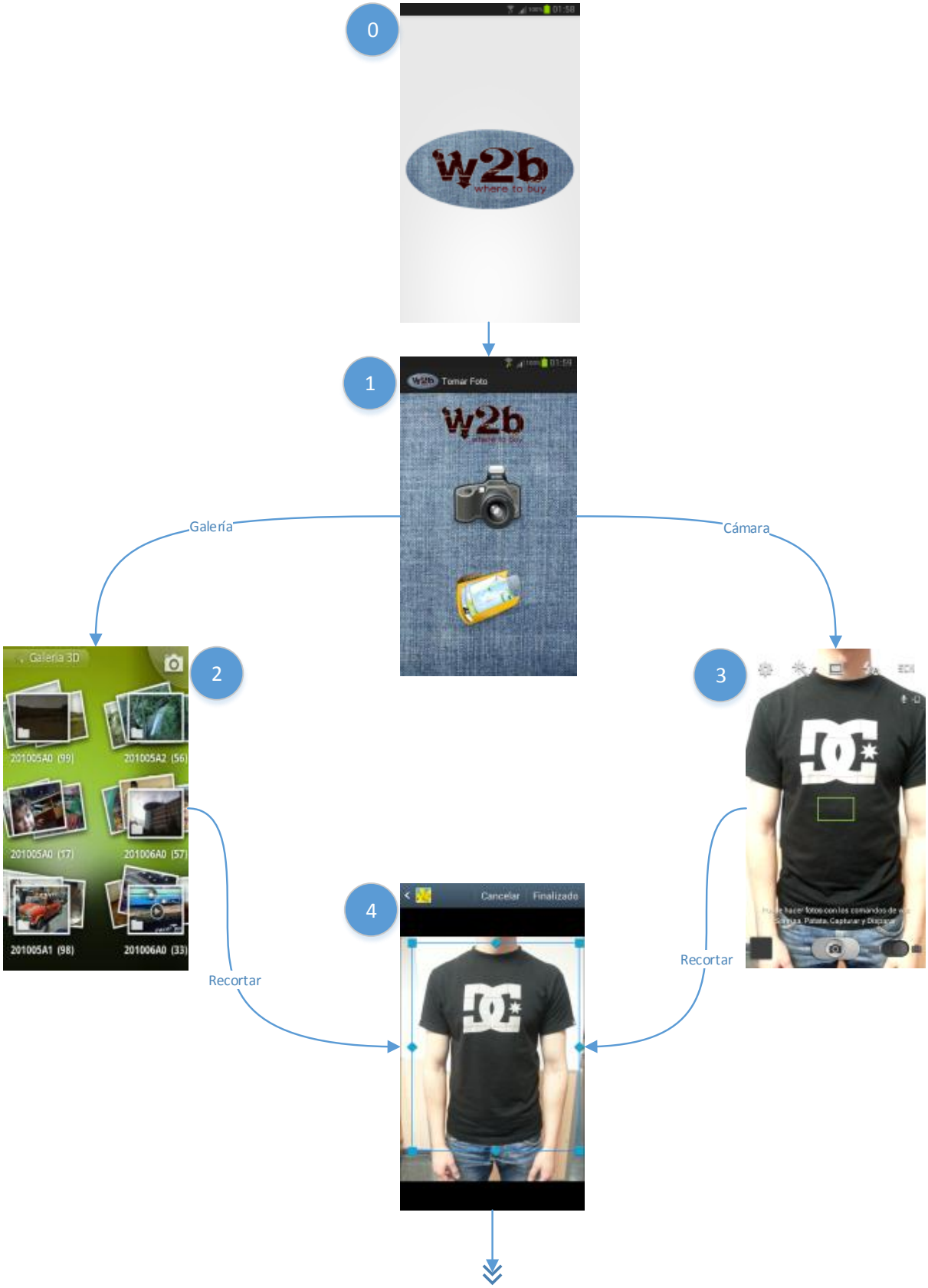
0. Inicialización mostrando el logotipo de la aplicación.
1. Selección de la foto. Se muestran dos botones:
    - Icono de una cámara de fotos, al pulsarlo pasara la etapa 3.
    - Icono que representa una galería de imágenes, al pulsarlo pasará a la etapa 4.
2. Se muestra la galería de imágenes del dispositivo. Una vez elegida la imagen se pasará a la etapa 4.

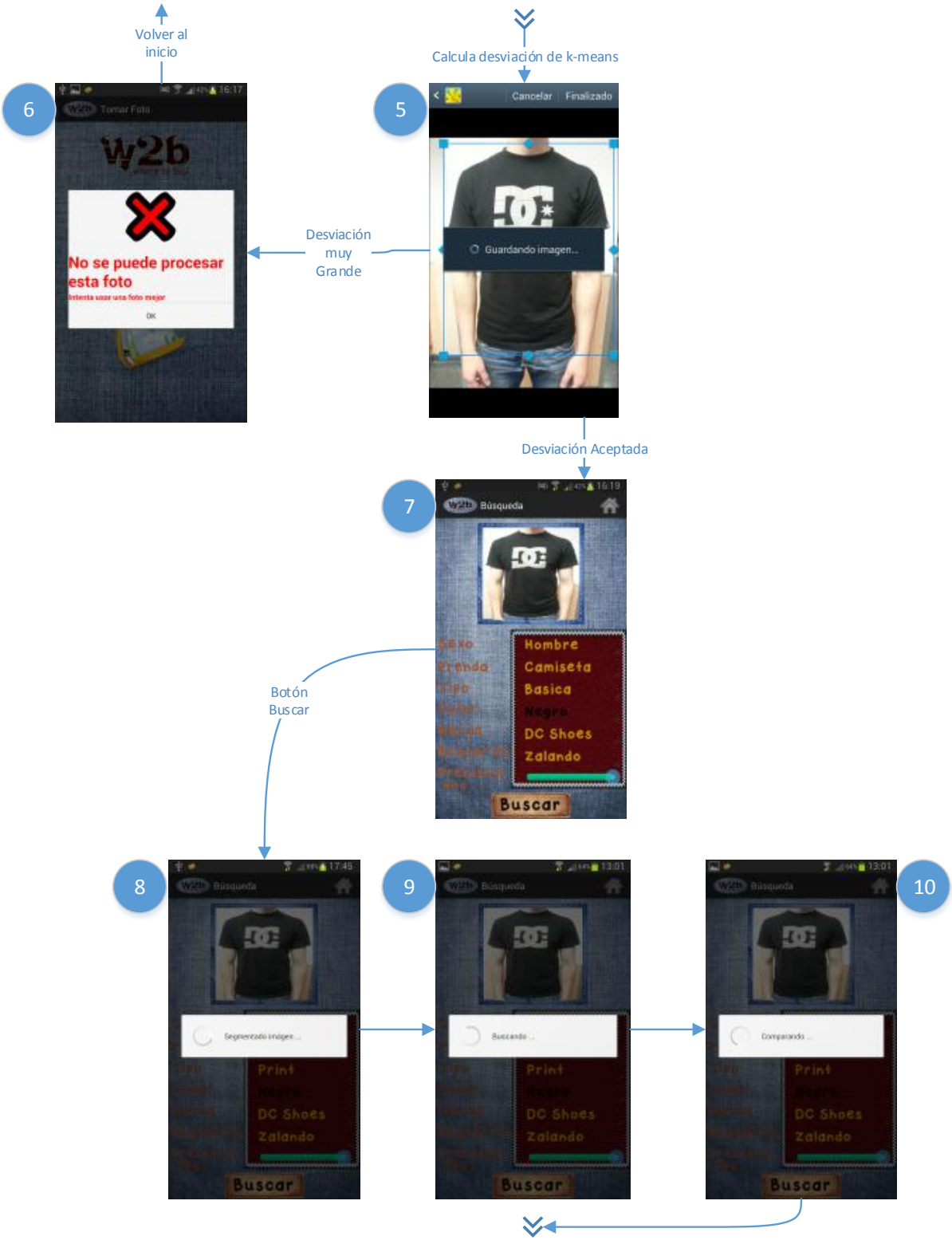
3. Se abre la aplicación de la cámara de fotos del dispositivo. Una vez tomada la imagen se pasará a la etapa 6. El usuario puede descartar imágenes hasta que haga la que más le gusta. Una vez elegida se pasará a la etapa 4
4. Se muestra una herramienta para recortar la imagen.
5. Una vez recortada la imagen, se detecta automáticamente si la imagen es buena para el propósito de la aplicación. En caso contrario se muestra la etapa 7. De no ser así se pasa a la etapa 6.
6. Se informa al usuario de que su imagen no es lo suficientemente buena. Volverá automáticamente a la etapa 1 donde empezara de nuevo el proceso.
7. Menú con todas las opciones de búsqueda útiles para acotar la búsqueda donde se pueden introducir los siguientes campos:
    - Sexo: hombre o mujer.
    - Prenda: dependiendo del campo anterior se rellenará automáticamente con prendas de mujer o de hombre.
    - Tipo: cada prenda tiene una serie de tipos específicos, este campo se rellanará automáticamente después de que el usuario haya especificado de que prenda se trata.
    - Color: el usuario podrá elegir de una colección de colores que son ofrecidos por la web Zalando.
    - Marca: una vez elegida el tipo de prenda la aplicación se conecta a la dirección donde se presentan éstas y busca que marcas están disponibles rellenando a continuación este campo.
    - Barra de precisión: sirve para poder elegir cuál de los tres modos de búsqueda se quiere usar. Baja, media o Alta.

    Excepto el campo para elegir la marca todos los demás son obligatorios, no dejando la opción al usuario de no usarlas.

8. Se muestra en la barra de progreso que se está segmentando la imagen.
9. Se muestra en la barra de progreso que se están buscando imágenes.
10. Se muestra en la barra de progreso que se está comparando la imagen original con las imágenes de la búsqueda. Al final de esta etapa se mostrará directamente la etapa 11.
11. Lista de los tres mejores resultados encontrados. De cada uno de ellos se muestra la imagen de la prenda, una pequeña descripción y botón en forma de etiqueta con el texto BUY. Pulsando esté botón se irá a la página web donde aparece esta prenda.
12. Página web del primer elemento de la lista de resultados.
13. Página web del segundo elemento de la lista de resultados.
14. Página web del tercer elemento de la lista de resultados.

Todas las activities de la aplicación cuentan con una barra de menú superior con la que el usuario puede navegar, volviendo a la pantalla anterior o al inicio de la aplicación.


0
1
Tomar Foto
Galería
Cámara
2
3
Recortar
Recortar
4
Cancelar
Finalizado


Volver al inicio
Calcula desviación de k-means
6
5
Cancelar
Finalizado
Guardando imagen...
No se puede procesar esta foto
Desviación muy Grande
Desviación Aceptada
7
Búsqueda
Hombre
Camiseta
Basica
Negro
DC Shoes
Zalando
Buscar
Botón Buscar
8
9
10
Print
DC Shoes
Zalando
Buscar


11
Resultados
DC Shoes - GALLANT - Camiseta print - negro
BUY
DC Shoes - STAR - Camiseta print - negro
BUY
DC Shoes - WETPAINT - Camiseta print - negro
BUY
Primer Resultado
Segundo Resultado
Tercer Resultado
12
Comprar
zalando
13
Comprar
14
Comprar
zalando

5. Resultados

En este punto de la memoria se van a mostrar una serie de pruebas reales realizadas con la aplicación, de las que se van a poder extraer las conclusiones pertinentes sobre el algoritmo y sus capacidades. Todas las pruebas se realizan sobre la aplicación auxiliar de escritorio por su mayor rapidez de cálculo y extracción de los datos y gráficas.

Se va a incluir un estudio completo, partiendo de un conjunto de imágenes que serán sometidas a los algoritmos de segmentación y matching vistos hasta ahora. El espacio de imágenes consta de 76 elementos (Apéndice A). Sólo los elementos que superen el threshold marcado para la segmentación pasarán a ser buscados por categoría en la web de prendas de ropa Zalando, obteniendo una base de datos con la que matchear la imagen correspondiente del espectro de prueba inicial.

En la siguiente figura se puede apreciar de una manera esquemática el proceso de segmentación por el que pasan las imágenes del espacio muestral, obteniendo sus respectivas desviaciones estándar para llegar a una conclusión a partir de la gráfica obtenida. Donde el eje horizontal representa el porcentaje de imágenes que se encuentran en los rangos definidos en el eje vertical, siendo este la desviación resultante del kmeans normalizada, para cada imagen.

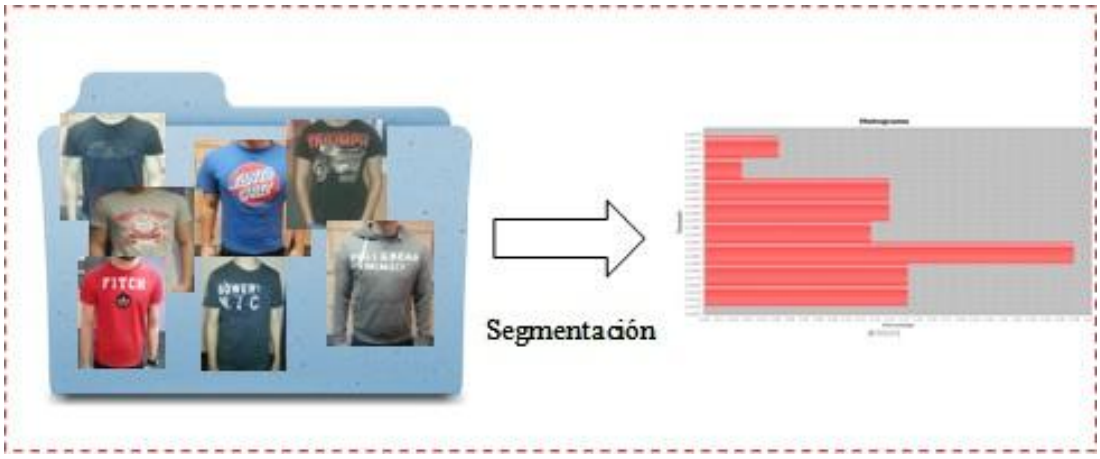


Figura 41: Detalle segmentación

En la figura 41, se puede apreciar como para cada una de las imágenes que pasaron de forma satisfactoria la segmentación, se va a realizar una búsqueda en la web Zalando teniendo en cuenta el color y categoría de la prenda. Para cada imagen ya segmentada se descargarán todas las imágenes de las que disponga Zalando, teniendo en cuenta los parámetros de búsqueda introducidos por el usuario. Esta etapa en la aplicación móvil se realiza de forma distinta como se ha explicado en el punto 3.5. El resultado del matching para cada imagen inicial serán las tres imágenes más similares, cuyo valor de error medio devuelto por el algoritmo será representado en el diagrama de barras oportuno. Donde las abscisas representan el porcentaje de imágenes que se encuentran en los rangos definidos en las

ordenadas, siendo éste el error medio de las tres mejores imágenes obtenidas en cada búsqueda.

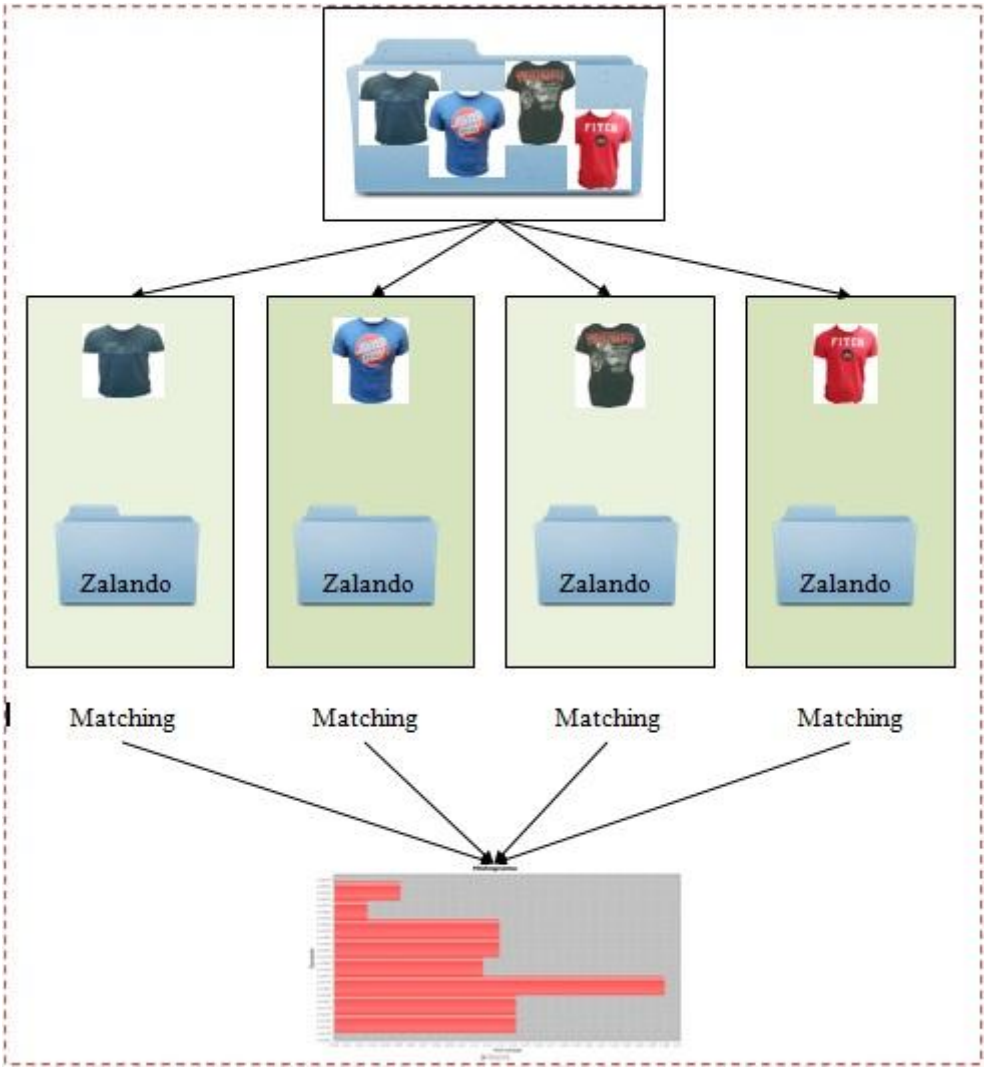


Figura 42: Detalle Matching

5.1 Resultados Segmentación

Todos los elementos del Apéndice A serán sometidos a los algoritmos de segmentación obteniendo los resultados visibles en el Apéndice B. Cada imagen candidata viene acompañada de una figura que muestra la imagen resultado tras el kmeans, la máscara a aplicar y el resultado final, además de la desviación media normalizada devuelta por el kmeans.

Con todas estas desviaciones se crea la siguiente gráfica. Dónde se puede observar la desviación normalizada que alcanza cada una de las 76 imágenes segmentadas. La media es 413,194, siendo la mayor desviación 678,56 y la menor 206,614.

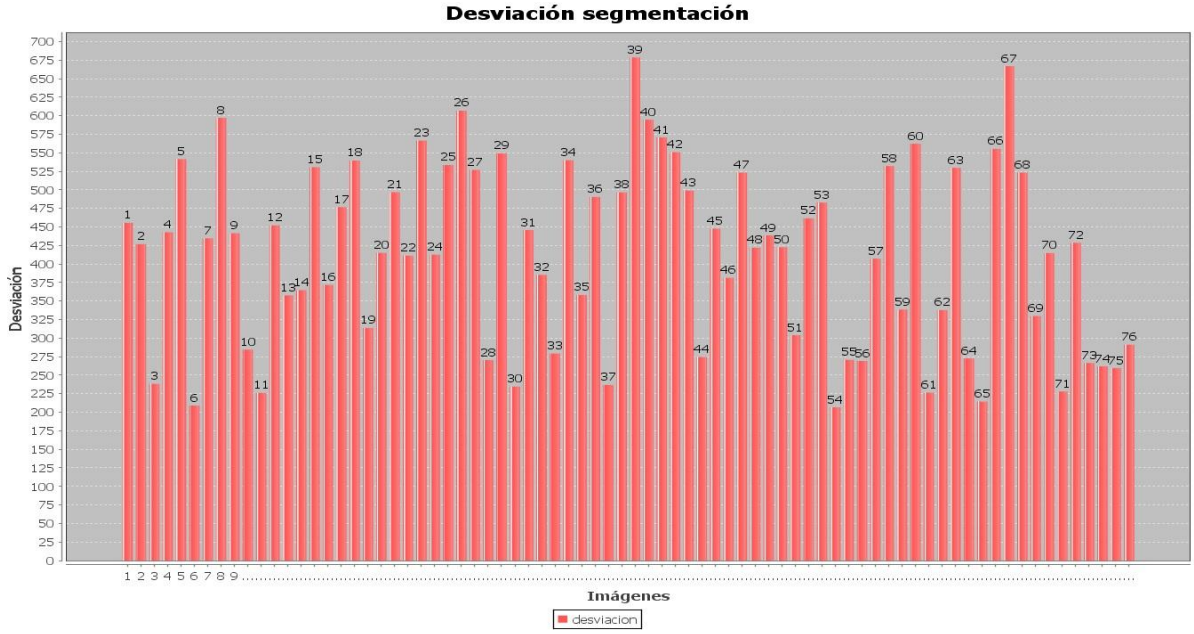


Figura 43: Desviación en la segmentación

También, a partir de las desviaciones se calcula el siguiente Histograma de distribución sesgada que se ve a continuación, donde los datos se concentran en la parte inferior de la distribución:

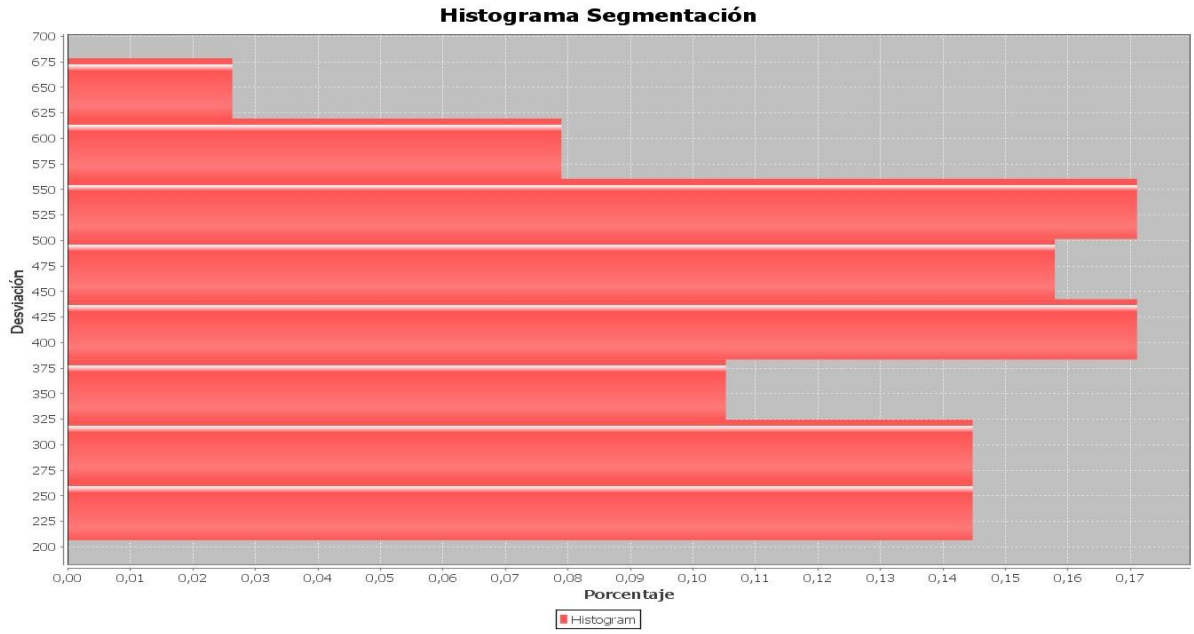


Figura 44: Histograma de la desviación en la segmentación

Mediante la observación de los resultados obtenidos de las segmentaciones y los porcentajes apreciables en la gráfica establecemos el threshold de la aplicación en 550, de tal manera el 10% de las imágenes serían rechazadas. Puesto que a partir de este umbral existen dos grupos de imágenes que representan el 8% y 2 % respectivamente. También apreciamos que un 25% de imágenes obtienen un resultado muy bueno, por debajo de 350 de desviación.

5.2 Resultados Matching – Nivel 3

Los elementos del Apéndice B que hayan superado el threshold establecido entrarán al proceso de Matching. En realidad, la aplicación está diseñada para que no pasen imágenes que estén por encima del threshold de segmentación, pero al ser un ejemplo y con ánimo de mostrar más resultados, se ha realizado el matching para todo el espectro de imágenes. A continuación observamos los resultados devueltos en las gráficas. Para un grado mayor de detalle ver Apéndice C.

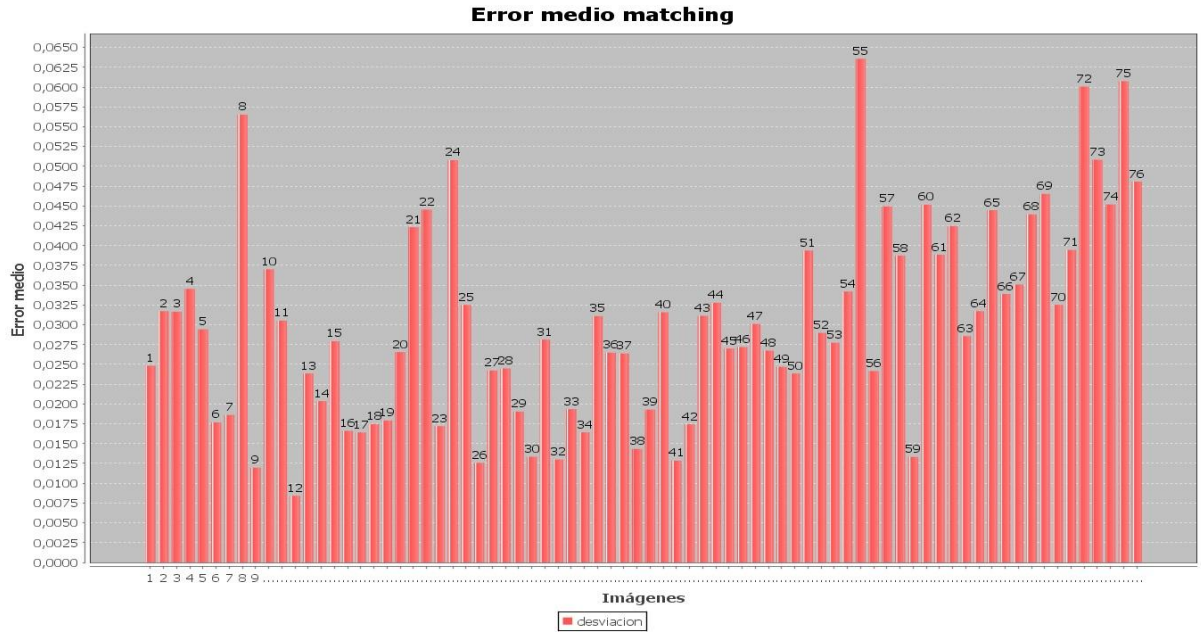


Figura 45: Error medio en el matching, nivel 3

En la figura 45, se puede apreciar el error medio de cada una de las imágenes de muestra. Cómo ya se ha comentado el error medio, es el formado por la media de los errores de las tres mejores imágenes candidatas para cada imagen de entrada. La media del error medio de la gráfica es de 0,0303, el mayor error tiene un valor de 0,064 y el menor error es de 0,008. Es decir, el mejor resultado obtenido tiene un error de 0,008 y el peor 0,064.

En la figura 46 se muestra el histograma obtenido a partir de los errores medios devueltos por el algoritmo de Matching en el nivel 3 de precisión. Se puede observar que el 27% de los resultados obtenidos presenta un error medio menor a 0,02 es decir que se han generado con imágenes muy similares a la original. El 50% de los errores medios son menores a 0,0275, lo cual también se considera aceptable, es este valor por tanto el que se elige como threshold del nivel 3. Por tanto el 50% de las imágenes quedarían por encima del threshold marcado lo cual

llevaría a la aplicación a realizar otra iteración buscando más imágenes candidatas en la siguiente página de Google o Zalando.

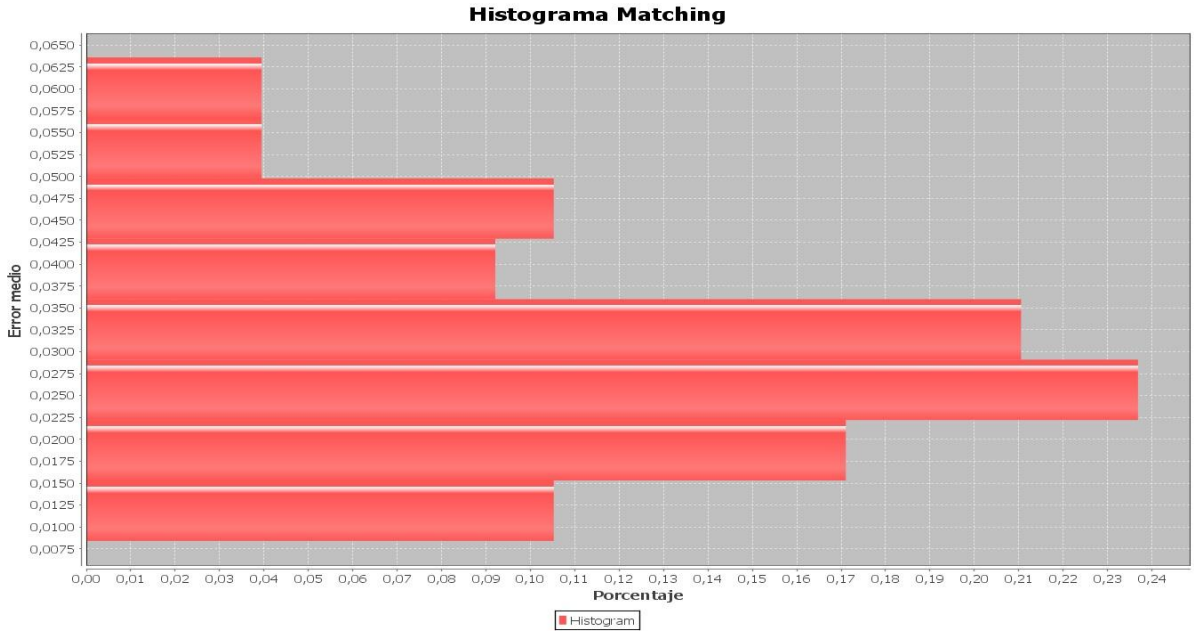


Figura 46: Histograma del error medio en el matching, nivel 3

5.3 Resultados Matching – Nivel 2

Para este nivel se procederá de manera análoga al nivel tres, cogiendo las imágenes ya segmentadas del Apéndice B y mostrando los resultados en el Apéndice D. Con los errores medios obtenidos se han construido las siguientes gráficas.

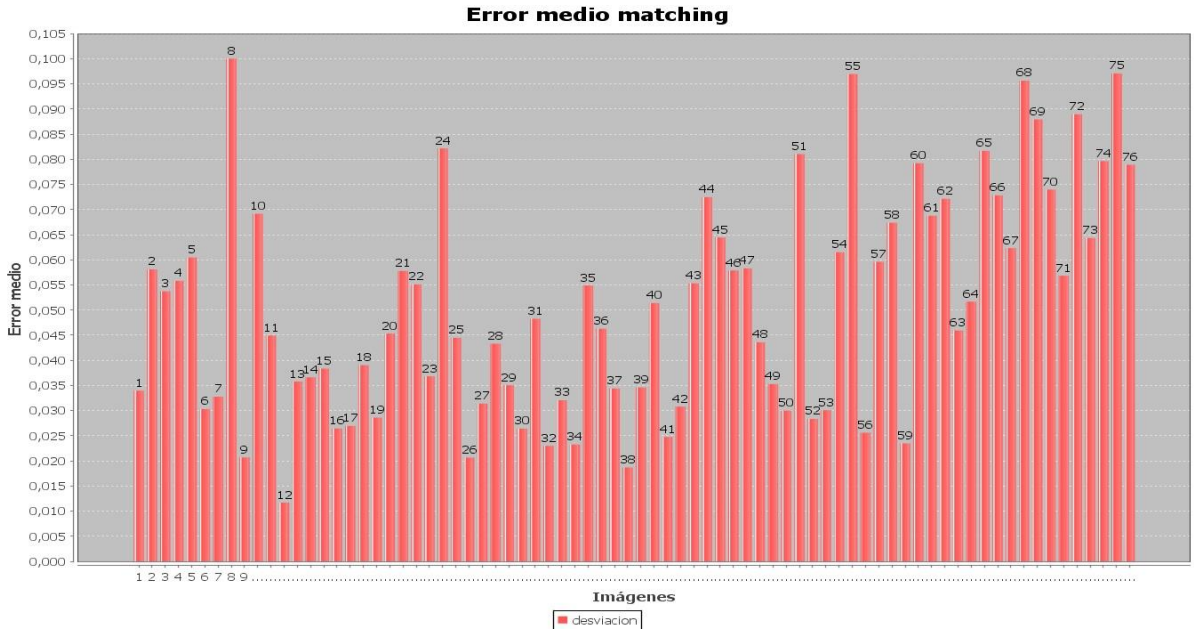


Figura 47: Error medio del matching, nivel 2

En la figura 47, la media del error medio es de 0,0507, bastante mayor al obtenido en el nivel 1. Lo que pone de manifiesto que el nivel 1 arroja mejores resultados que el nivel 2. El mayor error tiene un valor de 0,1 y el menor error es de 0,012. El mayor error y el menor también están por encima de sus homónimos en el nivel 1.

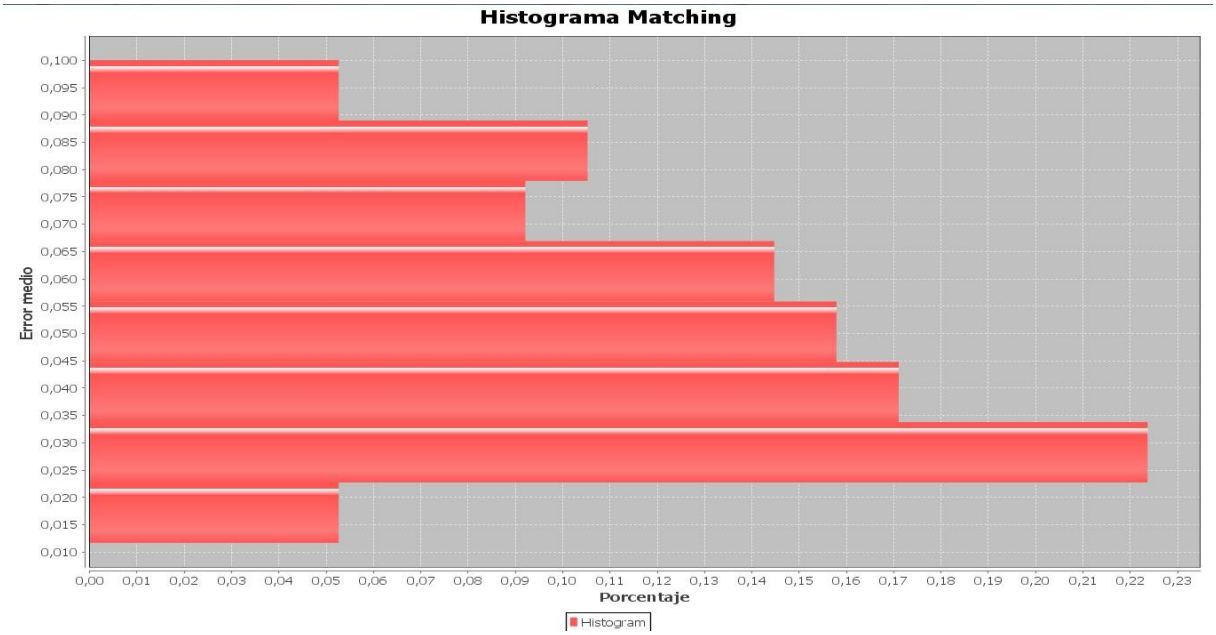


Figura 48: Histograma del error medio en el matching, nivel 2

En la figura 48 se aprecia el histograma obtenido a partir de los errores medios devueltos por el algoritmo de Matching en el nivel 2 de precisión. Se puede observar que el 44% de los resultados obtenidos presenta un error medio menor a 0,045 es decir que se han generado con imágenes muy similares a la original para este nivel. El threshold para este nivel se establecería en dicho valor. Por lo tanto, el 56% de las imágenes volverían a ser matcheadas contra la siguiente página de imágenes candidatas.

5.4 Resultados Matching – Nivel 1

Al igual que en el nivel dos y tres, el uno captará las imágenes del Apéndice B y producirá los resultados del Apéndice E. Éste nivel es el menos preciso, pero el más rápido, y produce resultados algo peores que los niveles anteriores pero nada desencaminados comparando el potencial algoritmo de los otros con respecto a éste.

En la figura 49 se puede observar los errores medios para cada una de las imágenes del Apéndice B. La media del error medio asciende a 0,0824, superior al nivel dos y obviamente al nivel tres de precisión. El valor más elevado de error medio es de 0,237, muy por encima de los niveles anteriores. Pero curiosamente el valor menor de error medio es de 0,001, muy por debajo de los niveles dos y tres. Puesto que sólo se utiliza en este nivel un descriptor de

imágenes como ORB, y un histograma con métrica Correlación, que son bastante potentes, pueden llegar a conseguir un matching tan destacable como el que se ha tratado.

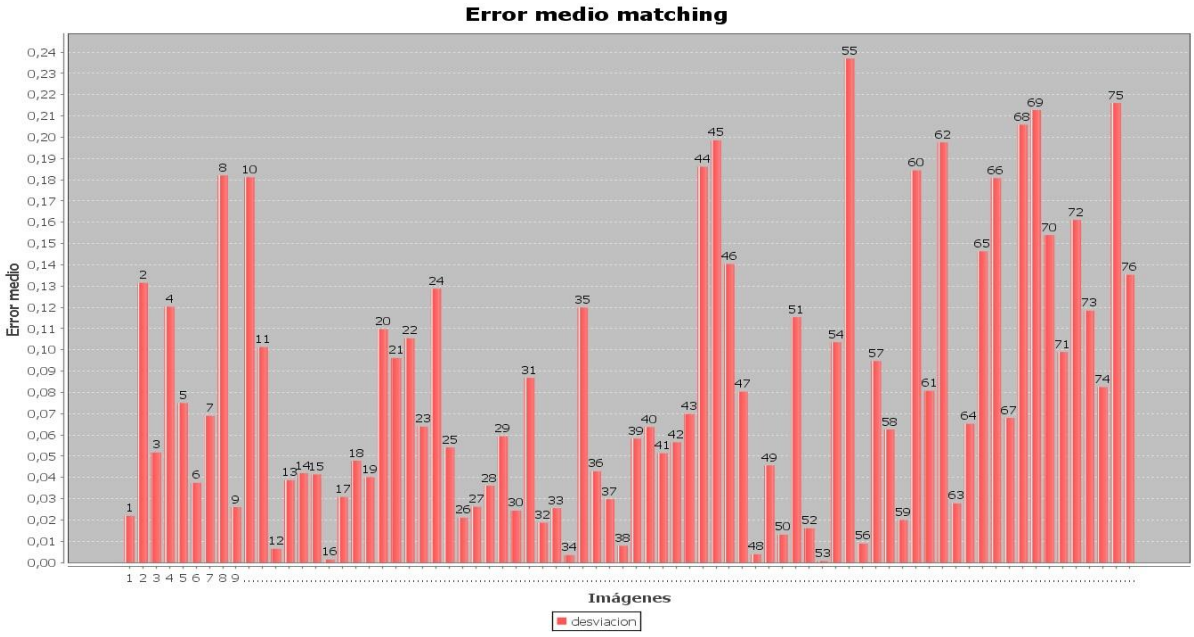


Figura 49: Error medio del matching, nivel 1

En la figura 50, se aprecia que el 46% de las imágenes están por debajo de 0,055 de error medio. Así que el threshold para este nivel se establecerá en 0,055 y el 54% de las imágenes volverán a iterar en la aplicación, completando el matching con otra página de resultados.

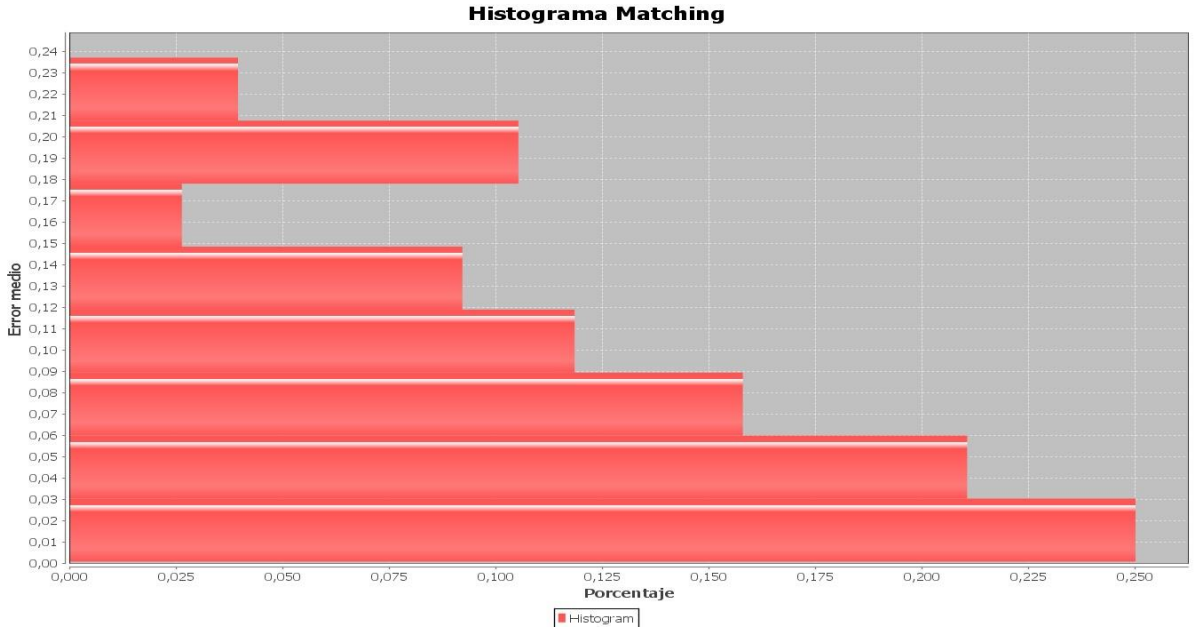


Figura 50: Histograma del error medio en el matching, nivel 1

A grandes rasgos, el threshold de la segmentación se establece en 550 de desviación, aceptando el 90% de las imágenes. Para el matching, el nivel tres tendría un threshold en

0,0275 de error medio dejando pasar al 50% de las imágenes. En el nivel dos, serian aceptadas el 44% de las imágenes con un threshold en 0,045 y por último para el nivel uno con un threshold en 0,055 de error medio se aceptarían el 46% de las imágenes. En la siguiente tabla se puede observar estos conceptos, junto con el tiempo medio de ejecución de los algoritmos de matching y la tasa de aciertos por eficiencia en cada nivel.

Modo o nivel	Error medio máximo	Error medio mínimo	Media error medio	Threshold	Imágenes aceptadas (%)	Tiempo medio (s)	Aciertos * eficiencia
3	0,0303	0,064	0,008	0,0275	50	22,709	862,942
2	0,1	0,012	0,0507	0,045	44	13,450	443,85
1	0,237	0,001	0,0824	0,055	46	12,995	454,825

Figura 51: Resumen resultados matching

En la figura 52, está representada la exactitud frente a la eficiencia para los tres modos de matching existentes. En el eje de abscisas se refleja la eficiencia, el tiempo de ejecución de los algoritmos de matching, frente a la exactitud en el eje de ordenadas, es decir la media del error medio en cada modo. En definitiva, la gráfica muestra como a mayor tiempo, menor error medio. El tiempo y la eficiencia son inversamente proporcionales. El modo tres es el que más tiempo consume pero el que mejores resultados devuelve. El modo uno es el más rápido, el que menos tiempo consume y por tanto el que mayor error medio consigue. El modo dos, es un modo intermedio entre los dos modos comentados.

La diferencia del modo dos con respecto a los otros dos no es proporcional, sin duda se encuentra mucho más cercana al modo uno que al tres. Esta diferencia reside en el número de descriptores e histogramas utilizados. Como ya se vio en la tabla de la figura 33, el modo uno, utiliza un descriptor y un histograma, el modo dos, dos descriptores y dos histogramas y el modo tres, cuatro descriptores y cuatro histogramas. La diferencia de número de descriptores e histogramas utilizados entre los modos tres y dos es mayor a la de los modos tres y uno, de ahí esa diferencia en la representación gráfica, donde el error medio aumenta y el tiempo disminuye cuanto menos descriptores e histogramas utilicemos.

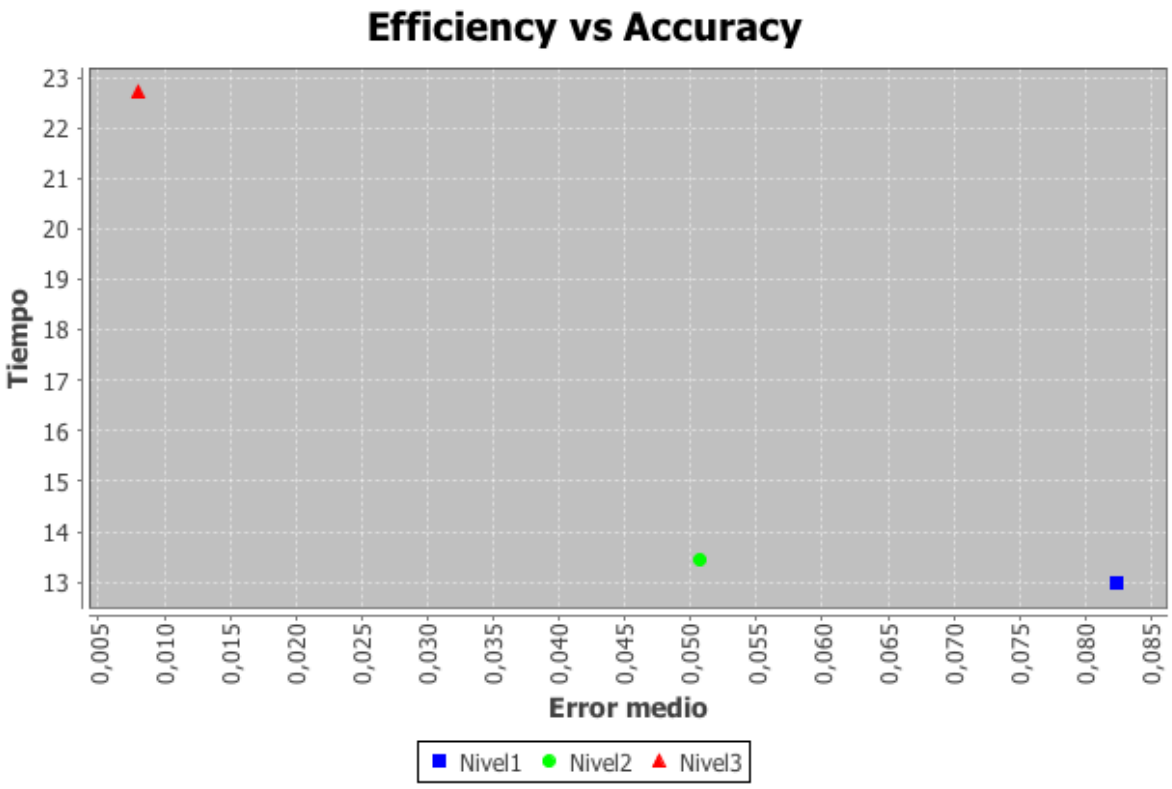


Figura 52: Efficiency vs Accuracy matching

6. Conclusiones y trabajos futuros

6.1 Conclusiones

El desarrollo de este proyecto es un ejemplo de que los procesos de segmentación y matching son campos del tratamiento de imágenes muy amplios, de resultados muy variables y dependientes de gran cantidad de factores. Obtener resultados aceptables en cada una de las fases ha supuesto un trabajo de experimentación para encontrar los valores adecuados para las distintas variables intervinientes en las distintas funciones utilizadas.

A lo largo del desarrollo de este proyecto varios problemas han sido solventados, desde el tiempo de ejecución pasando por la elección de los algoritmos de tratamiento de imagen hasta la búsqueda de una base de datos de imágenes candidatas que cumpliese los requerimientos de la aplicación, como se ha mencionado anteriormente. La toma de decisiones ha sido clave para encaminar el proyecto en la dirección finalmente elegida, ya que al ser, tanto el desarrollo de aplicaciones móviles como el tratamiento de imágenes, campos muy amplios, definir el punto de encuentro entre ambos mundos a pesar de ser un proceso complicado, se ha conseguido de forma satisfactoria.

Se ha conseguido segmentar de una manera satisfactoria alrededor del 90% de imágenes propuestas, de este conjunto de imágenes se ha conseguido matchear de manera correcta alrededor del 40%. Pudiendo estos resultados verse mejorados con el uso de terminales de altas prestaciones, con mayor rendimiento y con el acceso a una base de datos de imágenes adecuada.

El mayor problema con el que nos hemos encontrado es el tiempo de ejecución de la aplicación, si bien es cierto que en terminales de altas prestaciones estos tiempos son asumibles. La etapa que más tiempo consume es la de búsqueda de imágenes candidatas y su correspondiente descarga al dispositivo para su comparación. Esta descarga depende principalmente de la velocidad de conexión, por tanto es aconsejado el uso de redes Wi-fi. Las etapas de segmentación y Matching consumen un tiempo similar siendo el del matching superior, sobre todo en el nivel más alto de precisión.

Todos los detectores de características de imágenes tratados en este proyecto acarrean consigo la dependencia de la orientación de la imagen tomada y los histogramas dependen mucho del buen trabajo realizado por estos detectores puesto que dos imágenes totalmente distintas pueden tener el mismo histograma. A mayor número de detectores de características e histogramas utilizados, mayor será el tiempo de ejecución de los algoritmos y menor el error medio obtenido.

Los resultados obtenidos para esta aplicación podrían verse mejorados exponencialmente contando con una base de datos con la que comparar las imágenes adaptada a nuestras necesidades. Un ejemplo claro de esta mejora es la diferencia de resultados obtenidos haciendo uso de Google o Zalando para la búsqueda de imágenes candidatas. Donde

utilizando Zalando los resultados son mucho mejores, y si contáramos con una base de datos ideal, donde se encontraran todas las imágenes sobre las que realizamos búsquedas el acierto sería del 100%.

Se concluye por tanto, que el proyecto cumple con los objetivos marcados en un principio, y es un buen punto de partida para aplicaciones de este género, admitiendo ampliaciones en el terreno del tratamiento de imágenes, y procesamiento móvil.

6.2 Trabajos futuros

Para abordar el problema, comentado en el punto anterior, del tiempo la biblioteca OpenCV sobre la que se ha realizado este proyecto cuenta con un módulo de CUDA con la que paralelizar en GPUs la aplicación y así aprovechar las capacidades computacionales de la GPU. Actualmente la mayoría de los dispositivos móviles no cuentan con una GPU que acepte este tipo de paralelización, la cual disminuiría considerablemente el tiempo de segmentación y matching al poder aplicar esta técnica en estos procesos.

Actualmente las etapas de búsqueda y matching son procesos que se realizan de forma secuencial y una manera de mejorar el tiempo de ejecución sería realizarlas de manera paralela mediante el uso de hilos, mientras un hilo se encarga de la descarga de imágenes otro recibiría estas para procederá su matcheo.

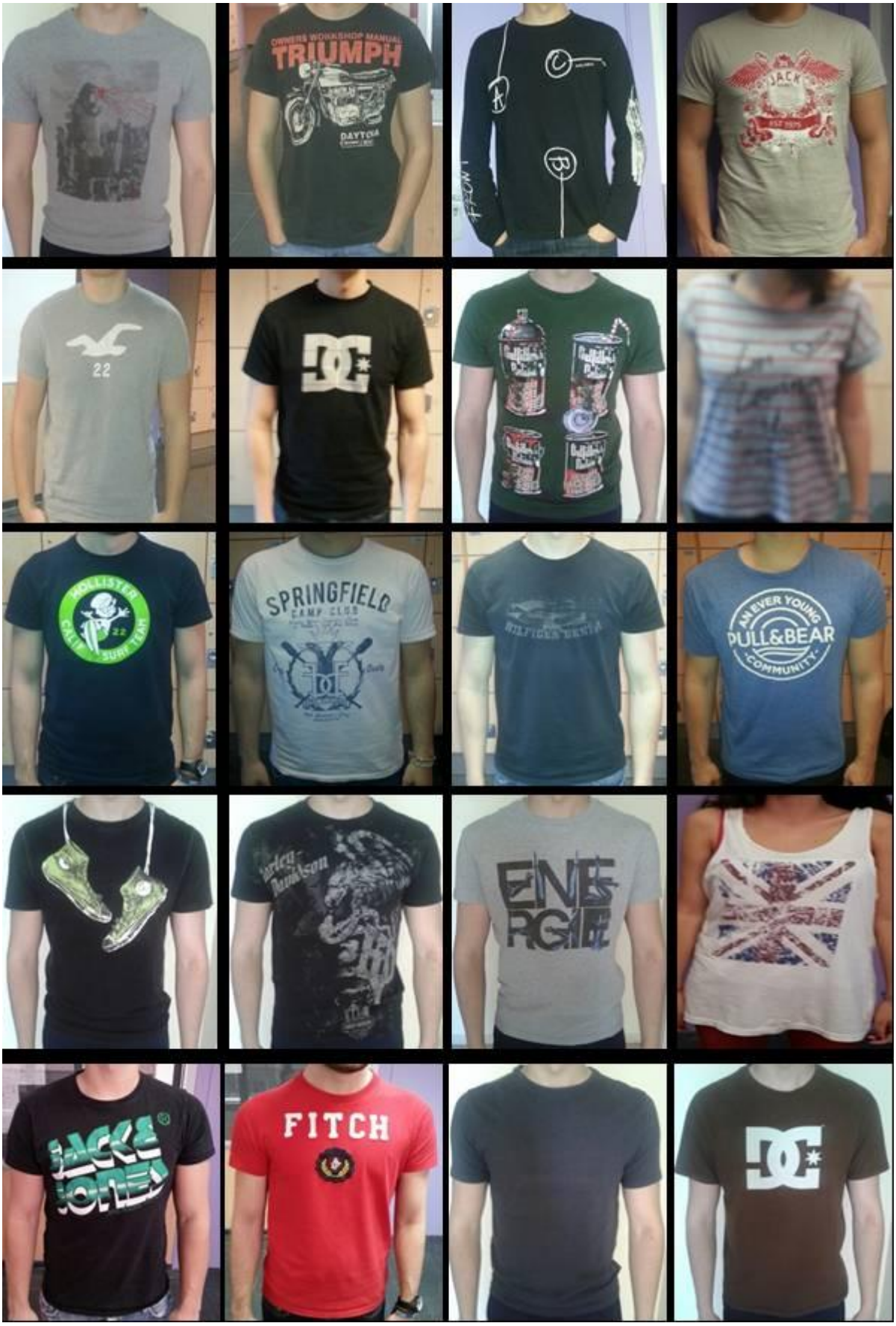
Si se consiguieran solucionar los problemas anteriormente mencionados, esta aplicación podría tener una salida comercial llegando a acuerdos con distintas tiendas de ropa online, al tener la aplicación acceso a una base de datos cerrada y controlada de imágenes.

Actualmente ninguna aplicación es capaz de segmentar todo tipo de objetos sin información adicional por lo que abordar este problema es un gran reto.

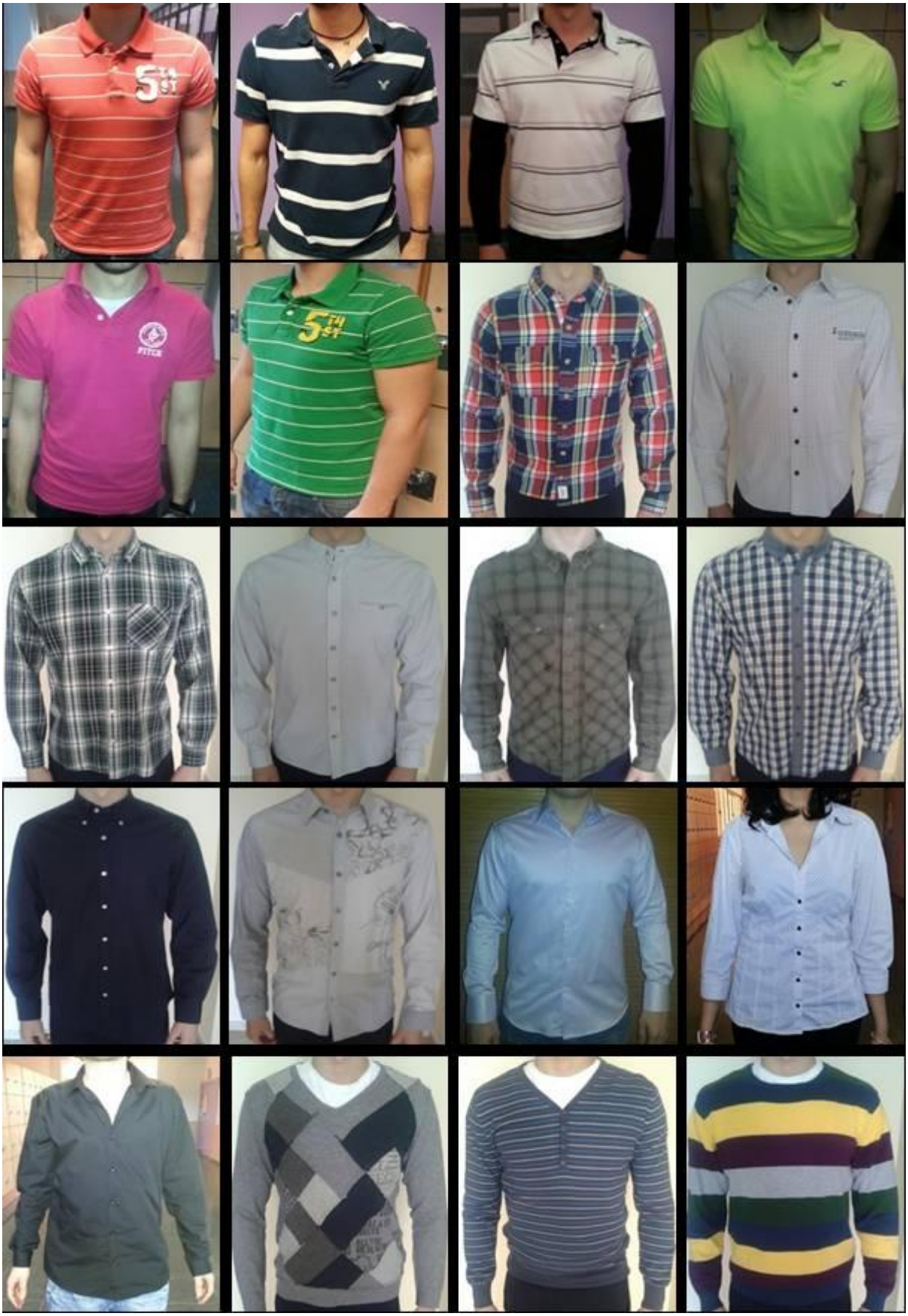
Teniendo en cuenta todo lo anterior, si se realizara una aplicación en la que introduciendo una imagen o parte de esta que formase parte de la base de datos

La finalidad de esta aplicación es la de reconocimiento de prendas de vestir, pero podría ampliarse a más objetos, fijándonos en las características visuales de cada uno y seleccionando nuevos atributos discriminantes para cada pixel, el uso de la aplicación puede extenderse a más ámbitos.

Apéndice A


TRIUMPH
DAYTONA
22
HOLLISTER
SPRINGFIELD
CAMP CLUB
AN EVER YOUNG
PULL&BEAR
COMMUNITY
FITCH

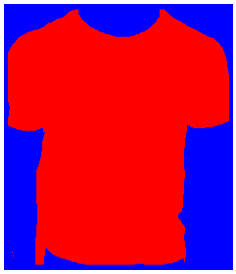
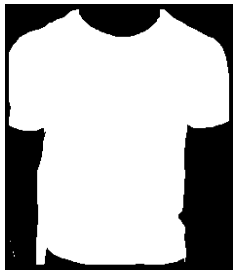

Times Sq
St Station
SANTA CRUZ
THE BEATLES
BOWERY
N.Y.C
LEGENDARY
MOTOR
HARLEY-DAVIDSON
CYCLES
MOTORCYCLES
Allstate




PULL&BEAR
MCMXCI
Sioux

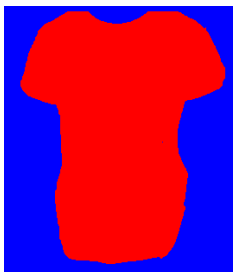
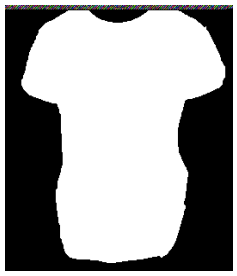
Apéndice B

Imagen 1: **K-means** **Máscara** **Final**


Desviación: 455,508

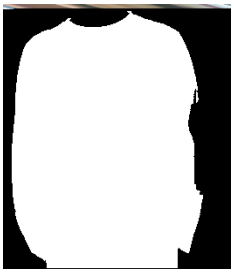
Imagen 2: **K-means** **Máscara** **Final**


Desviación: 426,696

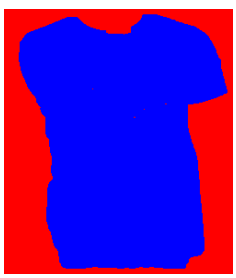
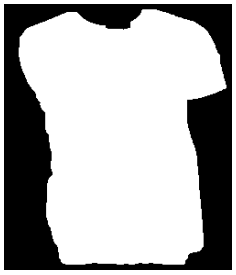
Imagen 3: **K-means** **Máscara** **Final**





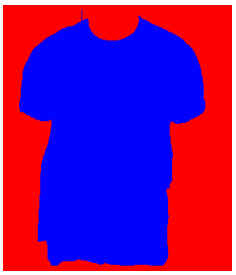
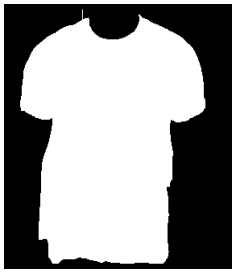

Desviación: 238,403

Imagen 4: **K-means** **Máscara** **Final**


Desviación: 442,722

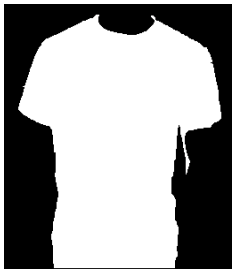
Imagen 5: **K-means** **Máscara** **Final**


Desviación: 541,442

Imagen 6: **K-means** **Máscara** **Final**

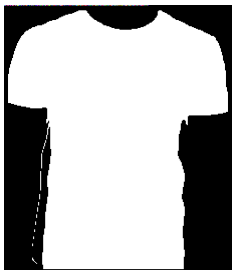





Desviación: 209,079

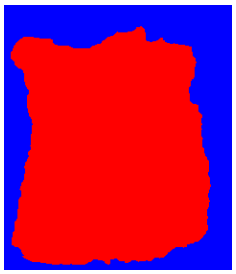
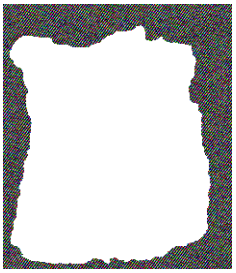
Imagen 7: **K-means** **Máscara** **Final**






Desviación: 434,687

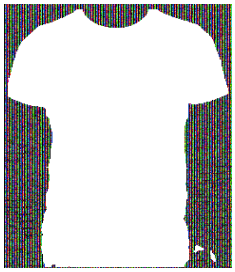
Imagen 8: **K-means** **Máscara** **Final**


Desviación: 597,129

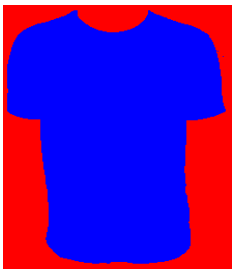
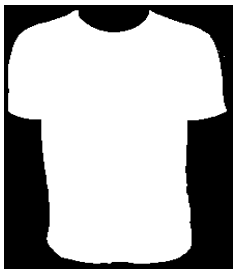
Imagen 9: **K-means** **Máscara** **Final**





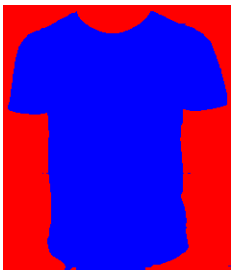
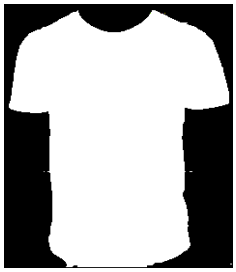

Desviación: 441,349

Imagen 10: **K-means** **Máscara** **Final**

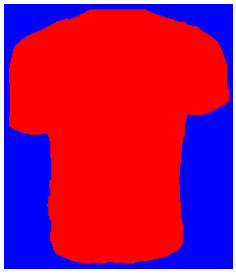
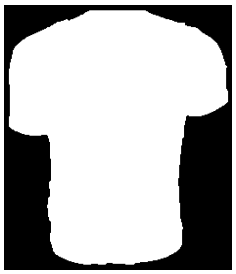

Desviación: 284,511

Imagen 11: **K-means** **Máscara** **Final**


Desviación: 226,056

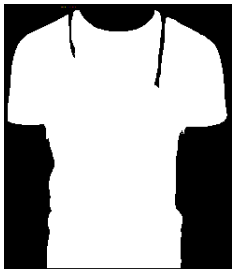
Imagen 12: **K-means** **Máscara** **Final**


Desviación: 452,096

Imagen 13: **K-means** **Máscara** **Final**

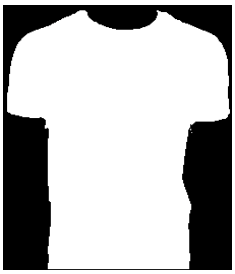





Desviación: 357,581

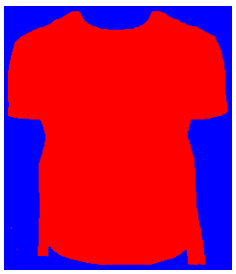
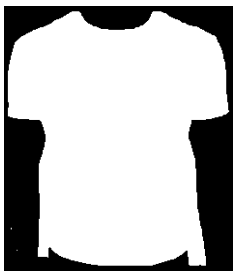
Imagen 14: **K-means** **Máscara** **Final**





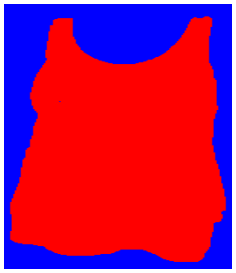
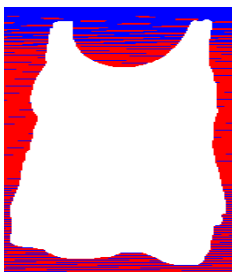

Desviación: 364,656

Imagen 15: **K-means** **Máscara** **Final**


Desviación: 530,501

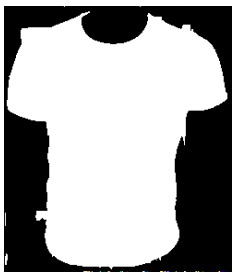
Imagen 16: **K-means** **Máscara** **Final**


Desviación: 371,724

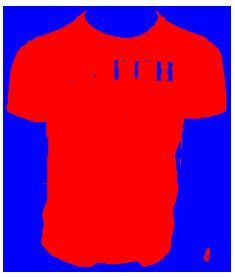
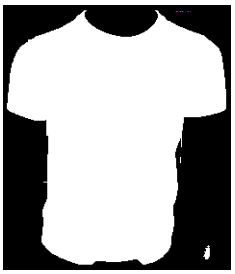
Imagen 17: **K-means** **Máscara** **Final**





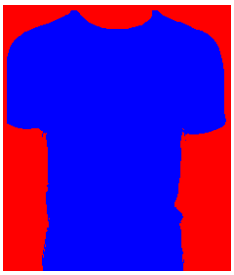

Desviación: 476,64

Imagen 18: **K-means** **Máscara** **Final**


Desviación: 539,513

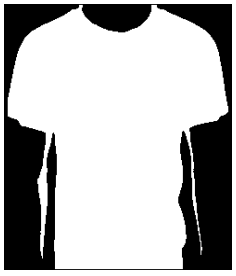
Imagen 19: **K-means** **Máscara** **Final**

Desviación: 313,806

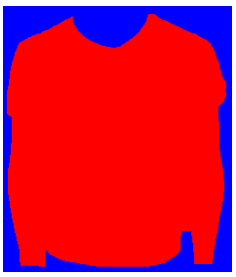
Imagen 20: **K-means** **Máscara** **Final**





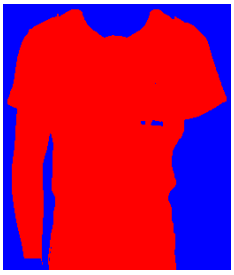
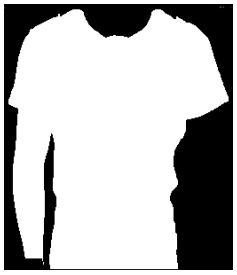

Desviación: 415,034

Imagen 21: **K-means** **Máscara** **Final**

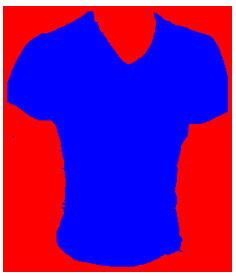
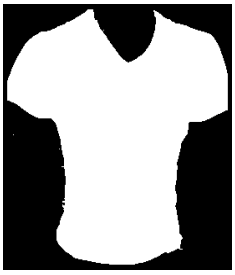
Desviación: 496,791

Imagen 22: **K-means** **Máscara** **Final**

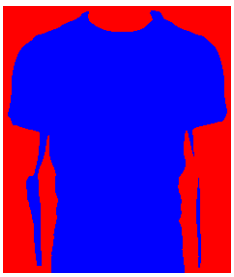
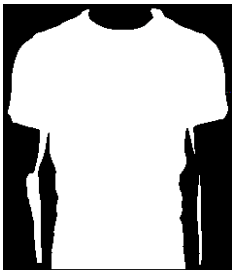

Desviación: 411,184

Imagen 23: **K-means** **Máscara** **Final**

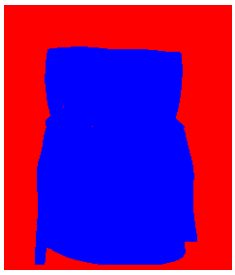

Desviación: 566,345

Imagen 24: **K-means** **Máscara** **Final**

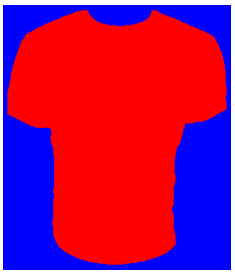
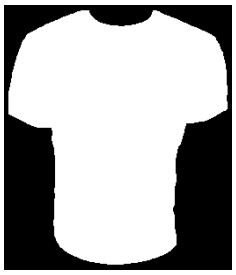

Desviación: 412,632

Imagen 25: **K-means** **Máscara** **Final**

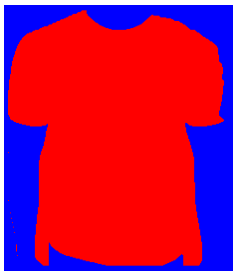
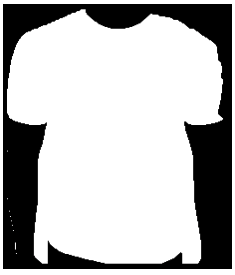
Desviación: 533,539

Imagen 26: **K-means** **Máscara** **Final**


Desviación: 606,801

Imagen 27: **K-means** **Máscara** **Final**

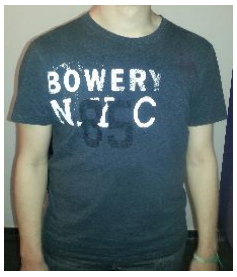

Desviación: 526,81

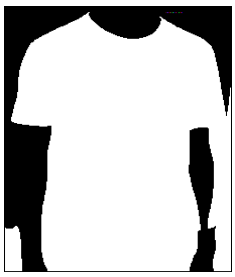
Imagen 28: **K-means** **Máscara** **Final**




Desviación: 270,301

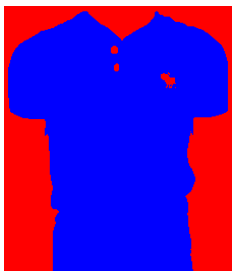
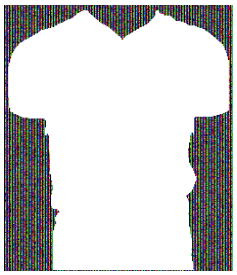
Imagen 29: **K-means** **Máscara** **Final**






Desviación: 549,218

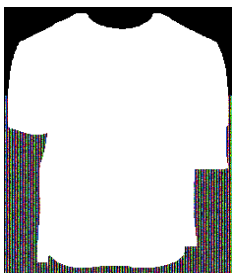
Imagen 30: **K-means** **Máscara** **Final**


Desviación: 234,635

Imagen 31: **K-means** **Máscara** **Final**

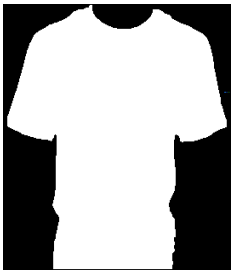





Desviación: 445,71

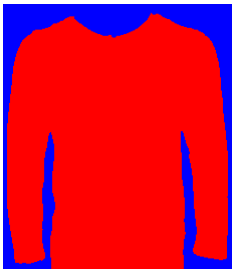
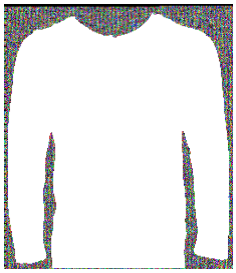
Imagen 32: **K-means** **Máscara** **Final**





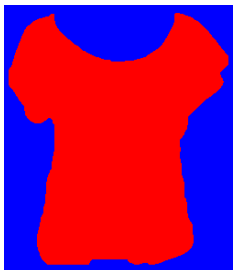

Desviación: 385,198

Imagen 33: **K-means** **Máscara** **Final**

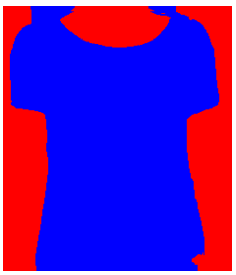
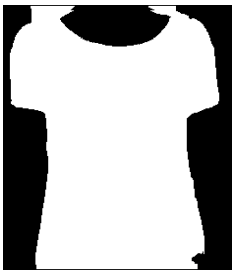

Desviación: 279,25

Imagen 34: **K-means** **Máscara** **Final**

Desviación: 539,954

Imagen 35: **K-means** **Máscara** **Final**

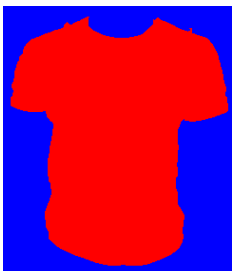
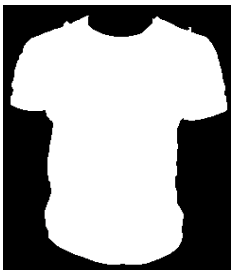

Desviación: 358,248

Imagen 36: **K-means** **Máscara** **Final**



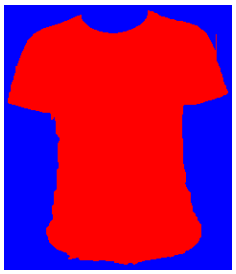
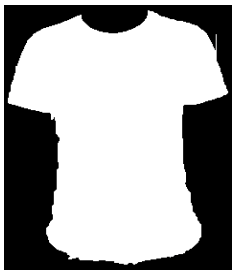

Desviación: 490,293

Imagen 37: **K-means** **Máscara** **Final**

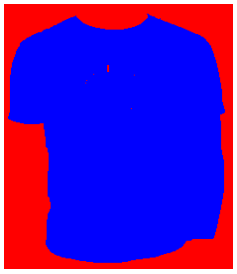
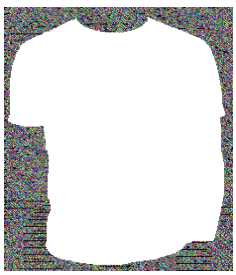

Desviación: 236,827

Imagen 38: **K-means** **Máscara** **Final**


Desviación: 496,532

Imagen 39: **K-means** **Máscara** **Final**


Desviación: 678,56

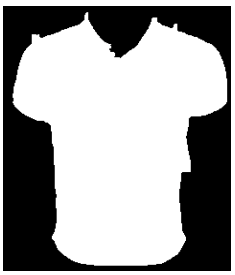
Imagen 40: **K-means** **Máscara** **Final**




Desviación: 594,573

Imagen 41: **K-means** **Máscara** **Final**





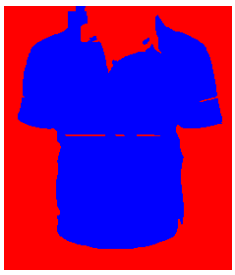

Desviación: 570,682

Imagen 42: **K-means** **Máscara** **Final**



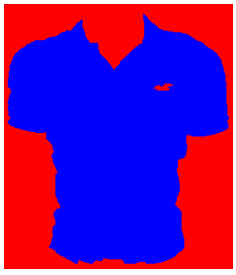

Desviación: 550,776

Imagen 43: **K-means** **Máscara** **Final**

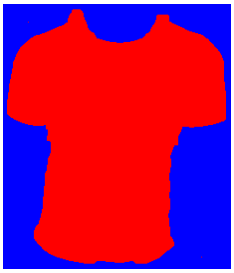
Desviación: 498,853

Imagen 44: **K-means** **Máscara** **Final**

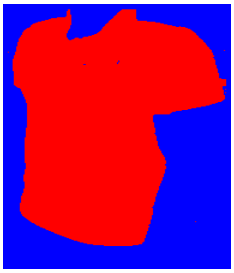
Desviación: 274,497

Imagen 45: **K-means** **Máscara** **Final**

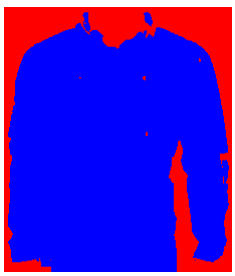
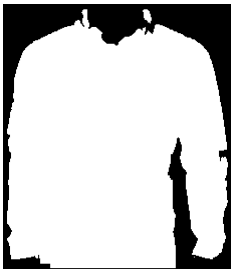
Desviación: 447,56

Imagen 46: **K-means** **Máscara** **Final**

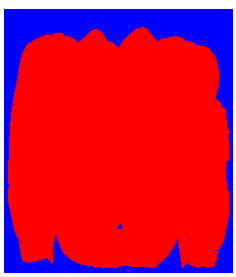
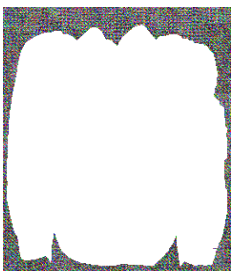
Desviación: 381,268

Imagen 47: **K-means** **Máscara** **Final**

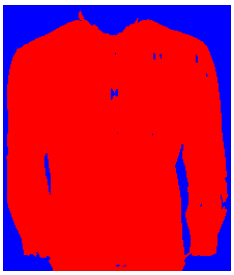

Desviación: 523,301

Imagen 48: **K-means** **Máscara** **Final**

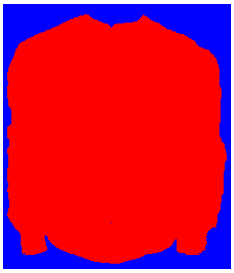
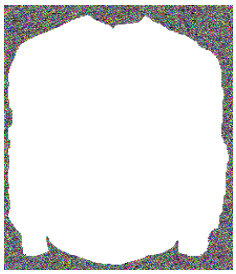

Desviación: 421,821

Imagen 49: **K-means** **Máscara** **Final**

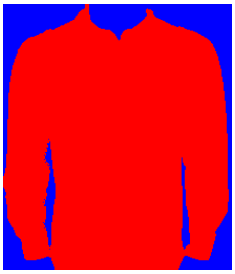
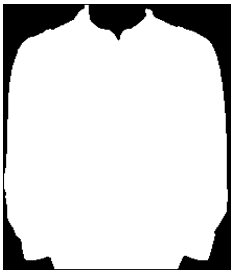
Desviación: 438,496

Imagen 50: **K-means** **Máscara** **Final**

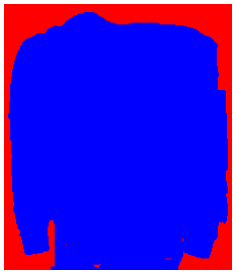
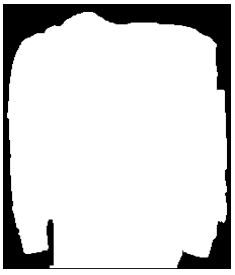

Desviación: 422,472

Imagen 51: **K-means** **Máscara** **Final**

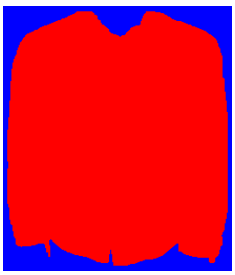
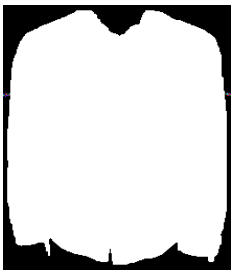

Desviación: 303,934

Imagen 52: **K-means** **Máscara** **Final**

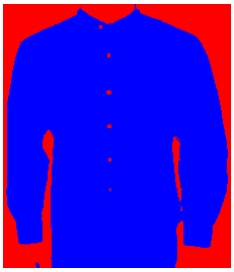
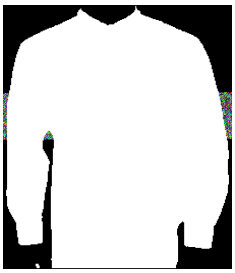

Desviación: 461,466

Imagen 53: **K-means** **Máscara** **Final**

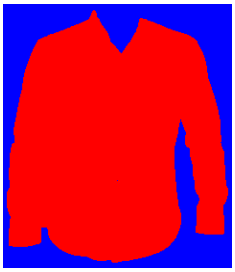
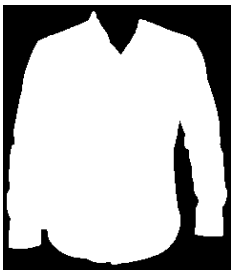

Desviación: 482,872

Imagen 54: **K-means** **Máscara** **Final**

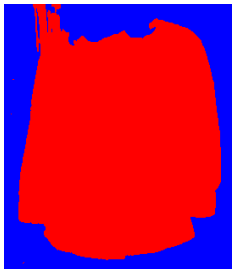

Desviación: 206,614

Imagen 55: **K-means** **Máscara** **Final**

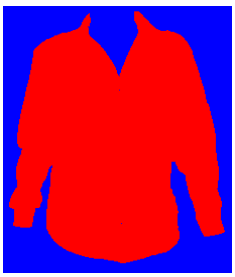

Desviación: 270,62

Imagen 56: **K-means** **Máscara** **Final**

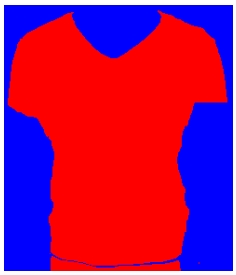
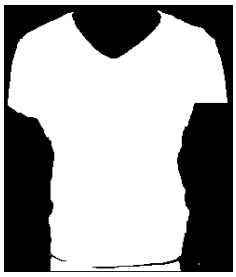
Desviación: 269,154

Imagen 57: **K-means** **Máscara** **Final**

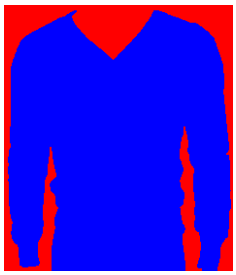
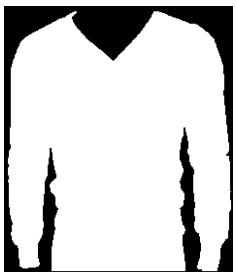
Desviación: 407,159

Imagen 58: **K-means** **Máscara** **Final**

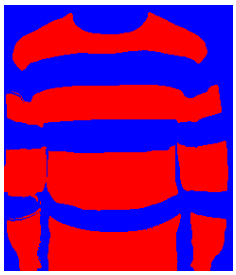

Desviación: 532,128

Imagen 59: **K-means** **Máscara** **Final**

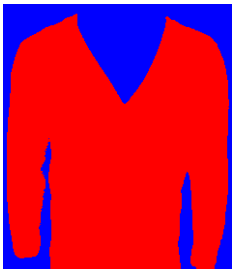
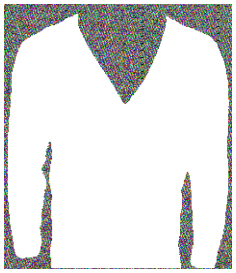

Desviación: 338,492

Imagen 60: **K-means** **Máscara** **Final**

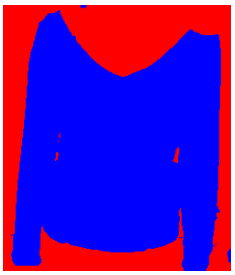
Desviación: 562,141

Imagen 61: **K-means** **Máscara** **Final**


Desviación: 226,482

Imagen 62: **K-means** **Máscara** **Final**

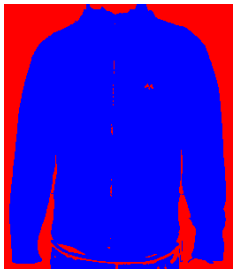
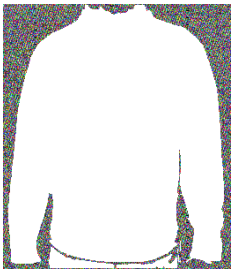
Desviación: 337,777

Imagen 63: **K-means** **Máscara** **Final**



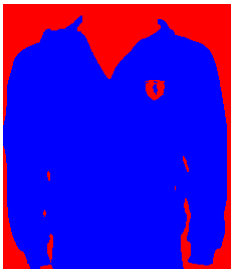

Desviación: 529,674

Imagen 64: **K-means** **Máscara** **Final**

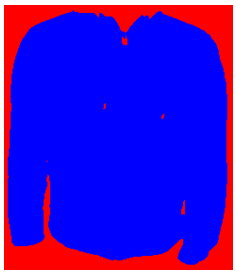
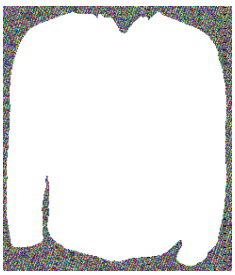

Desviación: 272,565

Imagen 65: **K-means** **Máscara** **Final**

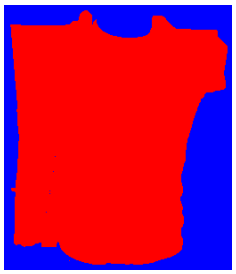
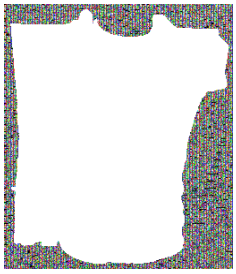
Desviación: 214,341

Imagen 66: **K-means** **Máscara** **Final**

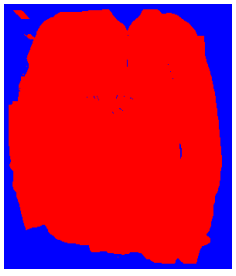
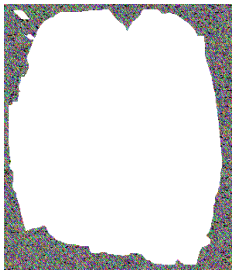

Desviación: 555,288

Imagen 67: **K-means** **Máscara** **Final**

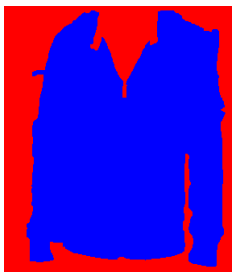
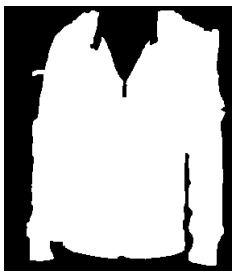

Desviación: 666,816

Imagen 68: **K-means** **Máscara** **Final**

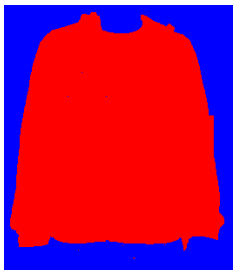
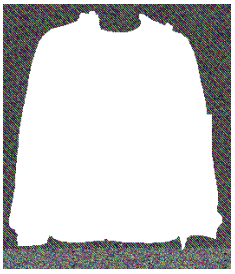

Desviación: 523,226

Imagen 69: **K-means** **Máscara** **Final**

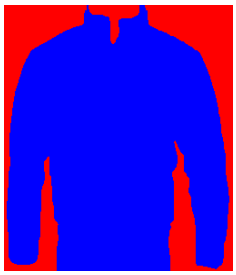
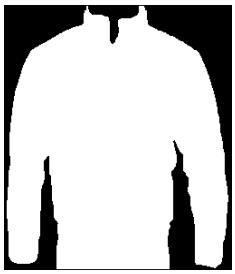

Desviación: 329,832

Imagen 70: **K-means** **Máscara** **Final**

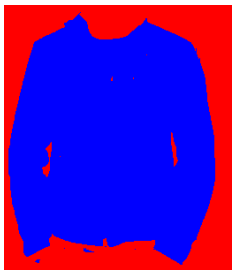
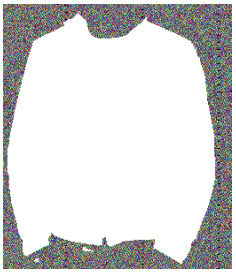

Desviación: 414,61

Imagen 71: **K-means** **Máscara** **Final**

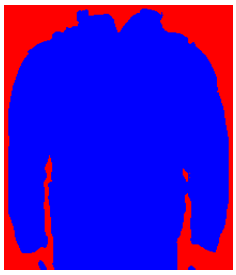
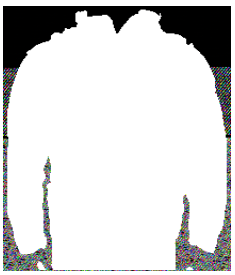

Desviación: 228,05

Imagen 72: **K-means** **Máscara** **Final**

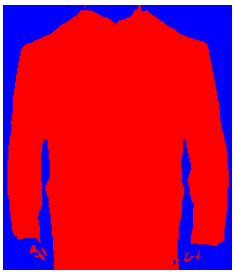
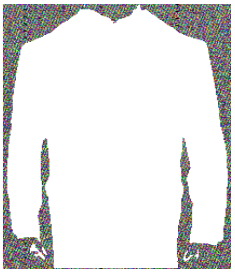

Desviación: 428,462

Imagen 73: **K-means** **Máscara** **Final**

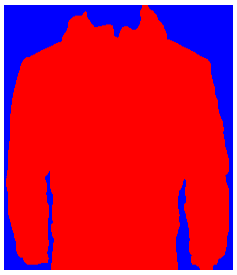

Desviación: 266,4

Imagen 74: **K-means** **Máscara** **Final**

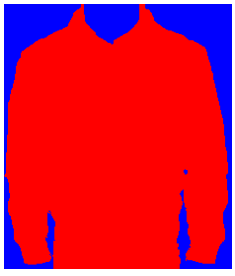
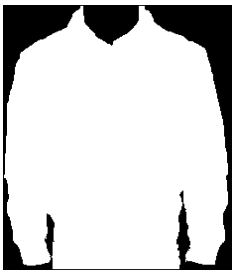

Desviación: 261,896

Imagen 75: **K-means** **Máscara** **Final**

Desviación: 259,529

Imagen 76: **K-means** **Máscara** **Final**


Desviación: 291,027

Apéndice C

Imagen 1: **Pódium**



Error medio: 0,025

Imagen 2: **Pódium**



Error medio: 0,032

Imagen 3: **Pódium**



Error medio: 0,032

Imagen 4: **Pódium**


Error medio: 0,035

Imagen 5: **Pódium**

Error medio: 0,029

Imagen 6: **Pódium**

Error medio: 0,018

Imagen 7: **Pódium**

Error medio: 0,019

Imagen 8: **Pódium**

Error medio: 0,057

Imagen 9: **Pódium**

Error medio: 0,012

Imagen 10: **Pódium**

Error medio: 0,037

Imagen 11: **Pódium**

Error medio: 0,031

Imagen 12: **Pódium**

Error medio: 0,008

Imagen 13: **Pódium**




Error medio: 0,024

Imagen 14: **Pódium**




Error medio: 0,02

Imagen 15: **Pódium**




Error medio: 0,028

Imagen 16: **Pódium**




Error medio: 0,017

Imagen 17: **Pódium**

Error medio: 0,016

Imagen 18: **Pódium**

Error medio: 0,017

Imagen 19: **Pódium**

Error medio: 0,018

Imagen 20: **Pódium**

Error medio: 0,027

Imagen 21: **Pódium**




Error medio: 0,042

Imagen 22: **Pódium**




Error medio: 0,045

Imagen 23: **Pódium**




Error medio: 0,017

Imagen 24: **Pódium**




Error medio: 0,051

Imagen 25: **Pódium**




Error medio: 0,033

Imagen 26: **Pódium**




Error medio: 0,013

Imagen 27: **Pódium**




Error medio: 0,024

Imagen 28: **Pódium**


Error medio: 0,024

Imagen 29: **Pódium**

Error medio: 0,019

Imagen 30: **Pódium**

Error medio: 0,013

Imagen 31: **Pódium**

Error medio: 0,028

Imagen 32: **Pódium**

Error medio: 0,013

Imagen 33: **Pódium**

Error medio: 0,019

Imagen 34: **Pódium**

Error medio: 0,016

Imagen 35: **Pódium**

Error medio: 0,031

Imagen 36: **Pódium**

Error medio: 0,026

Imagen 37: **Pódium**




Error medio: 0,026

Imagen 38: **Pódium**




Error medio: 0,014

Imagen 39: **Pódium**




Error medio: 0,019

Imagen 40: **Pódium**




Error medio: 0,032

Imagen 41: **Pódium**

Error medio: 0,013

Imagen 42: **Pódium**

Error medio: 0,017

Imagen 43: **Pódium**

Error medio: 0,031

Imagen 44: **Pódium**

Error medio: 0,033

Imagen 45: **Pódium**

Error medio: 0,027

Imagen 46: **Pódium**

Error medio: 0,027

Imagen 47: **Pódium**

Error medio: 0,03

Imagen 48: **Pódium**

Error medio: 0,027

Imagen 49: **Pódium**




Error medio: 0,025

Imagen 50: **Pódium**




Error medio: 0,024

Imagen 51: **Pódium**




Error medio: 0,039

Imagen 52: **Pódium**




Error medio: 0,029

Imagen 53: **Pódium**

Error medio: 0,028

Imagen 54: **Pódium**

Error medio: 0,034

Imagen 55: **Pódium**

Error medio: 0,064

Imagen 56: **Pódium**

Error medio: 0,024

Imagen 57: **Pódium**




Error medio: 0,045

Imagen 58: **Pódium**




Error medio: 0,039

Imagen 59: **Pódium**




Error medio: 0,013

Imagen 60: **Pódium**




Error medio: 0,045

Imagen 61: **Pódium**

Error medio: 0,039

Imagen 62: **Pódium**

Error medio: 0,042

Imagen 63: **Pódium**

Error medio: 0,029

Imagen 64: **Pódium**

Error medio: 0,032

Imagen 65: **Pódium**




Error medio: 0,044

Imagen 66: **Pódium**




Error medio: 0,034

Imagen 67: **Pódium**



Error medio: 0,035

Imagen 68: **Pódium**



Error medio: 0,044

Imagen 69: **Pódium**




Error medio: 0,047

Imagen 70: **Pódium**




Error medio: 0,033

Imagen 71: **Pódium**




Error medio: 0,039

Imagen 72: **Pódium**




Error medio: 0,06

Imagen 73: **Pódium**




Error medio: 0,051

Imagen 74: **Pódium**




Error medio: 0,045

Imagen 75: **Pódium**




Error medio: 0,061

Imagen 76: **Pódium**




Error medio: 0,048

Apéndice D

Imagen 1: **Pódium**


Error medio: 0,034

Imagen 2: **Pódium**


Error medio: 0,058

Imagen 3: **Pódium**



Error medio: 0,054

Imagen 4: **Pódium**


Error medio: 0,056

Imagen 5: **Pódium**




Error medio: 0,06

Imagen 6: **Pódium**




Error medio: 0,03

Imagen 7: **Pódium**


Error medio: 0,033

Imagen 8: **Pódium**




Error medio: 0,1

Imagen 9: **Pódium**

Error medio: 0,021

Imagen 10: **Pódium**



Error medio: 0,069

Imagen 11: **Pódium**



Error medio: 0,045

Imagen 12: **Pódium**



Error medio: 0,012

Imagen 13: **Pódium**




Error medio: 0,036

Imagen 14: **Pódium**

Error medio: 0,037

Imagen 15: **Pódium**



Error medio: 0,038

Imagen 16: **Pódium**




Error medio: 0,026

Imagen 17: **Pódium**

Error medio: 0,027

Imagen 18: **Pódium**

Error medio: 0,039

Imagen 19: **Pódium**

Error medio: 0,029

Imagen 20: **Pódium**

Error medio: 0,045

Imagen 21: **Pódium**




Error medio: 0,058

Imagen 22: **Pódium**




Error medio: 0,055

Imagen 23: **Pódium**




Error medio: 0,037

Imagen 24: **Pódium**




Error medio: 0,082

Imagen 25: **Pódium**



Error medio: 0,044

Imagen 26: **Pódium**


Error medio: 0,021

Imagen 27: **Pódium**




Error medio: 0,031

Imagen 28: **Pódium**


Error medio: 0,043

Imagen 29: **Pódium**

Error medio: 0,035

Imagen 30: **Pódium**

Error medio: 0,026

Imagen 31: **Pódium**

Error medio: 0,048

Imagen 32: **Pódium**

Error medio: 0,023

Imagen 33: **Pódium**

Error medio: 0,032

Imagen 34: **Pódium**

Error medio: 0,023

Imagen 35: **Pódium**

Error medio: 0,055

Imagen 36: **Pódium**

Error medio: 0,046

Imagen 37: **Pódium**

Error medio: 0,034

Imagen 38: **Pódium**

Error medio: 0,019

Imagen 39: **Pódium**

Error medio: 0,035

Imagen 40: **Pódium**

Error medio: 0,051

Imagen 41: **Pódium**

Error medio: 0,025

Imagen 42: **Pódium**

Error medio: 0,031

Imagen 43: **Pódium**

Error medio: 0,055

Imagen 44: **Pódium**

Error medio: 0,072

Imagen 45: **Pódium**

Error medio: 0,064

Imagen 46: **Pódium**

Error medio: 0,058

Imagen 47: **Pódium**

Error medio: 0,058

Imagen 48: **Pódium**

Error medio: 0,044

Imagen 49: **Pódium**




Error medio: 0,035

Imagen 50: **Pódium**




Error medio: 0,03

Imagen 51: **Pódium**




Error medio: 0,081

Imagen 52: **Pódium**




Error medio: 0,028

Imagen 53: **Pódium**




Error medio: 0,03

Imagen 54: **Pódium**




Error medio: 0,062

Imagen 55: **Pódium**




Error medio: 0,097

Imagen 56: **Pódium**




Error medio: 0,026

Imagen 57: **Pódium**




Error medio: 0,06

Imagen 58: **Pódium**




Error medio: 0,067

Imagen 59: **Pódium**




Error medio: 0,023

Imagen 60: **Pódium**




Error medio: 0,079

Imagen 61: **Pódium**

Error medio: 0,069

Imagen 62: **Pódium**

Error medio: 0,072

Imagen 63: **Pódium**

Error medio: 0,046

Imagen 64: **Pódium**

Error medio: 0,052

Imagen 65: **Pódium**




Error medio: 0,082

Imagen 66: **Pódium**




Error medio: 0,073

Imagen 67: **Pódium**


Error medio: 0,062

Imagen 68: **Pódium**



Error medio: 0,096

Imagen 69: **Pódium**



Error medio: 0,088

Imagen 70: **Pódium**




Error medio: 0,074

Imagen 71: **Pódium**




Error medio: 0,057

Imagen 72: **Pódium**




Error medio: 0,089

Imagen 73: **Pódium**

Error medio: 0,064

Imagen 74: **Pódium**

Error medio: 0,08

Imagen 75: **Pódium**

Error medio: 0,097

Imagen 76: **Pódium**

Error medio: 0,079

Apéndice E

Imagen 1: **Pódium**




Error medio: 0,022

Imagen 2: **Pódium**


Error medio: 0,132

Imagen 3: **Pódium**




Error medio: 0,052

Imagen 4: **Pódium**


Error medio: 0,12

Imagen 5: **Pódium**


Error medio: 0,075

Imagen 6: **Pódium**




Error medio: 0,037

Imagen 7: **Pódium**



Error medio: 0,069

Imagen 8: **Pódium**



Error medio: 0,182

Imagen 9: **Pódium**

Error medio: 0,026

Imagen 10: **Pódium**

Error medio: 0,181

Imagen 11: **Pódium**

Error medio: 0,101

Imagen 12: **Pódium**

Error medio: 0,006

Imagen 13: **Pódium**



Error medio: 0,039

Imagen 14: **Pódium**



Error medio: 0,042

Imagen 15: **Pódium**



Error medio: 0,041

Imagen 16: **Pódium**




Error medio: 0,001

Imagen 17: **Pódium**

Error medio: 0,031

Imagen 18: **Pódium**

Error medio: 0,048

Imagen 19: **Pódium**

Error medio: 0,04

Imagen 20: **Pódium**

Error medio: 0,11

Imagen 21: **Pódium**


Error medio: 0,096

Imagen 22: **Pódium**


Error medio: 0,105

Imagen 23: **Pódium**



Error medio: 0,064

Imagen 24: **Pódium**




Error medio: 0,129

Imagen 25: **Pódium**




Error medio: 0,054

Imagen 26: **Pódium**

Error medio: 0,021

Imagen 27: **Pódium**




Error medio: 0,026

Imagen 28: **Pódium**

Error medio: 0,036

Imagen 29: **Pódium**

Error medio: 0,059

Imagen 30: **Pódium**

Error medio: 0,024

Imagen 31: **Pódium**

Error medio: 0,087

Imagen 32: **Pódium**

Error medio: 0,019

Imagen 33: **Pódium**




Error medio: 0,026

Imagen 34: **Pódium**




Error medio: 0,003

Imagen 35: **Pódium**




Error medio: 0,12

Imagen 36: **Pódium**



Error medio: 0,043

Imagen 37: **Pódium**




Error medio: 0,03

Imagen 38: **Pódium**




Error medio: 0,008

Imagen 39: **Pódium**




Error medio: 0,058

Imagen 40: **Pódium**




Error medio: 0,064

Imagen 41: **Pódium**




Error medio: 0,051

Imagen 42: **Pódium**




Error medio: 0,057

Imagen 43: **Pódium**




Error medio: 0,07

Imagen 44: **Pódium**




Error medio: 0,186

Imagen 45: **Pódium**




Error medio: 0,199

Imagen 46: **Pódium**




Error medio: 0,14

Imagen 47: **Pódium**




Error medio: 0,08

Imagen 48: **Pódium**




Error medio: 0,004

Imagen 49: **Pódium**




Error medio: 0,046

Imagen 50: **Pódium**




Error medio: 0,013

Imagen 51: **Pódium**




Error medio: 0,115

Imagen 52: **Pódium**




Error medio: 0,016

Imagen 53: **Pódium**

Error medio: 0,001

Imagen 54: **Pódium**

Error medio: 0,104

Imagen 55: **Pódium**

Error medio: 0,237

Imagen 56: **Pódium**

Error medio: 0,009

Imagen 57: **Pódium**




Error medio: 0,095

Imagen 58: **Pódium**




Error medio: 0,063

Imagen 59: **Pódium**




Error medio: 0,02

Imagen 60: **Pódium**




Error medio: 0,184

Imagen 61: **Pódium**

Error medio: 0,081

Imagen 62: **Pódium**

Error medio: 0,198

Imagen 63: **Pódium**

Error medio: 0,028

Imagen 64: **Pódium**

Error medio: 0,065

Imagen 65: **Pódium**

Error medio: 0,146

Imagen 66: **Pódium**

Error medio: 0,181

Imagen 67: **Pódium**

Error medio: 0,068

Imagen 68: **Pódium**

Error medio: 0,206

Imagen 69: **Pódium**




Error medio: 0,213

Imagen 70: **Pódium**




Error medio: 0,154

Imagen 71: **Pódium**




Error medio: 0,099

Imagen 72: **Pódium**




Error medio: 0,161

Imagen 73: **Pódium**

Error medio: 0,118

Imagen 74: **Pódium**

Error medio: 0,082

Imagen 75: **Pódium**

Error medio: 0,216

Imagen 76: **Pódium**

Error medio: 0,135

7. Bibliografía y referencias

1. **Zalando.** http://www.zalando.es/. [En línea]

2. **Teruel, Francisco Jordán.** Estudio de la plataforma Android para dispositivos móviles y desarrollo de aplicación para la administración de redes de sensores inalámbricos. 2010.

3. **AndroidPolice.** http://www.androidpolice.com/. [En línea]

4. **Inc, Google.** http://www.google.com/mobile/goggles/#text. [En línea]

5. **AG, kooaba.** http://www.kooaba.com/. [En línea]

6. **Inc., Idée.** http://www.tineye.com/. [En línea]

7. **OpenCV.** http://opencv.willowgarage.com/wiki/. [En línea]

8. *Some methods for clasification and analysis of multivariate observations.* **MacQUEEN, J.** Los Angeles : s.n., 1967.

9. *Exploring Colour Photography: A Complete Guide.* **Hirsch, Robert.**

10. *Encyclopedia of Distances.* **Elena Deza, Michel Marie Deza.**

11. *Theory of Communication.* **Gabor, D.**

12. *A Multilevel Banded Graph Cuts Method for Fast Image Segmentation.* **Herve Lombaert, Yiyong Sun, Leo Grady, Chenyang Xu.** 2005.

13. **Lindeberg, Tony.** Scale-space. [aut. libro] John Wiley and Sons. *Encyclopedia of Computer Science and Engineering (Benjamin Wah, ed).* 2008, Vol. IV, pages 2495-2504.

14. *A combined corner and edge detector.* **Harris, C.G., Mohr, R., Bauckhage, C.** 1998. Alvey Vision Conference.

15. *Evaluación de Detectores de Puntos de Interés para Slam Visual.* **Ballesta, M., Gil, A., Reinoso, Ó., Payá, L., & Mozos, Ó. M.** s.l. : Universidad Miguel Hernández, 2007.

16. *Simultaneous localisation and map-building using active vision.* **Davidson, A.J., Murray, D.W.** s.l. : IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002.

17. *Good Features to Track.* **J. Shi, C. Tomasi.** IEEE Conference on Computer : s.n., 1994.

18. *ORB: an efficient alternative to SIFT or SURF.* **Ethan Rublee, Vicent Rabaud, Kurt Konolige, Gary Bradski.** International Conference on Computer Vision, 2011.

19. *Machine learning for highspeed corner detection.* **Drummond, E. Rosten and T.** European Conference on Computer Vision, volume 1, 2006.

20. *BRIEF: Binary Robust Independent Elementary.* **Calonder, M., Lepetit, V., Fua, P.** ECCV, 778{792 (2010).

21. *Understanding Cryptography: A Textbook for Students and Practitioners.* **Christof Paar, Jan Pelzl, Bart Preneel.**

22. *Error detecting and error correcting codes.* **Hamming, Richard W.**

23. *Color indexing.* **Ballard, M. Swain and D.** International journal of computer : s.n., 1991.

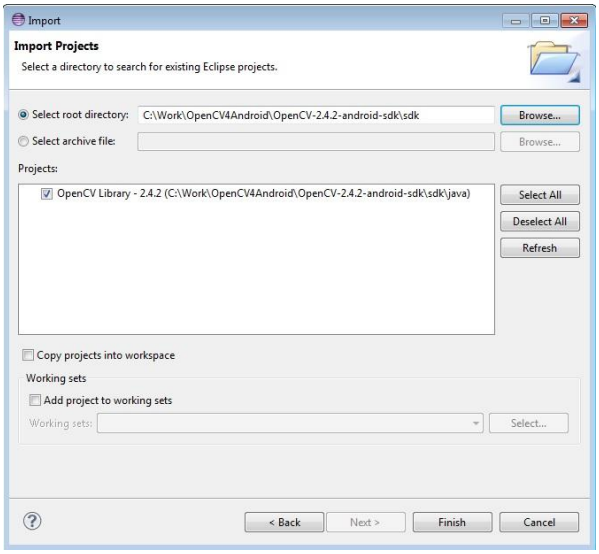
24.**Jeong,Sangoh.**
http://scien.stanford.edu/pages/labsite/2002/psych221/projects/02/sojeong/.[Enlínea]

Anexo A

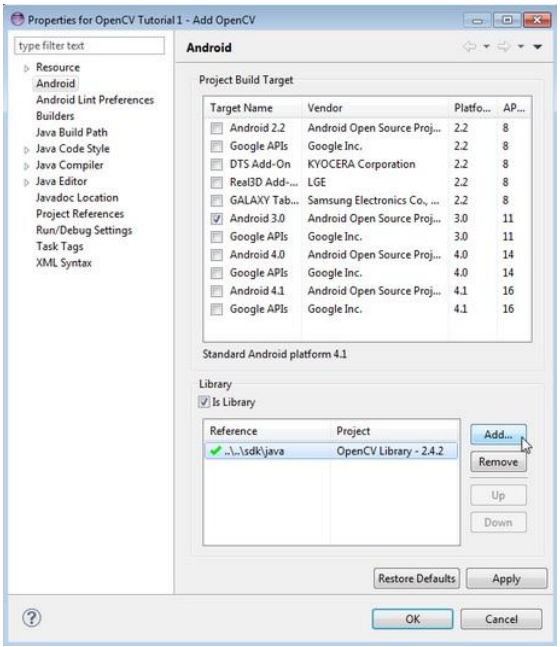
Añadir biblioteca OpenCV en proyecto Android con Inicialización asíncrona

Usar la inicialización asíncrona es lo recomendado mientras se desarrolla la aplicación. De esta manera la biblioteca accederá externamente a las funciones a través del OpenCV Manager instalado en la memoria del dispositivo. Siguiendo los pasos detallados a continuación se consigue añadir al proyecto la biblioteca OpenCV:

1 Añadir el proyecto OpenCV Library al workspace. Usando el menú *File -> Import -> Existing Projects into Workspace*. Pulsar el botón Browse y localizar OpenCV Library en el SDK de OpenCV.



2 En el proyecto de la aplicación añadir una referencia a la biblioteca en *Project -> Properties -> Android -> Library -> Add* select OpenCVLibrary.



3 En el siguiente fragmento de código se muestra la inicialización asíncrona de la biblioteca en onResume(), que es la llamada en Android justo antes de que el usuario pueda empezar a interactuar con la aplicación.

```
public class Sample1Java extends Activity implements CvCameraViewListener {

    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS:
                {
                    Log.i(TAG, "OpenCV loaded successfully");
                    mOpenCvCameraView.enableView();
                } break;
                default:
                {
                    super.onManagerConnected(status);
                } break;
            }
        }
    };

    @Override
    public void onResume()
    {
        super.onResume();
        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_5, this, mLoaderCallback);
    }

    ...
}
```

El OpenCV Manager puede ser instalado automáticamente desde Google Play. Al ejecutar la aplicación en el dispositivo móvil, éste detectará que no dispone del Manager y dará la opción de acudir a Google Play para instalarla. Una vez instalada, no necesitará ser reinstalada en cada instalación de la aplicación durante su desarrollo.

Como se ha comentado anteriormente, ésta es la mejor opción mientras que se desarrolla la aplicación puesto que la aplicación que se corre en el dispositivo móvil tiene un tamaño menor que si tuviera que incluir las funciones del Manager que ya se encuentran en el dispositivo de manera independiente. La versión definitiva de la aplicación incluirá todas las funciones para comodidad del usuario, evitando tener que descargarse la aplicación auxiliar.

Anexo B

Añadir biblioteca OpenCV en proyecto Android con Inicialización estática

Para hacer la inicialización estática de la biblioteca, los dos primeros pasos son idénticos a los de la asíncrona.

1 Añadir al workspace el proyecto de la biblioteca OpenCV.

2 Referenciarla en las propiedades del proyecto principal.

3 Copiar en la carpeta libs que cuelga del directorio de la aplicación en Eclipse, los archivos que encontraremos en la carpeta */sdk/native/libs* del SDK de OpenCV. Esta carpeta contiene todos los paquetes que forman OpenCV con sus respectivas funciones y que la aplicación debe llevar dentro al no disponer del Manager como aplicación auxiliar.

Solo se utilizará la inicialización estática en la última versión de la aplicación, para que el usuario se ahorre el trabajo de instalar el Manager y quede un acabado más profesional con un solo archivo *apk*.

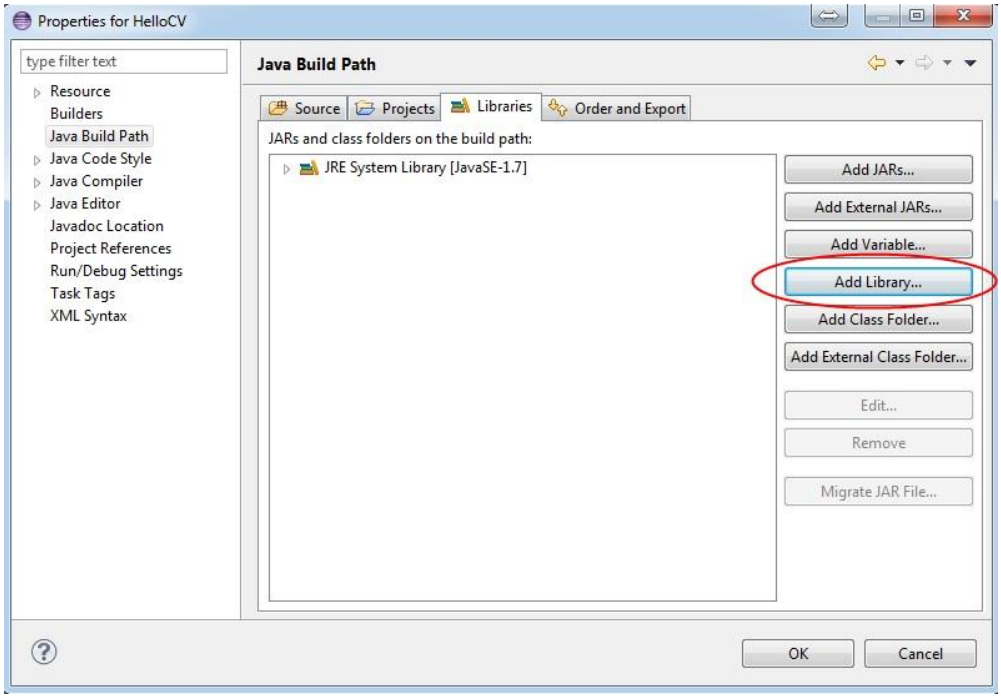
Anexo C

Añadir biblioteca OpenCV en proyecto auxiliar de escritorio

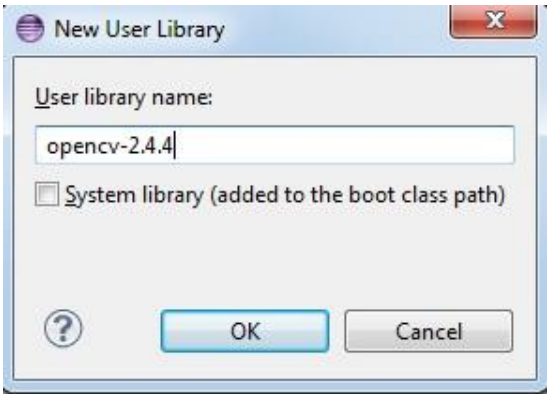
Para añadir a la aplicación de escritorio la biblioteca OpenCV utilizando Eclipse, se pueden seguir los siguientes pasos:

1 Crear un nuevo proyecto Java con *File –> New –> Java Project.*

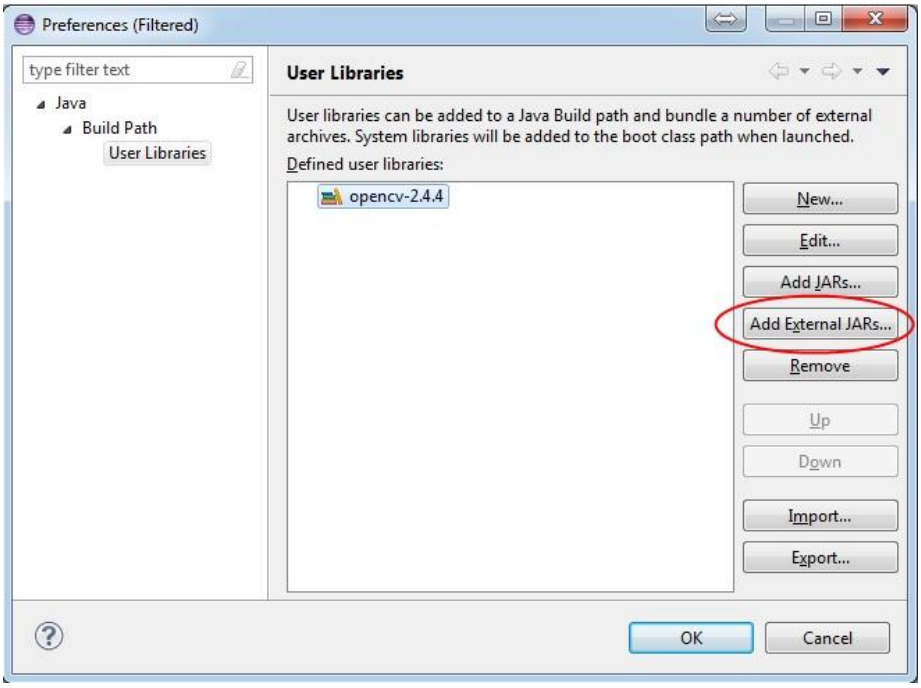
2 Acceder a la pestaña Libraries en Java Build Path dentro de las propiedades del proyecto, pulsar Add Library…



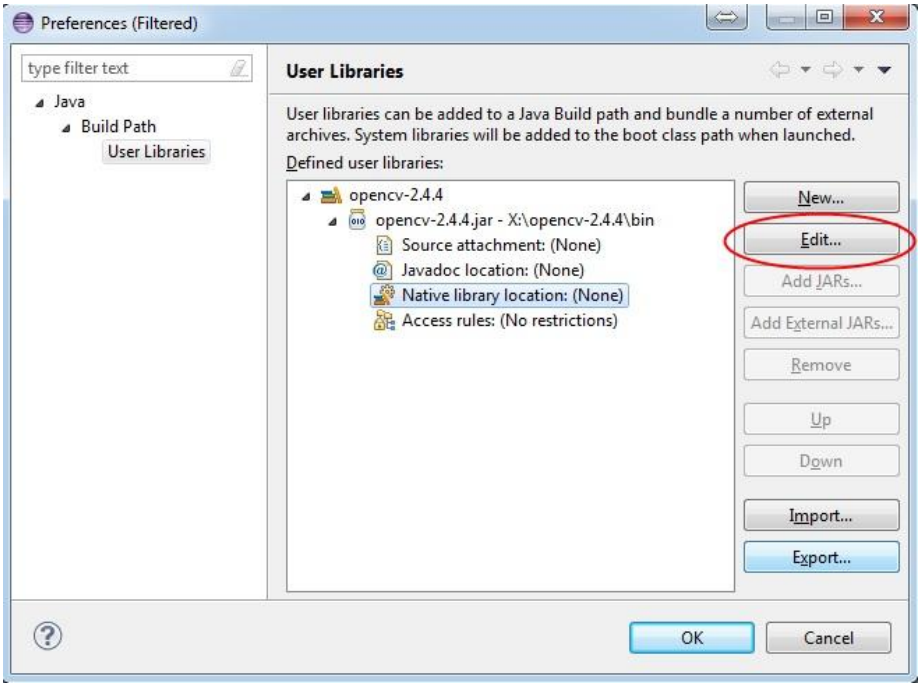
3 Crear una nueva entrada para la biblioteca eligiendo el nombre.



4 Añadir el *.jar* de OpenCV, situado en el SDK.



5 Indicar la localización del .dll, ya sea 64 bits o 32 bits, dependiendo del ordenador. Ambos también incluidos en el SDK.



6 Por último, hay que realizar desde el programa principal una llamada a la biblioteca para inicializarla.

```
import org.opencv.core.CvType;
import org.opencv.core.Mat;

public class Main {
    public static void main(String[] args) {
        System.loadLibrary("opencv_java244");
        Mat m  = Mat.eye(3, 3, CvType.CV_8UC1);
        System.out.println("m = " + m.dump());
    }
}
```