UNIVERSIDAD DE LAS FUERZAS ARMADAS

DPTO. DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERIA EN SISTEMAS E INFORMÁTICA

"ANÁLISIS DE EFICIENCIA EN ALGORITMOS DE RECONOCIMIENTO DE IMÁGENES DIGITALES APLICABLES A DISPOSITIVOS MÓVILES BAJO LA PLATAFORMA ANDROID"

Previa a la obtención del Título de:

INGENIERO EN SISTEMAS E INFORMATICA

POR: **MIGUEL FABRICIO ÑAUÑAY ILBAY**

LUIS GONZALO TIPANTUÑA CÓRDOVA

SANGOLQUÍ, Septiembre 2013

CERTIFICACIÓN

Certifico que el presente trabajo fue realizado en su totalidad por el Sr. Miguel Fabricio Ñauñay Ilbay y el Sr. Luis Gonzalo Tipantuña Córdova, como requerimiento parcial a la obtención del título de INGENIERO EN SISTEMAS E INFORMÁTICA.

Sangolquí , Septiembre de 2013

_____ _____

ING. GEOVANNY RAURA
DIRECTOR

ING.TATIANA GUALOTUÑA
COODIRECTOR

DECLARACIÓN

Nosotros, Fabricio Miguel Ñauñay Ilbay y Luis Gonzalo Tipantuña Córdova, declaramos que el presente trabajo es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación personal y que hemos consultado las referencias bibliográficas que se incluyen en el documento.

La Universidad de las Fuerzas Armadas puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la ley de propiedad intelectual por su reglamento y por la normativa institucional vigente.

Sangolquí , Septiembre de 2013

_____ _____

Sr. MIGUEL FABRICIO ÑAUÑAY ILBAY

Sr. LUIS GONZALO TIPANTUÑA CÓRDOVA

AUTORIZACIÓN

Autorizamos a la Universidad de las Fuerzas Armadas la publicación en la Biblioteca Virtual de la institución, del trabajo **"ANÁLISIS DE EFICIENCIA EN ALGORITMOS DE RECONOCIMIENTO DE IMÁGENES DIGITALES APLICABLES A DISPOSITIVOS MÓVILES BAJO LA PLATAFORMA ANDROID"**, cuyo contenido, ideas y criterios son de nuestra exclusiva responsabilidad y autoría.

Sangolquí , Septiembre de 2013

_____	_____
Sr. MIGUEL FABRICIO ÑAUÑAY ILBAY	**Sr. LUIS GONZALO TIPANTUÑA CÓRDOVA**

DEDICATORIA

Esta tesis se la dedico a mi familia que gracias a su apoyo pude concluir mi carrea. A mis padres y hermanos por su apoyo y confianza en todo lo necesario para cumplir mis objetivos como persona y estudiante.

FABRICIO MIGUEL ÑAUÑAY ILBAY

DEDICATORIA

La presente tesis se la dedico a Dios por ser la razón de mí existir y el pilar fundamental de mi vida. A mis padres Gonzalo y Guadalupe por su incondicional apoyo, tanto al inicio como al final de mi carrera; y a mis hermanos por hacerme compañía e infundirme ánimo.

LUIS GONZALO TIPANTUÑA CÓRDOVA

AGRADECIMIENTO

A la Universidad de las Fuerzas Armadas, porque en sus aulas, recibimos el conocimiento intelectual y humano. También agradezco a nuestro Director de Tesis el Ing. Geovanny Raura y a nuestra codirectora de tesis la Ing. Tatiana Gualotuña, gracias por sus consejos y amistad.

FABRICIO MIGUEL ÑAUÑAY ILBAY

AGRADECIMIENTO

Agradezco a Dios por la fuerza y el impulso que ha dado a mi vida. A mis padres Gonzalo y Guadalupe por mostrarme su amor y apoyo a pesar de las circunstancia y a mis hermanos por infundirme aliento.

Por ultimo agradezco a mis profesores porque han impartido de sus conocimientos en mí, y a cada una de las personas que de una u otra forma me ayudaron a conseguir esto.

LUIS GONZALO TIPANTUÑA CÓRDOVA

ÍNDICE DE CONTENIDOS

LISTADOS DE TABLAS, FIGURAS, Y ANEXOS

TABLAS:

FIGURAS:

ANEXOS:

Los anexos se encuentran disponibles después de la bibliografía

ABREVIATURAS

SIFT:	Scale-invariant feature transform, algoritmo detector de reconocimiento de imágenes.
SURF:	Speeded Up Robust Features, algoritmo detector de reconocimiento de imágenes.
ORB:	Oriented FAST and Rotated BRIEF, algoritmo detector de reconocimiento de imágenes.
API:	Interfaz de programación de aplicaciones, es el conjunto de funciones y procedimientos que ofrece una biblioteca o librería para ser utilizadas por otro software.
S.O:	Siglas utilizadas para abreviar Sistema Operativo
RIM:	Research In Motion Limited, compañía canadiense fabricante de BlackBerry
Wi-FI:	Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.
EPOC32:	Sistema utilizado en PDAs y computadoras de mano, en el cual se basó Symbian OS
PDA's:	Personal digital assistant, es una computadora de mano.
PSION:	Empresa fabricante de computadoras portátiles.
FOMA:	Compañía fabricante de tecnología de comunicaciones
HTC:	Fabrícate de teléfonos móviles
SDK:	Software Development kit, conjunto de herramientas de desarrollo de software que permite al programador crear aplicaciones

GLOSARIO

Kernel:	Elemento más importante del sistema operativo, es el encargado de que el software y el hardware del computador trabajen juntos
Middleware:	Software de computadora que conecta componentes de software o aplicaciones para que puedan intercambiar datos entre ellas.
Codec:	Software capaz de codificar un flujo de información o señal para luego recuperarlo o decodificarlo en un formato adecuado para su reproducción o manipulación.
Plugin:	Software que incrementa o aumenta las funcionalidades de un programa principal.
Tablet:	Computadora con forma de tabla, sin teclado y con una gran pantalla sensible al tacto.
iPhone:	Teléfono inteligente producido por Apple
iPad:	Tablet producida por Apple
Computadora de mano:	Microcomputadora de características sumamente reducidas
MAC OSX:	Sistema operativo para computadoras MAC
Darwin BSD:	Sistema que subyace en Mac OSX
Windows CE:	Windows Embedded Compact, sistema operativo desarrollado por Microsoft para sistemas embebidos

Cocoa Touch:	Es un API para la creación de programas para el iPad, iPhone y iPod Touch de la compañía Apple Inc.
SpringBoard:	Término utilizado para referirse al escritorio del iPhone.
Acelerómetro:	Elemento sensor que mide la aceleración, el ángulo de inclinación, la rotación, la vibración, el choque y la gravedad de un dispositivo móvil.
Giroscopio:	Elemento sensor que permite cambiar la orientación de un dispositivo móvil haciendo girar su pantalla.
Xbox Live:	Nombre del servicio online de Microsoft que da soporte a los juegos de la Xbox 360 y la antigua Xbox.
Streaming:	Es la distribución de multimedia a través de una red de computadoras de manera que el usuario consume el producto al mismo tiempo que se descarga.
Tienda apps:	Lugar en línea, donde se descargan aplicaciones móviles
Bug:	Error o fallo en un programa de computador
Widgets:	Los widgets son aplicaciones en miniatura que se colocan en las pantallas principales del Android
Framework:	Estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado.
Project Butter:	Aplicación que mejora la velocidad en la interfaz y las animaciones de Android.
Google Now:	Aplicación que actúa como un asistente inteligente.
Bytecode Java:	Es el tipo de instrucciones que la máquina de virtual de Java puede interpretar.

Ciclo de vida:	Etapas seguidas por una aplicación desde su creación hasta su implementación y actualización.
Netbeans:	Entorno de desarrollo integrado, hecho principalmente para el lenguaje de programación Java.
Perl:	Lenguaje de programación
Python:	Lenguaje de programación
Cliente enriquecido (RCP):	Un cliente enriquecido consiste en proporcionar una interfaz gráfica, escrita con una sintaxis basada en XML, que proporciona funcionalidades (arrastrar y soltar, pestañas, ventanas múltiples, menús desplegables).
Gap:	Hace referencia a una brecha, una apertura o un espacio vacío comprendido entre dos puntos de referencia.
Diferencia Gaussiana:	Función matemática que obtiene la diferencia de dos imágenes en la escala-espacio
Filtro de Harris Corner:	Función que obtiene datos relevantes de una imagen a partir de sus esquinas
Decisiones binarias:	Es una estructura de datos utilizada para representar una función booleana.
Threshold:	Parámetro de umbral o radio para ajustar algoritmos de reconocimiento de imágenes.

RESUMEN

En la actualidad existen aplicaciones móviles cada vez más complejas, pero limitadas al uso de los recursos en los dispositivos móviles. Por lo tanto, el eficiente manejo de hardware se convierte en un elemento clave en la evaluación de una aplicación, especialmente si se trata de una aplicación que requiere un alto procesamiento computacional como es el caso del reconocimiento de imágenes.

En vista de esta problemática, el presente proyecto analiza los descriptores SIFT, SURF y ORB en busca del mejor algoritmo para la detección de objetos, mediante la utilización de métricas de eficiencia y eficacia.

Es importante mencionar que los experimentos serán implementados y ejecutados en un teléfono móvil con sistema operativo Android, a diferencia de investigaciones previas realizadas en ambientes computacionales.

Al finalizar la investigación se desarrollara un prototipo funcional con la capacidad de reconocer imágenes, específicamente el logo de una empresa o producto para posteriormente mostrar información relacionada a la imagen identificada.

CAPÍTULO 1

VISIÓN GENERAL DEL PROYECTO

1.1 INTRODUCCIÓN

El progreso de la tecnología y la búsqueda de información en tiempo real han hecho de los dispositivos móviles un instrumento transcendental en la vida de los seres humanos, es por ello que en la actualidad dicho progreso tiende a tecnificar parcialmente a los usuarios desarrollando un factor de digitalización social donde son capaces de relacionarse con personas u organizaciones de todo el mundo a través de un dispositivo móvil.

Según la revista (Redes&Telecom, 2012) de España se indica que para "el 2013 el número de teléfonos móviles en el mundo alcanzará los 2.000 millones de unidades" mientras que según estudios realizados por (Cisco, 2012) se marcó que en "2016 habrá más teléfonos móviles que personas en el mundo, estimándose que para el 2016 la población mundial alcanzará la marca de los 7,300 millones de habitantes según la ONU." De hecho años atrás, Steve Jobs predijo que la era post PC había comenzado (TechRepublic, 2010) y que la penetración de Tablet y teléfonos inteligentes seria cada vez más grande.

Tablet: Computadora con forma de tabla, sin teclado y con una gran pantalla sensible al tacto.

Un hecho que se destaca en el mundo de los dispositivos móviles es la amplia aceptación y popularidad del sistema operativo Android, se estima que se realizan aproximadamente 850.000 activaciones diarias y se venden 300 millones (Theverge, 2012) de equipos con este sistema.

Android es un sistema operativo pensado para dispositivos móviles, está basado en Linux y fue creado para proporcionar a los teléfonos móviles capacidades que se asemejan a las de una PC como navegar en la web, leer emails, descargar aplicaciones, entre otros.

En la actualidad se encuentra en el mercado aplicaciones como Goggles (Google) y Flow (Amazon) que haciendo uso de técnicas de reconocimiento de imágenes, permiten realizar búsquedas a partir de una imagen tomada con la cámara de un teléfono móvil.

Es así que en el reconocimiento de imágenes existen algoritmos como: "Scale-Invariant Feature Transform" (SIFT), "Affine – Surf" (ASIF) y "Speeded Up Robust Feature" (SURF[I]) entre otros, los cuales permiten extraer características que sinteticen la mayor parte de información útil para la detección de una imagen.

Reconocimiento de Imágenes: Proceso que busca identificar mediante la comparación de dos imágenes si ambas representan al mismo objeto o escena.
SIFT: Scale-invariant feature transform, algoritmo detector de reconocimiento de imágenes.
SURF: Speeded Up Robust Features, algoritmo detector de reconocimiento de imágenes.

Estos algoritmos tratan de describir la imagen a través de un conjunto de puntos concretos de la misma y mediante la información espacial de cada punto (Chen, 2007).

El presente estudio, se enfoca en la evaluación del rendimiento de algoritmos de reconocimiento de imágenes en dispositivos móviles, específicamente sobre la plataforma Android. En base a este análisis, se realizara un prototipo que permita el reconocimiento de logotipos y se asocie información obtenida desde un servicio web.

1.2 PLANTEAMIENTO DEL PROBLEMA

1.2.1 Investigación

Al reconocimiento de imágenes se lo define como el proceso que permite identificar objetos físicos por medio de una cámara digital, este proceso se lleva a cabo a través de la aplicación de técnicas de ingeniería, computación y matemáticas que conjugadas entre sí permiten establecer las propiedades de un objeto.

En la actualidad existen algoritmos de reconocimiento de imágenes que son de libre licenciamiento y que han sido aplicados en proyectos computacionales con gran capacidad de procesamiento, la aplicación de estos algoritmos en dispositivos móviles se ve afectada por la limitante de recursos como capacidad de procesamiento, almacenamiento, memoria y uso de batería.

Es aquí cuando se vuelve indispensable el analizar estos algoritmos para determinar de forma clara y precisa cuál de ellos sería el más idóneo para ser aplicado en un dispositivo móvil.

1.2.2 Desarrollo del Prototipo:

El logotipo o "logo" es la representación gráfica de una empresa o producto, este es de vital importancia ya que el cliente asociara una imagen al producto o servicio que la empresa provea.

Las empresas tratan de hacer llegar a sus clientes promociones y nuevos productos a través de los diferentes medios de difusión masiva como lo es la radio, televisión, periódico entre otros y en los últimos años se ha consolidado la difusión a través de las redes sociales como Facebook y Twitter.

Si bien las campañas publicitarias llegan a ser vistas o escuchadas por los clientes no aseguran la entrega total de la información que la empresa quiere transmitir, esto debido a las limitaciones que un espacio publicitario pueda tener como: el tiempo de duración, el espacio en donde se muestra, etc.

En la actualidad los dispositivos móviles son más asequibles para las personas por lo que muchas empresas están enfocando sus esfuerzos publicitarios hacia este sector a través de los aplicativos móviles.

1.3 JUSTIFICACIÓN

Uno de los principales problemas que enfrentan los teléfonos móviles (independiente del sistema operativo que tenga: Android, IOS o Windows Phone) es la cantidad limitada de recursos de: CPU, memoria RAM y batería (Carroll, 2010). Por lo tanto, el uso eficiente de recursos se convierte en un elemento clave en la evaluación de una aplicación móvil especialmente si se trata de una aplicación que requiere un alto procesamiento computacional como es el caso del reconocimiento de imágenes.

El reconocimiento de imágenes o reconocimiento de objetos, es una de las aplicaciones más importantes de la visión por computador y aunque existe una gran cantidad de literatura sobre el tema, aun no se halla un algoritmo cien por ciento capaz de enfrentarse a la gama de variaciones que puede afectar a una imagen (MalikYasir, 2012) como los cambios de iluminación, oclusión o puntos de vista diferentes.

Es así que durante los últimos años, ha existido un creciente interés en el enfoque basado en la descripción de un objeto, utilizando “descriptores locales” (MalikYasir, 2012). Los descriptores locales permiten detectar estructuras o puntos significativos en la imagen y obtener una descripción discriminante de estas estructuras a partir de sus alrededores, con el objetivo de ser comparados con otros descriptores usando medidas de similitud.

Quizá uno de los descriptores locales que más influencia ha tenido en el campo de la visión por computador es Scale Invariant Feature Transform (SIFT) propuesto por David Lowe (Lowe, 1999). Aunque el detector y descriptor de puntos clave SIFT ha sido puesto a prueba en varias investigaciones y es ampliamente utilizado en aplicaciones como: composición panorámica, reconocimiento de objetos o la navegación visual, el alto procesamiento computacional que requiere este descriptor ha sido el punto de partida para la creación de nuevos descriptores tales como Speeded-Up Feature Transform (SURF) y Oriented FAST and Rotated BRIEF (ORB) los cuales se enfocan en mejorar la eficiencia y la precisión de correspondencias entre puntos clave con respecto a SIFT.

Frente a este escenario, como contribución la presente tesis basará sus esfuerzos en analizar el mejor algoritmo para la detección de objetos, mediante la utilización de criterios y métricas de eficiencia y eficacia. Es importante mencionar que los experimentos serán implementados y ejecutados en un teléfono móvil con sistema operativo Android, a diferencia de investigaciones previas como la evaluación del desempeño de los métodos de extracción para procesamiento de imágenes 3D (Dwarakanath D, 2012), en donde se analiza el desempeño los algoritmos SIFT, SURF y ORB pero orientado a un sistema computacional que procesa imágenes o la investigación denominada evaluación del método de reconocimiento SIFT en Robots móviles (Ramisa A, 2011), que analiza el desempeño de algoritmo SIFT aplicado a Robots.

ORB: Oriented FAST and Rotated BRIEF, algoritmo detector de reconocimiento de imágenes.

Al finalizar la investigación se desarrollará un prototipo funcional con la capacidad de reconocer imágenes, específicamente el logo de una empresa o producto y posteriormente mostrar toda la información relacionada a la misma, para ello se valdrá de la cámara de un teléfono móvil con sistema operativo Android versión 2.3 o superior

1.4 OBJETIVOS

1.4.1 Objetivo General

Realizar un análisis de eficiencia y eficacia en algoritmos de reconocimiento de imágenes, mediante la investigación experimental, para optimizar el consumo de recursos en un teléfono inteligente con sistema operativo Android.

1.4.2 Objetivos Específicos

- Analizar e implementar algoritmos de reconocimiento de imágenes digitales que sean aplicables a dispositivos móviles, sobre plataforma Android.
- Realizar un estudio comparativo de eficiencia y uso de recursos en dispositivos móviles, entre los algoritmos implementados.
- Demostrar la eficiencia del algoritmo mediante el desarrollo de un prototipo capaz de reconocer logotipos de empresas y asociar información mediante el consumo de un servicio web.

1.5 ALCANCE

Este proyecto abarcará dos fases: La primera fase enfocada al desarrollo de una investigación que permitirá analizar el funcionamiento y las prestaciones dadas por los algoritmos de reconocimiento de imágenes y procesamiento digital SIFT SURF y ORB; en base a este análisis se implementaran los algoritmos sobre dispositivos móviles que soporten el sistema operativo Android.

Se realizará un estudio comparativo sobre la eficiencia y eficacia en el uso de los algoritmos, considerando las siguientes variables: número de Puntos de interés obtenidos, tiempo de ejecución, porcentaje de descarga de batería, pesos de los descriptores y por último el desempeño.

La segunda fase abarcará el desarrollo de un prototipo funcional capaz de reconocer imágenes de logotipos, para ello se utilizara uno de los algoritmos investigados en este caso el que demuestre ser más eficiente.

1.6 HIPÓTESIS

La adecuada selección de un algoritmo de reconocimiento de imágenes en aplicaciones móviles, disminuye el uso de recursos disponibles en un teléfono inteligente.

1.7 METODOLOGÍA

Este proyecto utilizará dos tipos de metodología, la primera relacionada a la investigación y la segunda relacionada al desarrollo de la aplicación.

1.7.1 Investigación

En esta etapa se emplea la metodología de investigación exploratoria, debido a que se analizará el comportamiento de los algoritmos SIFT, SURF y ORB con respecto a las variables de eficiencia y eficacia.

1.7.2 Desarrollo

En la etapa de desarrollo se aplicará el modelo de prototipos, debido a que se implementará un aplicativo con el algoritmo que mejores prestaciones presente.

CAPÍTULO 2

MARCO TEÓRICO

2.1 DISPOSITIVOS MÓVILES

2.1.1 Definición

Los dispositivos móviles son aparatos electrónicos de tamaño pequeño, fáciles de transportar, con capacidad de procesamiento, memoria limitada y con una conexión permanente o intermitente a una red (Monteagudo, 2005), la figura 1 detalla las características de un dispositivo móvil.

Figura 1 Características de los Dispositivos móviles

2.1.2 Categorías de los dispositivos móviles:

Teniendo claro el concepto de dispositivo móvil, es importante mencionar que esta definición agrupa a un gran conjunto de equipos electrónicos, entre ellos teléfonos móviles inteligentes, reproductores de música, tablets y hasta sistemas de posicionamiento global.

Es por ello que en el 2005, T38 y DuPont Global Mobility Innovation Team propusieron los siguientes estándares para la definición de los dispositivos móviles, los cuales están agrupados en la Tabla 1.

Tabla 1 Categorías de los Dispositivos móviles

DISPOSITIVOS MÓVILES		
	CARACTERÍSTICAS	EJEMPLOS
Dispositivo Móvil de Datos Limitados	Dispositivos que tienen una pantalla pequeña, principalmente basada en pantalla de tipo texto con servicios de datos generalmente limitados a SMS y acceso WAP	Teléfonos móviles
Dispositivo Móvil de Datos Básicos	Dispositivos que tienen pantallas de medianas a grandes (por encima de los 240 x 120 pixels), navegación de tipo stylus, y que ofrecen las mismas características que el "Dispositivo Móvil de Datos Básicos" (Basic Data Mobile Devices) más aplicaciones nativas y aplicaciones corporativas usuales, en versión móvil.	BlackBerry y los Teléfonos Inteligentes
Dispositivo Móvil de Datos Mejorados	Dispositivos que tienen pantallas de medianas a grandes (por encima de los 240 x 120 pixels), navegación de tipo stylus, y que ofrecen las mismas características que el "Dispositivo Móvil de Datos Básicos" (Basic Data Mobile Devices) más aplicaciones nativas y aplicaciones corporativas usuales, en versión móvil.	Dispositivos incluyen el sistema operativo Android, Windows Mobile, iPhone OS, entre otros.

Nota. Fuente: T38 y DuPont Global Mobility Innovation Team (2005)

2.2 TELÉFONOS INTELIGENTES

Un teléfono inteligente es un teléfono móvil que incorpora características de una computadora personal (ALEGSA, 1998 - 2013). En un concepto más amplio se lo define como un dispositivo electrónico que funciona como teléfono móvil, cuenta con sistema operativo y con características similares a las de un ordenador personal.

Una característica importante de casi todos los teléfonos inteligentes es que permiten la instalación de programas para incrementar el procesamiento de datos y la conectividad.

2.3 SISTEMAS OPERATIVOS MÓVILES

Un Sistema Operativo Móvil, es el software básico de un teléfono inteligente básicamente provee al teléfono una interfaz entre el usuario, las aplicaciones y los dispositivos de hardware, además coordina la conectividad inalámbrica que sería uno de los factores más importantes en estos dispositivos.

Un Sistema Operativo Móvil, está compuesto por varias capas, las cuales son el núcleo del sistema o Kernel , la interfaz de usuario y el Middleware (Torres, 2011).

Kernel: Elemento más importante del sistema operativo, es el encargado de que el software y el hardware del computador trabajen juntos

El Kernel proporciona el acceso a los distintos elementos de hardware del dispositivo, también ofrece servicios a los elementos superiores como los controladores o drivers para el hardware, permite la gestión de procesos, maneja el sistema de archivos y la gestión de la memoria.

El Middleware es el componente del sistema operativo que permite la ejecución de las aplicaciones diseñadas para las plataformas, su funcionamiento no depende del usuario ni de configuraciones externas, ya que viene programado para un correcto funcionamiento. Puede prestar servicios como codecs , plugins para la interpretación de páginas web, software para mensajería instantánea, entre otras aplicaciones de multimedia.

La interfaz de usuario es el componente de un sistema operativo que hace llamativo el dispositivo móvil, es la parte con la que el usuario interactúa para acceder a las aplicaciones, incluye menús, elementos gráficos, que permiten un manejo sencillo del móvil.

Middleware: Software de computadora que conecta componentes de software o aplicaciones para que puedan intercambiar datos entre ellas.
Codec: Software capaz de codificar un flujo de información o señal para luego recuperarlo o decodificarlo en un formato adecuado para su reproducción o manipulación.
Plugin: Software que incrementa o aumenta las funcionalidades de un programa principal.

2.3.1 Visión general de los sistemas operativos en los dispositivos móviles

En la actualidad existe una gran cantidad de sistemas operativos para teléfonos inteligentes, siendo el más popular Android según una investigación realizada por PCWorld (PCWorld, 2012). Es así que esta tesis aparta la sección 2.2 para detallar de forma profunda al sistema operativo Android.

A continuación se detallan de forma general los sistemas operativos de mayor acogida en el mercado:

2.3.1.1 Sistema Operativo Android

Android es un sistema operativo basado en Linux y orientado a dispositivos móviles, como teléfonos inteligentes y tablets. Fue desarrollado inicialmente por Android Inc, una firma comprada por Google en el 2005 (Martinez, 2011), para mayor detalle sobre Android refiérase a la sección 2.5.

2.3.1.2 Sistema Operativo iOS

iOS (anteriormente denominado iPhone OS) es un sistema operativo móvil de Apple desarrollado originalmente para el iPhone , siendo después usado en el iPod

---

iPhone: Teléfono inteligente producido por Apple

Touch e iPad . Es un derivado de Mac OS X , que a su vez está basado en Darwin BSD .

El iOS tiene 4 capas de abstracción: la capa del “núcleo del sistema operativo”, la capa de "Servicios Principales", la capa de "Medios de comunicación" y la capa de "Cocoa Touch " (Martinez, 2011).

Las características y especificaciones actuales de iOS son (Martinez, 2011):

- Interfaz de usuario intuitiva, basada en una pantalla multitáctil y un conjunto de componentes hardware internos (acelerómetros y giroscopios) que permiten interactuar con el S.O .
- Una pantalla principal (llamada “SpringBoard ”) donde están ubicados los iconos de las aplicaciones.
- Soporte para mensajería SMS y MMS
- Cliente de correo (Mail)
- Navegador web (Safari)
- Soporte para videoconferencia

---

iPad: Tablet producida por Apple.
MAC OSX: Sistema operativo para computadoras MAC
Darwin BSD: Sistema que subyace en Mac OSX
Cocoa Touch: Es un API para la creación de programas para el iPad, iPhone y iPod Touch de la compañía Apple Inc.
Acelerómetro: Elemento sensor que mide la aceleración, el ángulo de inclinación, la rotación, la vibración, el choque y la gravedad de un dispositivo móvil.
Giroscopio: Elemento sensor que permite cambiar la orientación de un dispositivo móvil haciendo girar su pantalla.
S.O: Siglas utilizadas para abreviar Sistema Operativo
SpringBoard: Término utilizado para referirse al escritorio del iPhone.

- Soporte para la mayoría de los formatos multimedia estándar. Aunque cabe destacar que iOS no soporta Adobe Flash y Java.
- Soporte para HTML5
- Soporte multitarea desde la versión 4.

2.3.1.3 Sistema Operativo BlackBerry OS

El BlackBerry OS es un sistema operativo móvil desarrollado por Research in Motion para sus dispositivos BlackBerry. El sistema permite multitarea y tiene soporte para diferentes métodos de entrada adoptados por RIM para su uso en computadoras de mano (Martinez, 2011).

Características y especificaciones actuales (Martinez, 2011):

- Gestor de correo electrónico y agenda compatible con Microsoft Exchange Server, Lotus Notes y Novell GroupWise.
- BlackBerry Enterprise Server, que proporciona el acceso al mail de grandes compañías.
- BlackBerry Internet Service, que proporciona acceso a internet y correo para usuarios particulares.
- Navegador con tecnología WebKit

RIM: Research In Motion Limited, compañía canadiense fabricante de BlackBerry
Computadora de mano: microcomputadora de características sumamente reducidas

- Soporte para Wi-Fi
- Múltiple lista de contactos
- Reconocimiento del rostro

2.3.1.4 Sistema Operativo Windows Phone

Windows Phone, anteriormente llamado Windows Mobile es un sistema operativo móvil compacto desarrollado por Microsoft, y diseñado para su uso en teléfonos inteligentes y otros dispositivos móviles.

Se basa en el núcleo del sistema operativo Windows CE y cuenta con un conjunto de aplicaciones básicas utilizando las API de Microsoft Windows. Está diseñado para ser similar a las versiones de escritorio de Windows estéticamente. Originalmente apareció bajo el nombre de Pocket PC, como una ramificación de desarrollo de Windows CE para equipos móviles con capacidades limitadas. (Martinez, 2011)

Características y especificaciones actuales:

- Interfaz gráfica intuitiva, con ventanas vivas.
- Pantalla táctil

Wi-FI: Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.
Windows CE: Windows Embedded Compact, sistema operativo desarrollado por Microsoft para sistemas embebidos
API: Interfaz de programación de aplicaciones, es el conjunto de funciones y procedimientos que ofrece una biblioteca o librería para ser utilizadas por otro software.

- Integración con redes sociales
- Soporte para los formatos multimedia más comunes.
- Soporte para Xbox Live
- Conectividad: Bluetooth, Wi-Fi
- Mensajería: SMS, MMS
- Navegador web: Internet Explorer
- Soporte para streaming

2.3.1.5 Sistema Operativo Symbian OS

Symbian es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran Nokia, Sony Ericsson, Samsung, Siemens, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. Sus orígenes provienen de su antepasado EPOC32 , utilizado en PDA's y computadoras de mano de PSION . (Martinez, 2011)

El objetivo de Symbian fue crear un sistema operativo para terminales móviles que pudiera competir con el Windows Mobile de Microsoft, el Android de Google, el iOS de Apple Inc. y el Blackberry de RIM. La gran mayoría de móviles con sistema operativo Symbian son de la compañía Nokia, aunque también se puede encontrar

---

Xbox Live: nombre del servicio online de Microsoft que da soporte a los juegos de la Xbox 360 y la antigua Xbox.
Streaming: Es la distribución de multimedia a través de una red de computadoras de manera que el usuario consume el producto al mismo tiempo que se descarga.
EPOC32: Sistema utilizado en PDAs y computadoras de mano, en el cual se basó Symbian OS
PDA's: Personal digital assistant, es una computadora de mano.
PSION: Empresa fabricante de computadoras portátiles.

este sistema operativo en algunos modelos de las marcas Sony-Erikson, Motorola, Siemens, Panasonic y FOMA (Martinez, 2011)

Es importante destacar que este sistema operativo ha quedado relegado debido al gran impacto causado por Android y iOS.

2.4 APLICACIONES EN DISPOSITIVOS MÓVILES

Las aplicaciones móviles son programas software capaces de ejecutarse en dispositivos móviles inteligentes y que pueden ser descargados desde la web de sus creadores o en las tiendas apps disponibles, su función es básicamente proveer al usuario consumidor un servicio específico.

2.4.1 Tipos de aplicaciones móviles

A las aplicaciones móviles se las divide en tres tipos (Geospatial, 2012 - 2013):

- Nativas
- Webs
- Híbridas

FOMA: Compañía fabricante de tecnología de comunicaciones
Tienda apps: Lugar en línea, donde se descargan aplicaciones móviles

A continuación se procede a detallar cada una de ellas:

2.4.1.1 Aplicaciones nativas

Este tipo de aplicaciones están hechas para ejecutarse en un dispositivo y sistema operativo específico (Geospatial, 2012 - 2013). Cabe destacar que para el desarrollo de estas aplicaciones es importante conocer el lenguaje de programación base en el que están hechas, es así que para el sistema operativo iOS los lenguajes utilizados son: Objective C, C, o C++, mientras que para desarrollar aplicaciones en el sistema operativo Android se utiliza el lenguaje Java. La figura 2, muestra los lenguajes utilizados para desarrollar aplicaciones nativas dependiendo del sistema operativo móvil.

				
Lenguajes	Obj-C, C, C++	Java (C, C++)	Java	C#, Vb .Net, etc
Herramientas	Xcode	Android SDK	BB Java Eclipse Plug-in	Visual Studio, Windows Phone, Dev Tools
Archivos Ejecutables	.app	.apk	.cod	.xap
Tiandas App	Apple Itunes	Google Play	BlackBerry App World	Windows Phone Market

Figura 2: Lenguajes y herramientas de desarrollo por sistema operativo (Geospatial, 2012 - 2013)

Una de las características importantes de este tipo de aplicaciones es la rápida ejecución sobre el dispositivo que está corriendo, esto debido a que interactúan directamente con el sistema operativo que los aloja.

Otra característica a resaltar en estas aplicaciones es el empleo de todos los componentes de hardware disponible en el dispositivo como sensores, GPS, cámara, acelerómetro, etc. Cabe destacar que estas dos características son las diferencias fundamentales con respecto a las aplicaciones web, descritas en el punto 2.4.1.2.

A continuación por medio de la tabla 2, se exponen las ventajas y desventajas de las aplicaciones nativas.

Tabla 2 Ventajas y desventajas de las aplicaciones nativas

APLICACIONES NATIVAS	
Ventajas	La gran ventaja de las aplicaciones nativas es el total acceso que puede tener la aplicación sobre los componentes de hardware del dispositivo, además no requieren de conexión web para ser ejecutadas
Desventajas	La principal desventaja de este tipo de aplicaciones es la pérdida de mercado, esto como consecuencia de la dependencia del dispositivo, la aplicación y el sistema operativo. Además una aplicación nativa debe esperar la aprobación de la compañía dueña del sistema operativo para poder ser publicada y accesible al gran público.

Nota. Fuente: Geospatial. (2009) “Aplicaciones Nativas”

La figura 3 ilustra el comportamiento de una aplicación nativa.

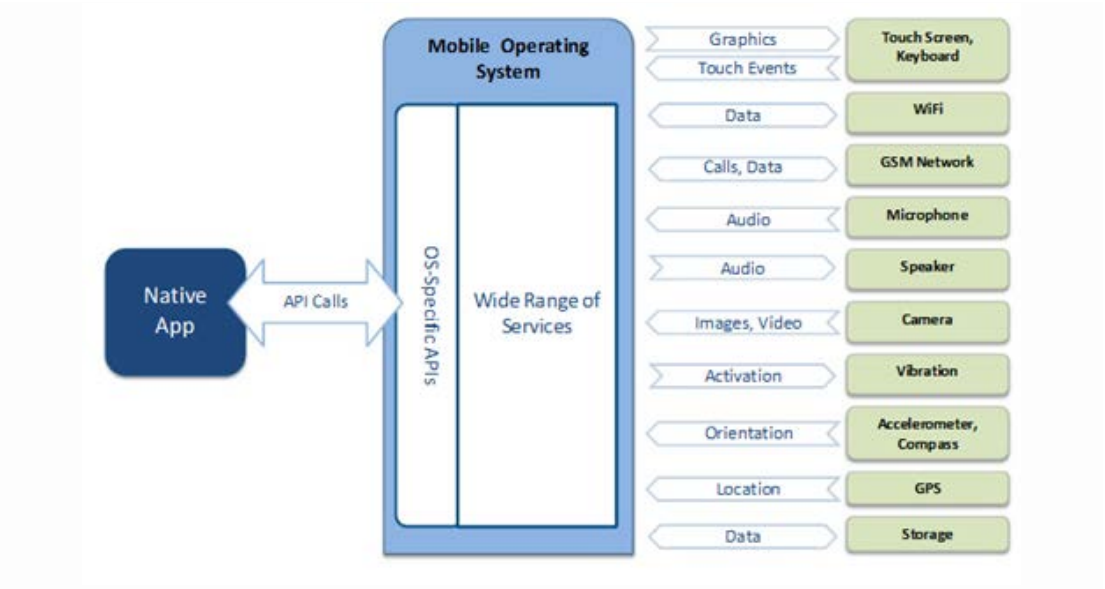

Figura 3: Comportamiento de una aplicación nativa con respecto al hardware (Geospatial, 2012 - 2013)

2.4.1.2 Aplicaciones Web móviles

Las aplicaciones web móviles, a diferencia de las aplicaciones nativas, se ejecutan dentro del navegador del teléfono. La figura 4, ilustra el comportamiento de una aplicación web móvil.

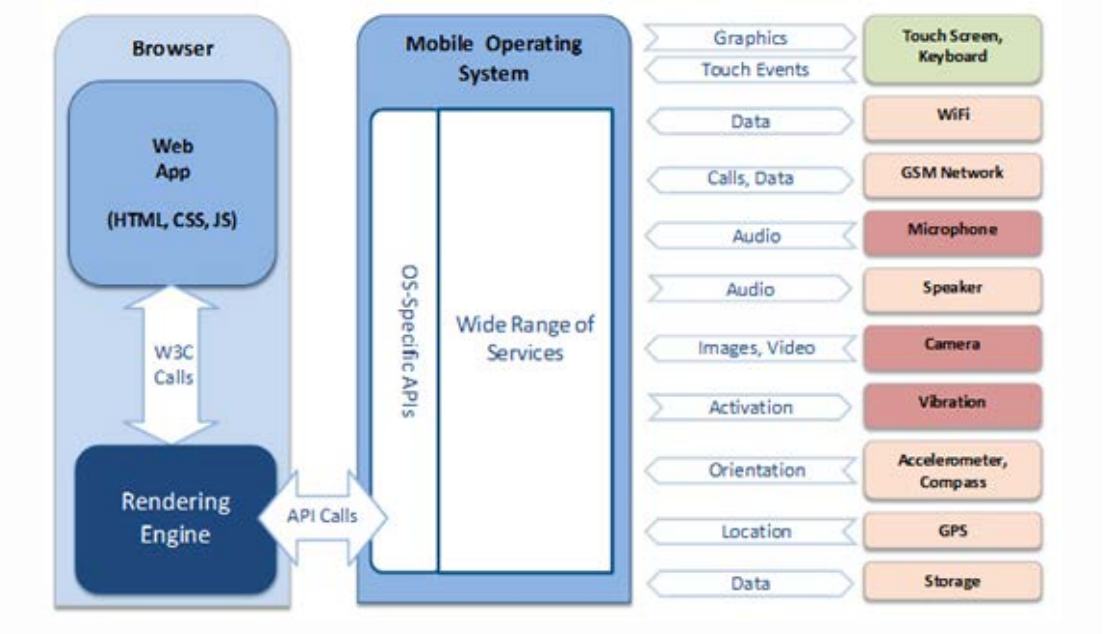

Figura 4: Comportamiento de aplicaciones Web móviles (Geospatial, 2012 - 2013)

Como se puede observar la tabla 3, se exponen las ventajas y desventajas de las aplicaciones web móviles.

Tabla 3 Ventajas y desventajas de las aplicaciones web móviles

APLICACIONES WEB MÓVILES	
Ventajas	Al contrario que las aplicaciones nativas, las aplicaciones web se pueden ejecutar en múltiples dispositivos evitando así las complejidades de tener que crear varias aplicaciones. El proceso de desarrollo es más sencillo ya que emplean tecnologías ya conocidas como HTML, CSS y Javascript. No necesitan de la aprobación de ningún fabricante para ser publicadas.
Desventajas	Como desventajas tenemos que el acceso a los elementos del teléfono son limitados. Además, estas aplicaciones no se pueden vender en las tiendas apps.

Nota. Fuente: Geospatial. (2009) “Aplicaciones Móviles”

2.4.1.3 Aplicaciones Híbridas

Las aplicaciones híbridas asocian lo mejor de los dos anteriores modelos. Este tipo de aplicaciones permite el uso de tecnologías multiplataforma como HTML, Javascript y CSS pero permiten acceder a parte de los dispositivos y sensores del teléfono (Geospatial, 2012 - 2013).

Un buen ejemplo de aplicaciones híbridas son Facebook, Skype, Goggles, etc. Las aplicaciones hibridas se pueden descargar de las tiendas app de los fabricantes del sistema operativo y cuenta con todas las características de una aplicación nativa pero también con las ventajas del web.

La figura 5 muestra el comportamiento de una aplicación hibrida.

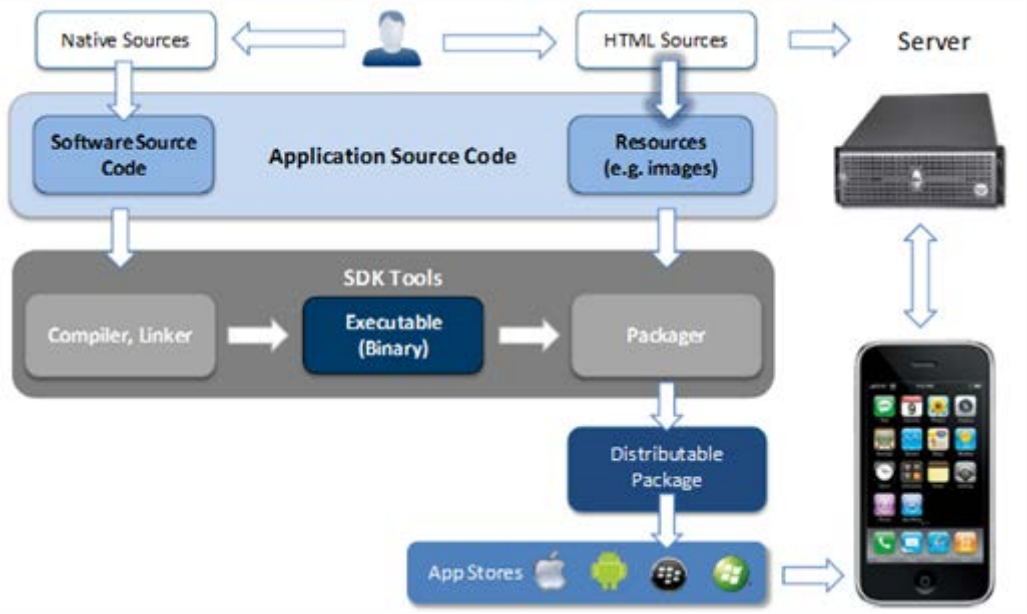

Figura 5: Comportamiento de una aplicación hibrida (Geospatial, 2012 - 2013)

El proceso de desarrollo para este tipo de aplicaciones es algo más complicado ya que se debe conocer sobre el lenguaje de programación soportado por cada sistema operativo móvil y sobre programación web.

2.5 SISTEMA OPERATIVO ANDROID

2.5.1 Definición

Android es un sistema operativo inicialmente pensado para teléfonos móviles, al igual que iOS, Symbian y Blackberry OS. Lo que lo hace diferente es que está basado en el núcleo de Linux, por lo cual se convierte en un sistema operativo libre, gratuito y multiplataforma (Gonzales, 2011).

El sistema operativo está compuesto por 12 millones de líneas de código, incluyendo 3 millones de líneas de XML, 2.8 millones de líneas de lenguaje C, 2.1 millones de líneas de Java y 1.75 millones de líneas de C++ (Wikipedia, 2012).

2.5.2 Historia

El sistema operativo Android nace en Android Inc de la mano de Andy Rubin como un proyecto para crear algo nuevo y diferente, con 22 meses de vida Android Inc es comprada por Google en agosto de 2005 (Salas, 2011) y fue justamente aquí donde el Android comenzó su etapa de crecimiento hasta convertirse en el sistema operativo número uno de los teléfonos inteligentes.

Android fue anunciado oficialmente el 5 de Noviembre de 2007 pero fue hasta octubre de 2008 cuando hizo su debut en el teléfono “dream” de HTC (Tofel, 2012).

HTC: Fabrícate de teléfonos móviles

Desde entonces Android ha sufrido muchos cambios, novedades y versiones, las versiones de Android reciben el nombre de postres en inglés y como dato interesante cabe destacar que cada versión de postre elegido empieza por una letra distinta siguiendo un orden alfabético:

- A: Apple Pie (v1.0), Tarta de manzana
- B: Banana Bread (v1.1), Pan de plátano
- C: Cupcake (v1.5), Magdalena glaseada
- D: Donut (v1.6), Rosquilla
- E: Éclair (v2.0/v2.1), pastel francés
- F: Froyo (v2.2), helado de yogur
- G: Gingerbread (v2.3), Pan de jengibre.
- H: Honeycomb (v3.0/v3.1/v3.2), Panal de miel.
- I: Ice Cream Sandwich (v4.0), Sándwich de helado.
- J: Jelly Bean (v4.1/v4.2), Judía de gominola.

2.5.3 Versiones

Android ha visto numerosas actualizaciones desde su liberación inicial, básicamente los reajustes al sistema operativo base arreglan bugs y agregan nuevas funciones.

Bug: Error o fallo en un programa de computador.

Generalmente cada actualización del sistema operativo Android es desarrollada bajo un nombre de un postre como se mencionó anteriormente.

A continuación se detallan las versiones que Android posee hasta la actualidad (Actualidadg, 2012).

2.5.3.1 Android 1.5 o Cupcake (magdalena)

Fue la primera actualización que recibió el sistema operativo una vez estuvo liberado y que elevaba la versión desde la 1.1 hasta 1.5.

Los cambios visuales respecto a su antecesora no fueron muy grandes ni elocuentes, pero se refinó y mejoró las transiciones de ventanas, el scroll en el navegador, y desde entonces fue compatible con las vistas en horizontal (Actualidadg, 2012).

Su interfaz era un poco simple sobre todo si la comparamos como es en la actualidad. Es así que esta versión no tiene la barra inferior de aplicaciones, aunque si posee la barra superior de notificaciones y widgets .

La figura 6 muestra una gráfica de la interfaz de usuario de la versión Cupcake, del sistema operativo Android.

Widgets: Los widgets son aplicaciones en miniatura que se colocan en las pantallas principales del Android

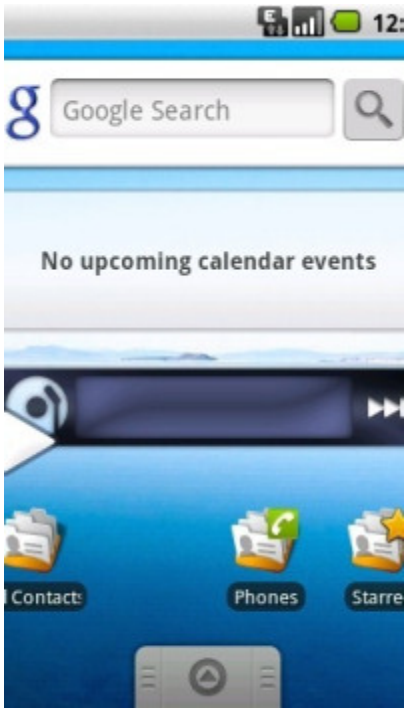

Figura 6: Interfaz gráfica del Android Cupcake (Actualidadg, 2012)

2.5.3.2 Android 1.6 o Donut (Rosquilla)

Fue la segunda actualización aunque las mejoras y nuevas implementaciones en el mismo no eran muy significativas. La interfaz permanece prácticamente inalterada aunque se rediseñan los iconos y los widgets, la aplicación para la gestión de la cámara también recibe una actualización de su interfaz pero sobre todo el cambio en el aspecto del Android market es lo que más llama la atención (Actualidadg, 2012).

Con android 1.6 se da soporte al sistema para trabajar con diferentes resoluciones de pantalla y se actualiza su kernel.

2.5.3.3 Android 2.0 o Eclair (rollo de crema)

Esta actualización sin duda fue un antes y un después para Android tanto en su interfaz como en funcionalidades que se añadían, su interfaz cambió radicalmente con un rediseño casi completo de todas sus partes, iconos, aplicaciones del framework de Android, opciones, etc (Actualidadg, 2012). Fue un salto cualitativo muy importante para el sistema y llegó acompañado por una gran campaña de marketing de mano de Motorola que fue el empujón final que necesitaba el sistema para ser la competencia del iOS. La figura 7 muestra el cambio de interfaz.

Figura 7 Interfaz de Android Eclair (Actualidadg, 2012)

Framework: Estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado.

En esta versión llego por primera vez los fondos animados de pantalla que fue toda una revolución para los usuarios de este sistema y que hoy en día siguen siendo un gran atractivo para la personalización de los terminales.

2.5.3.4 Android 2.2 o Froyo (yogourt helado)

Al igual que sucedió con la versión Donut, Froyo es una actualización menor del sistema si bien en esta ocasión la interfaz de Android cambia algo su aspecto introduciéndose la barra inferior en los escritorios, algunos widgets e iconos actualizan su aspecto con un leve cambio, la aplicación de la cámara y la galería fotográfica son las que mayor mejora sufrieron (Actualidadg, 2012). La figura 8, muestra gráficamente los leves cambios realizados a la versión Froyo del sistema operativo Android.

Figura 8 Interfaz de Android Froyo (Actualidadg, 2012)

2.5.3.5 Android 2.3 o Gingerbread (galletas de jengibre)

Esta versión también marcó un antes y un después en la interfaz de Android, quizás menos llamativo que el que hubo entre Donut y Eclair pero en Gingerbread se rediseñan los menús, colores e iconos del sistema (Actualidadg, 2012). Llega un nuevo teclado con opciones para seleccionar texto de manera mucho más clara y precisa, además como un plus Google rediseña la interfaz del android market. La figura 9 muestra gráficamente la interfaz del Android Gingerbread.

Figura 9 Interfaz de Android Gingerbread (Actualidadg, 2012)

2.5.3.6 Android 3.0 o HoneyComb (panal de abeja)

Esta versión de Android fue destinada para tablets únicamente y lo que planteo fue un rediseño de toda la interfaz de usuario, que luego sería heredada por las

siguientes versiones de Android. La figura 10 ilustra la interfaz de Android HoneyComb (Actualidadg, 2012).

Figura 10 Interfaz de Android HoneyComb (Actualidadg, 2012)

En Honeycomb se crea una nueva barra de menús superior, que se integra el buscador como parte del escritorio y nace una nueva barra inferior donde se tienen los botones del control del sistema.

2.5.3.7 Android 4.0 o Ice Cream Sandwich (sandwich de helado)

Es una de las últimas versiones de versión de Android, de nuevo la interfaz sufre un cambio radical y se toman partes de Gingerbread y de HoneyComb mezclándose. Entre las características que se resaltan de esta versión están las nuevas funciones en la barra de notificaciones, además permite a las aplicaciones ser compatibles en todos los terminales Android sin tener en cuenta su tamaño de pantalla o resolución

(Actualidadg, 2012). Una de las cosas más llamativas para los usuarios es el rediseño completo de la aplicación de gestión de cámara, de la galería multimedia y de la aplicación de contactos. La figura 11 muestra la interfaz de Android Ice Cream Sandwich.

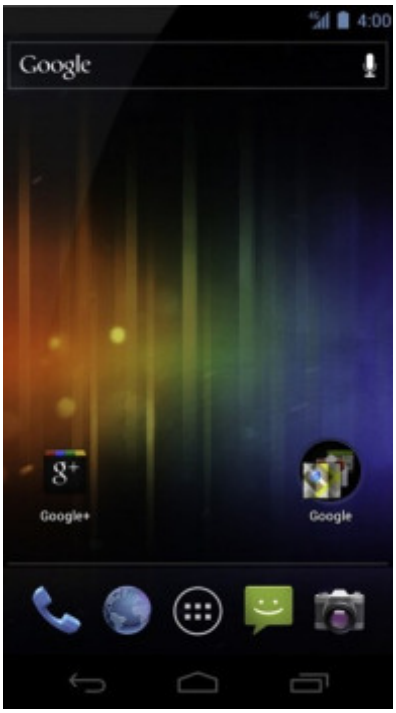

Figura 11: Interfaz de Android Ice Cream Sandwich (Actualidadg, 2012)

2.5.3.8 Android 4.1 o Jelly Bean (caramelitos)

Es la última versión de Android puesta en el mercado, Jelly Bean, entre sus novedades se encuentra que su interfaz cambia poco o casi nada respecto a la anterior versión Android 4.0.

Se añade un nuevo sistema de notificaciones expandibles situados en la misma zona donde estaban su antecesora, se incorpora un nuevo sistema de gestión de datos

entre la CPU y la GPU llamado “Project Butter ” haciendo las transiciones entre aplicaciones o entre ventanas mucho más rápidas, suaves y optimizadas (Actualidadg, 2012).

Otra novedad es el desplazamiento automático de los iconos del escritorio para dejar sitio a nuevas aplicaciones que corren en ese momento, se mejoró el gestor de la aplicación de la cámara de fotos, se añadió un teclado más rápido y con mejor predicción, se incorporó Google Now , y se hizo al sistema más intuitivo. La figura 12 grafica la interfaz de usuario de Android Ice Cream Sandwich.

Figura 12: Interfaz de Android Ice Cream Sandwich (Actualidadg, 2012)

Project Butter: Aplicación que mejora la velocidad en la interfaz y las animaciones de Android.
Google Now: Aplicación que actúa como un asistente inteligente.

2.5.3.9 Android 4.2

Android 4.2 es una actualización menor de Jelly Bean, presenta pocos cambios en la interfaz y agrega nuevas funcionalidades que mejoran el sistema en general. Entre las funciones añadidas a Android 4.2, entre las que más resaltan están la inclusión de multicuentas que posibilita el crear diferentes perfiles de usuarios en un mismo dispositivo, cada uno con sus propias aplicaciones y configuraciones (Actualidadg, 2012). También agrega un nuevo teclado con mejor capacidad predictiva y que incluye una función que posibilita el escribir palabras deslizando el dedo por las diferentes letras que componen estas palabras.

2.6 ARQUITECTURA DEL SISTEMA OPERATIVO ANDROID

El sistema operativo Android en su ámbito arquitectónico está estructurado por varias capas, la figura 13 ilustra de forma gráfica la arquitectura del sistema.

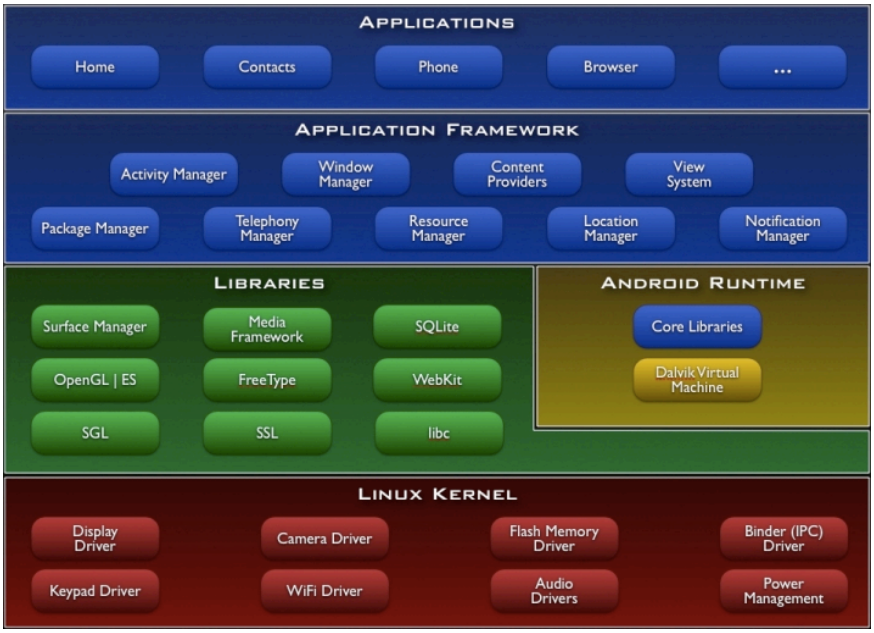

Figura 13: Arquitectura de Android (Condesa, 2011)

A continuación se detallará cada uno de los componentes de la arquitectura del sistema operativo Android.

2.6.1 Kernel de Linux

Es el núcleo del sistema operativo Android está basado en el kernel de Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, solo que adaptado a las características del hardware en el que se ejecutará Android, es decir, para dispositivos móviles (Condesa, 2011).

El núcleo actúa como una capa de abstracción entre el hardware y el resto de las capas de la arquitectura.

El kernel también se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación, etc.

2.6.2 Librerías

La siguiente capa que se sitúa justo sobre el kernel y la componen las bibliotecas nativas de Android, también llamadas librerías. Estas están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono.

Las librerías normalmente están hechas por el fabricante, quien también se encarga de instalarlas en el dispositivo antes de ponerlo a la venta. El objetivo de las

librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma “más eficiente” (Condesa, 2011).

Entre las librerías incluidas habitualmente encontramos openGL (motor gráfico), bibliotecas multimedia (formatos de audio, imagen y video), webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto), SQLite (base de datos), entre otras.

2.6.3 Entorno de ejecución

El entorno de ejecución de Android no se considera una capa en sí mismo, dado que también está formado por librerías. Su principal componente es la máquina virtual Dalvik (Condesa, 2011).

Es así que cuando se desarrolla una aplicación Android se la codifica en Java y se la compila en un formato específico para que Dalvik la pueda ejecutar.

La ventaja de esto es que las aplicaciones se compilan una única vez y de esta forma estarán listas para distribuirse con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera la aplicación.

Cabe aclarar que Dalvik es una variación de la máquina virtual de Java, por lo que no es compatible con el bytecode Java . Es por ello que Java se usa únicamente como lenguaje de programación, y los ejecutables que se generan con el SDK de Android tienen una extensión .dex que es específico para Dalvik.

2.6.4 Framework de aplicaciones

La siguiente capa está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik (Condesa, 2011).

A continuación se detalla cada uno de los componentes de la capa de framework de aplicaciones (Condesa, 2011).

- Activity Manager: Se encarga de administrar la pila de actividades de la aplicación así como su ciclo de vida .
- Windows Manager: Se encarga de organizar lo que se mostrará en pantalla. Básicamente crea las superficies en la pantalla que posteriormente pasarán a ser ocupadas por las actividades.
- Content Provider: Esta librería es muy interesante porque crea una capa que encapsula los datos que se compartirán entre

---

Bytecode Java: Es el tipo de instrucciones que la máquina de virtual de Java puede interpretar.
SDK: cliente enriquecido (Rich Client Platform RCP)
Ciclo de vida: Etapas seguidas por una aplicación desde su creación hasta su implementación y actualización.

aplicaciones para tener control sobre cómo se accede a la información.

- Views: En Android, las vistas son elementos que ayudan a construir las interfaces de usuario: botones, cuadros de texto, listas y hasta elementos más avanzados como un navegador web o un visor de Google Maps.
- Notification Manager: Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de estado. Un dato importante es que esta biblioteca también permite jugar con sonidos, activar el vibrador o utilizar los LEDs del teléfono.
- Package Manager: Esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes.

  Un paquete define la forma en que se distribuyen las aplicaciones Android, estos contienen el archivo .apk, que a su vez incluyen los archivos .dex con todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.
- Telephony Manager: Con esta librería podremos realizar llamadas o enviar y recibir SMS/MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.
- Resource Manager: Con esta librería podremos gestionar todos los elementos que forman parte de la aplicación y que están fuera

del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o capas.

- Location Manager: Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles y trabajar con mapas.
- Sensor Manager: Permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.
- Cámara: Con esta librería podemos hacer uso de la cámara del dispositivo para tomar fotografías o para grabar vídeo.
- Multimedia: Permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

2.6.5 Aplicaciones

En la última capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, las nativas (programadas en C o C++) y las administradas (programadas en Java), las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha instalado. En esta capa encontramos también la aplicación principal del sistema: Inicio (Home) o lanzador (launcher), porque es la que permite ejecutar otras aplicaciones mediante una lista y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso widgets, que son también aplicaciones de esta capa (Condesa, 2011).

2.7 MÁQUINA VIRTUAL DE ANDROID

La máquina virtual utilizada por Android es Dalvik, esta trabaja en la capa de ejecución y ha sido diseñada para optimizar la memoria y los recursos de hardware en el entorno de los teléfonos móviles o tabletas. Dalvik es una máquina virtual, diseñada por Dan Borntein y cumple la función de intérprete en la ejecución de archivos en el formato (*.dex) (García, 2010). Dalvik al igual que cualquier máquina virtual, permite que el código sea compilado a un bytecode independiente de la máquina en la que se va a ejecutar, y la máquina virtual interpreta este bytecode a la hora de ejecutar el programa.

Entre las características sobresalientes de Dalvik se encuentra (Delgado, 2012):

- Máquina virtual de aplicación o proceso. Es decir, cada aplicación se ejecuta en un proceso independiente y con su propia instancia de la máquina virtual.
- La arquitectura está basada en registros. Según varios estudios, se ha demostrado que una máquina virtual basada en registros, se ejecuta más rápido y requiere menos instrucciones que una máquina virtual basada en pilas.

La gran diferencia entre la máquina virtual de Java y Dalvik está en:

- La máquina virtual de java está basada en el uso de pilas.
- Dalvik está basada en el uso de registros, por lo cual es compatible con los procesadores de los teléfonos móviles.

El punto fuerte de Dalvik es la ejecución de clases compiladas en Java, para esto es necesario convertir el fichero (.class) original de Java a un fichero (.dex) entendido por Dalvik, esto es posible mediante el uso de la herramienta dex, que se encuentra disponible en el SDK de Android.

2.8 APLICACIONES ANDROID

Una aplicación Android es un programa informático capaz de ejecutarse sobre el sistema operativo Android y cuya finalidad es aportar algún tipo de servicio al usuario que lo manipula. Hay cinco bloques que componen una aplicación Android (Delgado, 2012) y se encuentran descritos en la tabla 4.

Tabla 4: Componentes de una aplicación Android

Componentes de una aplicación Android	
Activity	Una actividad es cada una de las pantallas dentro de una aplicación. Básicamente es la responsable de presentar los elementos visuales y reaccionar a las acciones del usuario. Toda actividad se inicia como respuesta a un Intent.
Intent	Se puede considerar como un deseo de realizar una tarea. Cuando se lanza un Intent el sistema busca que actividades son capaces de dar respuesta a ese Intent y elige la más adecuada.
Service	Un servicio es una tarea que se ejecuta durante periodos prolongados y no interacciona con el usuario.
Content Provider	Un proveedor de contenido es un almacén de información provisto de una API mediante la cual el usuario y aplicaciones pueden acceder al contenido sin necesidad de conocer los detalles del almacenamiento.
Broadcast Receiver	Permite escuchar y procesar Intens generales, como la hora o la entrada de un mensaje, y posteriormente lanzar las actividades o servicios deseados.

Nota. Fuente: Delgado, B. Vargas, M “Descubriendo la anatomía de una aplicación Android”

Hay que aclarar que no es indispensable ni mucho menos regla general el uso de los cinco componentes, de hecho la mayoría de aplicaciones usan combinaciones de las mismas. Hay que enfatizar, que todos los elementos de Android son un objeto y por tanto estos cinco componentes no son ninguna excepción (Delgado, 2012).

En general los elementos que componen una aplicación Android se encuentran en un archivo llamado AndroidManifest.xml. Este archivo XML contiene la declaración de los componentes de la aplicación, sus capacidades y requisitos. Mientras que el comportamiento de cada uno de ellos está definido en su correspondiente clase base, las cuales se definirían así:

- Android.app.Activity
- Android.app.Intent
- Android.app.Service
- Android.app.ContentProvider

Entonces cada elemento que se desea implementar será una subclase de la clase original, obteniéndose así los beneficios de la programación orientación a objetos como la herencia o la reutilización.

A continuación se detalla cada uno de los componentes:

2.8.1 Activity

Una activity o actividad en español, es una pantalla individual o interfaz dentro de la aplicación, su labor primordial es desplegar una interfaz compuesta de vistas y

responder a eventos solicitados en las mismas. Lo común es que una aplicación conste de múltiples pantallas, cada una de estas pantallas debe ser implementada como una "Activity". La navegación entre las pantallas se hace iniciando una nueva "Activity". En algunos casos, una "Activity" podría devolver un valor a la "Activity" anterior (Belatrix, 2012)

Básicamente la “Activity” es responsable de mostrar la interfaz gráfica, sin embargo ésta no forma parte de la “Activity”. Todos los elementos gráficos y visuales se definirán dentro de una vista o View, de este modo se asegura la separación del aspecto funcional del aspecto visual. Cada Activity se implementa como una única subclase que hereda de la clase android.app.Activity (Delgado, 2012). Hay dos métodos que casi todas las subclases de “Activity” implementarán (AndroidDevelopers, 2012):

- OnCreate (Bundle) es el método que permite inicializar la Activity y darle funcionamiento. Lo más importante aquí es la llamada a la función setContentView (int) como un recurso de diseño que define la interfaz de usuario, y el uso de la función findViewById (int) para recuperar los datos de la interfaz de usuario con los cuales trabajara la programación.

- onPause () es el método que permite al usuario parar la Activity que está realizando. Lo más importante, es que cualquier cambio realizado por el usuario en ese momento debe permanecer (por lo general permanece en la ContentProvider o agrupador de los datos).

2.8.1.1 Ciclo de Vida de una “Activity”

Las actividades dentro de una aplicación Android son manejadas mediante una pila de actividades. Es decir cuando una nueva actividad es iniciada pasa a la parte superior de la pila y la actividad anterior es pausada y almacenada en la pila de actividades un nivel más debajo de la actualmente utilizada, así cuando la nueva actividad termine, la siguiente actividad en la pila es reactivada.

Puesto que el sistema puede decidir finalizar una actividad pausada o detenida por motivos de memoria cada actividad es responsable de salvar su estado de forma que sea posible restaurarla tras haber sido pausada o detenida.

Una actividad puede tener cuatro estados (AndroidDevelopers, 2012):

- Activa: si la actividad está en primer plano en la pantalla, y por tanto en la parte superior de la pila de actividades está activa y ejecutándose.
- Pausada: si la actividad ha dejado de estar en primer plano pero aún es visible (una nueva actividad que no ocupa toda la pantalla o es traslucida ha sido llamada) pasa a estar pausada. Una actividad pausada se puede considerar completamente activa, conserva su estado y toda la información, pero puede ser finalizada por el sistema en casos extremos de falta de recursos.

- Detenida: si una actividad no es visible pasa a ser detenida. Puede mantener activa la información sobre su estado, aunque lo más probable es que sea finalizada por el sistema para liberar memoria y recursos. Cualquier aplicación y actividad que no sea visible será detenida para liberar recursos. Sin embargo la actividad debe guardar su estado actual de ejecución de forma que cuando la actividad vuelva a ser la primera en la pila pueda recuperar el estado en el que estaba por última vez.
- Finalizada: una vez que la actividad ha completado su ejecución libera todos los recursos que estaba utilizando y finaliza el ciclo. Si es llamada de nuevo iniciará el ciclo completamente a diferencia de una actividad detenida, la cual aún almacena su estado de ejecución a fin de recuperarlo cuando sea llamada de nuevo.

En general, el movimiento a través del ciclo de vida de una actividad es el siguiente (AndroidDevelopers, 2012):

```
public class Activity extends ApplicationContext {
   protected void onCreate(Bundle savedInstanceState);
   protected void onStart();
   protected void onRestart();
   protected void onResume();
   protected void onPause();
   protected void onStop();
   protected void onDestroy();
}
```

La figura 14 tomada desde la web oficial de desarrolladores de Android, describe el ciclo de vida de una actividad.

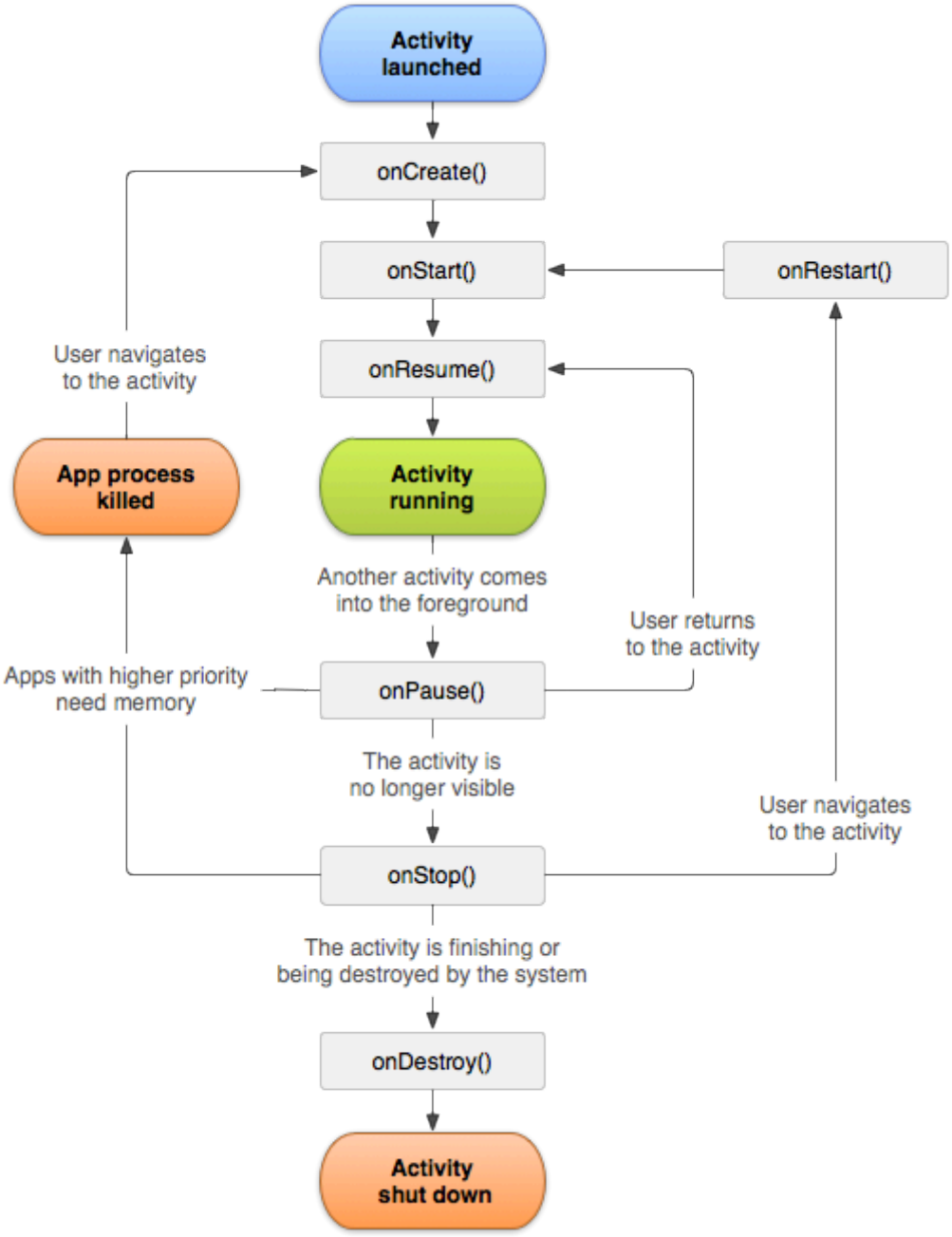

Figura 14: Ciclo de vida de una actividad (DevelopersAndroid, 2012)

Dónde:

- onCreate(): Es llamada cuando se crea la actividad y básicamente allí se la define.
- onRestart(): Llamado cuando la actividad ha sido detenida y es reiniciada la actividad de nuevo.
- onStart(): Es llamado cuando la actividad empieza a ser visible al usuario.
- onPause(): Es Llamado cuando la actividad actual va a ser pausada y la actividad previa será reanudada.
- onResume(): Es llamada cuando la actividad iniciara la interacción con el usuario.
- onStop(): Es llamado cuando la actividad ya no es visible para el usuario.
- onDestroy(): Llamado antes de que la actividad sea destruida por el sistema

2.8.2 Intent

Un Intent describe lo que una aplicación desea que se haga, podría considerarse como declarar una necesidad[24]. Básicamente Android utiliza al “Intent” para moverse de una pantalla a otra dependiendo de lo requerido por el usuario. Las dos partes más importantes de la estructura de datos de un "Intent" son la acción y los datos sobre los cuales se actuará (AndroidDevelopers, 2012).

Una actividad puede declarar sus Intent Filters tan genéricos o específicos como desee, teniendo en cuenta los datos asociados al Intent. Los Intent e Intent Filters son

una manera de encapsular tareas, aislar dependencias y asegurar la reutilización de funcionalidades (AndroidDevelopers, 2012).

Los Intents sirven como interfaz de comunicación entre aplicaciones, el equivalente a una API. Cualquier aplicación puede lanzar su Intent sin necesidad de conocer ningún detalle sobre la aplicación que lo va a resolver.

2.8.2.1 Objeto Intent

El objeto Intent se compone de tres partes: acción, categoría y datos (AndroidDevelopers, 2012), pero adicionalmente puede incluir un conjunto de elementos opcionales, aunque los tres nombrados harían la base de un Intent.

- Acción: es una cadena de texto completamente cualificada que indica la acción a realizar. Por ejemplo, android.intent.action.VIEW. Que sería una llamada a la vista.
- Categoría: Son los metadatos referidos al Intent, por ejemplo, android.intent.category.LAUNCHER indica que el Intent puede iniciar una aplicación si es requerido.
- Datos: están definidos en forma de un objeto de dirección (URI), por ejemplo, content://contacts/123, que sería la búsqueda de un contacto.

Cuando una actividad lanza un Intent, normalmente se trata de una invocación implícita. La actividad indica la tarea que desea realizar y el sistema asigna el Intent a la aplicación más apropiada (AndroidDevelopers, 2012).

2.8.2.2 Resolución de Intents

Hay tres tipos de componentes Android que pueden registrarse para resolver Intents:

- Activity,
- BroadcastReceiver, y
- Service.

Estos componentes se declaran capaces de tratar Intents registrando elementos <intent-filter> en el archivo AndroidManifest.xml.

Cada elemento <intent-filter> es parseado en un objeto IntentFilter. Cuando se instala el paquete, el sistema toma nota de los componentes que pueden tratar los diversos Intents. Una vez registrados, cada vez que se emita el Intent el sistema elegirá el elemento más adecuado para procesarlo según nuestras preferencias (AndroidDevelopers, 2012).

2.8.3 Service

Un servicio es un componente de aplicación que representa ya sea el deseo de una aplicación para realizar una operación de más larga duración, mientras que no interactúa con el usuario, o de proporcionar la funcionalidad para otras aplicaciones a utilizar (DevelopersAndroid, 2012).

Es decir un "Service" es una aplicación que se mantiene activada por un largo tiempo y no despliega una interfaz gráfica. Cada clase de servicio debe poseer su correspondiente declaración de <service> en el paquete AndroidManifest.xml.

Estos servicios pueden ser iniciados por los elementos Context.startService() y Context.bindService (). Los servicios pueden ser clasificados en locales y remotos. Un servicio local solo puede ser accedido por la aplicación que lo contiene.

Un servicio remoto permite a otras aplicaciones asociarse con él mediante el método bindService() para de esta forma tomar el control. Como se mencionó anteriormente, un servicio carece de interfaz gráfica, sin embargo tiene la capacidad de lanzar notificaciones para informar al usuario.

2.8.3.1 Ciclo de Vida

Una de las cualidades que Android ejerce sobre los servicios, es la perpetuidad de los mismos, es decir Android tratará de mantener el proceso de un servicio todo el tiempo que el servicio se ha iniciado o tiene clientes vinculados al mismo.

Cuando existe una crisis de memoria y existe la necesidad de matar a los procesos existentes, la prioridad de un proceso que aloja un servicio será la mayor de las siguientes posibilidades (DevelopersAndroid, 2012):

- Si el servicio está ejecutando código en los métodos onCreate() , onStartCommand() o OnDestroy(), entonces el proceso se ejecutara en primer plano, asegurando así que el servicio no pueda ser matado.
- Si se ha iniciado un servicio, pero no es visible, entonces el proceso es considerado menos importante que todos los procesos que están actualmente visibles para el usuario en la pantalla. Por lo general son pocos los procesos visibles para el usuario, aclarándose que esto no es una excusa para matar procesos, excepto en condiciones extremas de baja memoria.
- Si hay clientes vinculados al servicio, entonces los procesos ejecutados por los clientes pasan a ser igual de importantes que el servicio ejecutado. Es decir, si uno de sus clientes es visible para el usuario, todos los procesos relacionados al servicio en sí pasa a ser considerados visibles.
- Un servicio iniciado puede utilizar el método startForeground (int, Notification) para poner el servicio en primer plano, así el sistema lo considera algo que actualmente el usuario usa y por lo tanto no es un candidato para matar su proceso. Aun cuando exista poca memoria.

2.8.4 Content Provider

Un “Content Provider” es una clase que implementa un conjunto estándar de métodos para que otras aplicaciones almacenen o recuperen el tipo de datos que el "Content Provider" manipula.

Dicho de otra manera el “Content Provider” define una interfaz que recubre el método de almacenamiento y proporciona métodos de forma que cualquier aplicación pueda acceder al contenido sin necesidad de conocer la implementación del almacenamiento (DevelopersAndroid, 2012).

Un claro ejemplo de proveedores de contenido, que viene ya implementado en cualquier sistema Android, es la gestión de los contactos. Cualquier aplicación puede acceder a los datos de la agenda a través de la API del proveedor de contenido.

A partir de aquí, se puede acceder a los datos individuales mediante expresiones semejantes a las usadas en SQL. Se envía una consulta al proveedor de contenido y este devuelve un objeto cursor mediante el cual se pueden ir recolectando individualmente los datos, haciendo uso de las funciones get(...), update(...), insert(...), delete(...).

2.8.5 Broadcast Receiver

Un Broadcast Receiver es un tipo de servicio capaz de registrar Intents referidos a eventos generales, como la llegada de un mensaje de texto o una llamada.

Al igual que los servicios carece de interfaz de usuario, sin embargo es perfectamente posible hacer que el Broadcast Receiver ejecute una aplicación, logrando así poder programar actividades y servicios como respuesta a eventos.

Otra particularidad es que el procesamiento de Intents no es exclusivo. Cuando una aplicación lanza un Intent, una y solo una actividad será elegida para tratarlo.

Una actividad puede lanzar Intents para Broadcast Receivers, pero en ese caso no será posible establecer permisos o niveles de seguridad. El Intent es lanzado al sistema y todo aquel registrado para recibirlo lo recibirá.

2.9 HERRAMIENTAS

2.9.1 ECLIPSE

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular" (Wikipedia, 2010). Eclipse es un IDE (entorno integrado de desarrollo) para Java similar a Netbeans pero mucho más potente, este IDE es libre y fue creado originalmente por IBM , básicamente para codificar, compilar y ejecutar aplicaciones de forma amigable para el desarrollador.

Una de las ventajas de eclipse sobre otros IDE´s es su capacidad de ampliarse mediante (plugins), siendo así un marco de trabajo modular ampliable. De hecho,

---

Netbeans: es un entorno de desarrollo integrado, hecho principalmente para el lenguaje de rogramación Java.
IBM: Empresa fabricante de computadores.

existen complementos que permiten usar Eclipse para programar en PHP , Perl , Python , C/C++, aplicación Android, etc (Object Technology International, 2003). Eclipse cuanta con un SDK que incluye herramientas de desarrollo Java y un compilador de Java interno, siendo de este modo JAVA el lenguaje predominante en Eclipse.

Características

Eclipse trabaja bajo una plataforma de cliente enriquecido (Rich Client Platform RCP) y cuanta con los siguientes componentes:

- Plataforma principal: Ejecuta el inicio de Eclipse y la carga de plugins
- OSGi: Plataforma para la agrupación de estándares, define las especificaciones para diseñar plataformas compatibles que puedan proporcionar múltiples servicios.
- Standard Widget Toolkit (SWT): Es un conjunto de componentes para construir interfaces gráficas en Java, los widgets de Eclipse están implementados por una herramienta de widget para Java llamada Standard Widget Toolkit, a diferencia de la mayoría de las aplicaciones Java, que usan las opciones estándar Abstract Window Toolkit (AWT) o Swing.

PHP: Lenguaje de programación de uso general originalmente diseñado para el desarrollo web de contenido dinámico.
Perl: Lenguaje de programación
Python: Lenguaje de programación
Un cliente enriquecido consiste en proporcionar una interfaz gráfica, escrita con una sintaxis basada en XML, que proporciona funcionalidades similares a las del cliente pesado (arrastrar y soltar, pestañas, ventanas múltiples, menús desplegables).

- JFace: Es un conjunto de widgets que permiten el manejo de archivos, manejo de texto y editores de texto, JFace simplifica la construcción de aplicaciones basadas en SWT.
- El Workbench de Eclipse: Es un marco de interfaz de usuario que permite manejar vistas, editores, perspectivas y asistentes.

2.9.2 OPEN CV

OpenCV por sus siglas (Open Source Computer Visión Library) es una librería de código abierto para el tratamiento de imágenes, está diseñada en C y C++, sus principales tareas son el procesamiento de imágenes y la visión artificial en tiempo real (OpenCV, 2013), fue originalmente desarrollada por Intel y apareció a la luz con una primera versión alfa en 1999.

OpenCV es una librería multiplataforma, de hecho tiene la capacidad de ser ejecutada en Windows, Linux, Mac y Android, además mediante interfaces hace uso de los lenguajes C, C++, Java, Python, Octave y Matlab (OpenCVWiki, 2013) para el desarrollo.

Esta librería cuenta con más de 2.500 algoritmos optimizados (OpenCVWiki, 2013) y todos enfocados hacia el procesamiento de imágenes, además cuanta con más de 2,5 millones de descargas (OpenCV, 2013) y es utilizado en un sin fin de aplicaciones alrededor del mundo.

OpenCV está compuesta de 7 módulos (Rodriguez, 2012):

- opencv_core: Contiene las funcionalidades principales de la librería, como son las estructuras de datos básicas y las funciones aritméticas.
- opencv_imgproc: Contiene las principales funciones para manipulación de imágenes.
- opencv_highgui: Contiene las funciones para la lectura y escritura tanto de imagen fija como de vídeo, así como otras funciones de interfaz de usuario.
- opencv_features2d: Es un módulo con el que se consigue la detección de puntos y descriptores y además posee un framework para la unión de esos puntos.
- opencv_calib3d: Módulo que se encarga de la calibración de la cámara, se encarga de la estimación de la geometría para vistas estereoscópicas.
- opencv_video: Contiene la estimación de movimiento, funciones y clases para el seguimiento de objetos y la extracción de fondos.
- opencv_objdetect: Este módulo se encarga de la detección de objetos, como caras y personas.

2.9.3 SDK ANDROID

El SDK (Software Development Kit) de Android, es un kit de desarrollo de software, que proporciona las bibliotecas API y las herramientas de desarrollo necesarias para crear, probar y depurar aplicaciones para Android (DevelopersAndroid, 2012).

Características

Entre las herramientas más importantes del SDK de Android se encuentra: el "Android Emulator" y el "Android Development Tools", componentes que serán descritos a continuación:

- Emulador de Android

El "Android Emulator" es una aplicación que permite emular un dispositivo móvil de forma virtual en un computador. Este emulador permite desarrollar y probar aplicaciones Android sin necesidad de utilizar un dispositivo físico (DevelopersAndroid, Emulator Android, 2012).

El emulador de Android es tan potente que imita todas las características de hardware y software de un dispositivo móvil normal, excepto que no puede realizar llamadas telefónicas reales.

Existen distintas versiones del emulador tanto para Windows como para Mac Os o como para Linux.

- Android Development Tools

El "ADT" es un plugin que permite agregar poderosas extensiones al IDE de Eclipse, para el desarrollo y la depuración de aplicaciones Android.

2.9.4 NDK ANDROID

El NDK (Native Development Kit) de Android es un framework de desarrollo que permite implementar código nativo en una aplicación Android.

Su funcionalidad se basa en la utilización de librerías nativas e invocación de código C o C++ desde Java. En las últimas versiones del NDK se permite escribir aplicaciones Android completamente en lenguaje nativo (DevelopersAndroid, Herramientas SDK y NDK, 2012).

La gran diferencia entre el SDK y el NDK de Android se encuentra en que el primero no permite implementar código nativo, mientras que el NDK sí.

El NDK de Android se compone de las siguientes utilidades:

- Un conjunto de herramientas y archivos de construcción que se utilizan para generar bibliotecas de código nativo de fuentes en C y C + +.
- Una manera de integrar las correspondientes bibliotecas nativas en un archivo de paquete de aplicaciones (.Apk) para que las puedan implementar los dispositivos Android.
- Un conjunto de cabeceras y bibliotecas nativas del sistema.
- Documentación, ejemplos y tutoriales.

CAPÍTULO 3

RECONOCIMIENTO DE IMÁGENES

3.1 RECONOCIMIENTO DE IMÁGENES

El reconocimiento de imágenes es el proceso que busca identificar mediante la comparación de dos imágenes si ambas representan al mismo objeto o escena. Para ejemplificar de mejor manera el concepto se añade la figura 15.

Comparación 1

Imagen 1 = **Imagen 2**

Iguales

Comparación 2

Imagen 3 < > **Imagen 4**

Diferentes

Figura 15: Proceso de reconocimiento de imágenes

Dicho proceso inicia con la búsqueda de características especiales en la imagen, a estas características se las llama puntos de interés y básicamente son puntos que sobresalen dentro de la imagen a analizar.

El siguiente paso consiste en identificar como los puntos de interés encontrados se relacionan, este proceso recibe el nombre de descriptor y obtiene como salida un modelo matemático.

Por último el modelo resultante se compara con otros modelos matemáticos guardados en la base de datos los mismos que pasaron por un proceso de entrenamiento donde se obtuvieron los puntos de interés ideales de varias imágenes para luego poder ser comparados con modelos entrantes.

El proceso de comparación debe ser flexible, esto quiere decir que algunos modelos no se corresponderán al 100%; para que no importe tanto si la imagen está volteada, reducida o ligeramente torcida (tomando en cuenta que diferentes fotos de un mismo objeto serán distintas), la figura 16 ilustra el cambio de perspectiva que una imagen sufrió.

Figura 16: Imágenes desde diferentes perspectivas

Existen algoritmos especializados en el reconocimiento de imágenes tales como: FAST, SIFT, SURF, BRIEF y ORB. Pero en este proyecto se analizara únicamente los algoritmos: SIFT, SURF y ORB; siendo estos los más usados para la reconocimiento de objetos (Ziegler, 2012).

3.2 DESCRIPTORES DE IMAGENES

Para resolver el problema de la correspondencia o comparación entre imágenes en el proceso de reconocimiento de patrones se hace uso de un artificio llamado descriptor, entendiéndose como un descriptor al proceso que nombra y analiza una pequeña área alrededor de un punto de interés con el afán de obtener correspondencias entre estos pequeños sectores.

Para que el trabajo de un descriptor sea satisfactorio, el mismo debería seguir por lo menos tres etapas, entre las que se encuentra: la detección, la extracción y la determinación de correspondencias:

- Detección (feature detector): Se obtiene un conjunto de puntos de la imagen que cumplan con ciertas características (repetible, cuantificable, preciso, eficiente e informativo). Según el algoritmo aplicado pueden variar los puntos encontrados, pero siempre se debe cumplir la característica de “repetitibilidad”, es decir, lo que se desea es que aplicando el algoritmo a diferentes imágenes de la misma escena se obtengan aproximadamente los

mismos puntos. Cada uno de los puntos detectados se denomina indistintamente punto de interés o feature.

- Extracción de descriptores (feature descriptor): para cada uno de los puntos de interés detectados en la etapa anterior se genera una firma que lo identifique. La información contenida en la firma, como en el caso anterior, la determina el algoritmo aplicado. Esta firma es comúnmente llamada descriptor local, debido a que "describe" el área seleccionada de imagen correspondiente, y lo que se busca en este caso es la característica de discriminabilidad y la invariancia frente a los cambios que puedan afectar a la imagen. Discriminabilidad se refiere a que cada descriptor debe permitir, mediante comparación con otros descriptores, determinar si pertenece a la misma área de imagen, o a áreas distintas, con el mínimo error posible (Moreno, 2011).
- Determinación de correspondencias (matching): la última etapa es la búsqueda de un descriptor "correspondiente" para cada uno de los descriptores extraído anteriormente. Para esto se examina un conjunto de descriptores extraídos en forma previa desde la imagen o imágenes que forman la base de datos de búsqueda. Un par de descriptores formará una correspondencia, o match, cuando la distancia entre ellos sea mínima. El criterio exacto puede variar pero, en rasgos generales, la imagen almacenada en la base de datos que más descriptores en común tenga con la imagen de consulta, es la que se devuelve como resultado del problema de reconocimiento (Moreno, 2011).

Cabe recalcar que al menos una parte de los descriptores deben corresponderse con los de la imagen que se encuentra en la base de datos para que el algoritmo aplicado tenga éxito, destacándose así la flexibilidad del mismo y la posibilidad de que la imagen está volteada, reducida o ligeramente torcida. La figura 17 detalla el proceso que sigue un descriptor.

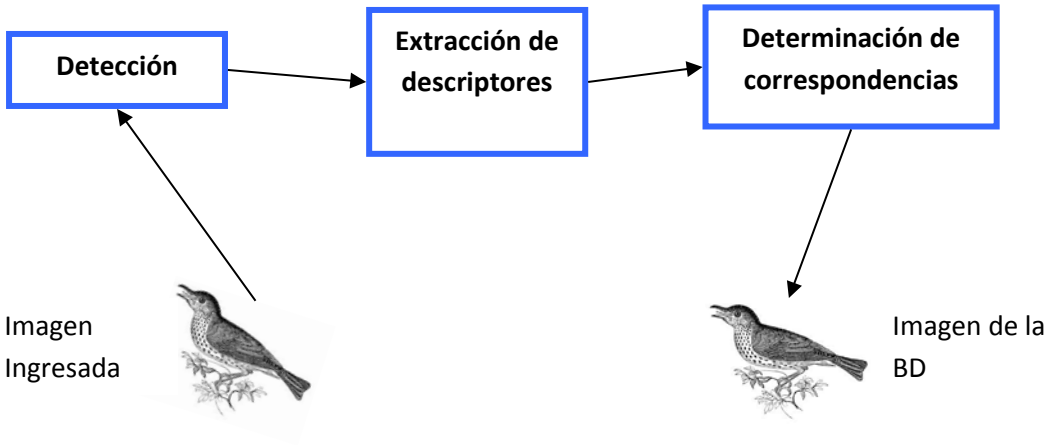

Figura 17 : Etapas de un descriptor

3.2.1 Propiedades de un descriptor ideal

Un descriptor de puntos de interés en su forma ideal debería cumplir las siguientes propiedades:

- Repetibilidad: Indica que el descriptor generado a partir de una imagen, debe ser el mismo independientemente del momento en el que se generó.
- Simplicidad: Hace referencia a la claridad y sencillez de las características extraídas de una imagen de modo que la interpretación de su contenido sea fácil.

- Discriminabilidad: Es la capacidad del descriptor para evitar la nula posibilidad de detectar correspondencias incorrectas.
- Eficiencia: Busca que los recursos consumidos por el descriptor sean aceptables, es necesario que guarde sus beneficios sin importar donde es utilizado dando buenos resultados en espacio y tiempo.
- Invariancia: Hace referencia a la casi nula sensibilidad o afectación por causas de deformaciones habituales en la imagen a comparar, el descriptor debería encontrar correspondencias aun cuando esté presente en la imagen ruido, cambios de iluminación, cambios de escala y rotación.

En la práctica un descriptor nunca logra acaparar todas estas propiedades al 100%, ya que los autores en la mayoría de los casos prefieren sacrificar una de estas tres propiedades para obtener beneficio en cualquiera de las otras. La figura 18 ilustra las deformaciones que una imagen puede sufrir y que un descriptor ideal debería resistir.

Figura 18 : Deformaciones en la imagen

3.2.2 Clasificación de los descriptores de imagen

Existe una amplia clasificación alrededor de los descriptores, desde aquellos descriptores que buscan correspondencias visuales hasta aquellos que buscan correspondencias audibles, este proyecto se enfocara en aquellos descriptores visuales que buscan analizar correspondencias en imágenes. A continuación se detallara la clasificación de los descriptores de imagen dependiendo de su nivel de abstracción:

- Descriptores de información de dominio específico: Son aquellos descriptores que buscan correspondencias informativas acerca de los objetos y eventos que posee una imagen, también llamados descriptores semánticos. Son utilizados en aquellas tareas en las que una descripción local del contenido de la imagen resulta más apropiada (Boullosa, 2011). Actúan sobre regiones de interés, previamente calculadas o identificadas, construyendo un vector de características de esa región que contiene información del punto de interés y de la región vecina a ese punto. Estos utilizan los descriptores de bajo nivel para cubrir el "gap" existente entre las características visuales disponibles y las diferentes categorías semánticas (J. Huang, 1999).
- Descriptores de información general: Son aquellos descriptores que buscan una correspondencia con respecto al color, formas, regiones, movimientos y texturas de una imagen; engloban los descriptores también llamados de bajo

---

Gap: Hace referencia a una brecha, una apertura o un espacio vacío comprendido entre dos puntos de referencia

nivel (Boullosa, 2011). Resumen el contenido de la imagen en un único vector o matriz de características.

A su vez los descriptores de información general tienen una sub -clasificación y son agrupados respecto del nivel de regiones sobre el cual actúan el descriptor.

- Descriptores globales: Se caracterizan por generar un único vector o matriz de características en el cual se abarca o resume el contenido de la imagen (Boullosa, 2011), su característica principal es la capacidad de encapsular una gran cantidad de información de la imagen requiriendo una pequeña cantidad de datos para describirla. Una virtud de este tipo de descriptores es su bajo coste computacional y las grandes prestaciones que se obtiene con él a pesar de su simplicidad

- Descriptores locales: Caracterizados por actuar sobre regiones de interés, previamente calculadas o identificadas, obteniendo como resultado un vector de características que describa dicha región de interés y la región adyacente al mismo o vecindario. Normalmente las regiones descritas se conocen como puntos de interés, también llamados puntos destacados o Puntos de interes (Boullosa, 2011), sin embargo estas regiones suelen referirse a bordes o pequeñas partes de la imagen. La agrupación de los vectores resultantes describirá la imagen en su totalidad.

A partir de este momento el presente proyecto se enfocará en los descriptores locales, esto debido a su referencia de eficiencia y su factibilidad de uso en equipos móviles. Los descriptores locales a analizar son SIFT, SURF Y ORB.

3.3 DESCRIPTORES LOCALES

Un descriptor local es un vector de características que es calculado sobre una pequeña región de interés de la imagen, este vector describe dicha región de interés y la región adyacente a la misma en dicha imagen.

Mientras mayor sea el número de puntos de interés y vectores mayor será la comprensión de la imagen a analizar facilitando así su correspondencia con alguna imagen a comparar guardada en la base de datos.

Cabe mencionar que los puntos y regiones de interés se deforman junto con la imagen, es así que su contenido no varía y además los descriptores (vectores de características) calculados son invariantes. Se pueden obtener correspondencias entre las dos imágenes al parear los descriptores parecidos, la figura 19 ilustra lo antes mencionado.

Figura 19 Proceso de un descriptor loca

3.3.1 Componentes de los Descriptores Locales

A continuación se detalla cada uno de los componentes:

- Detector de puntos de interés
- Generador de descriptores
- Generador de correspondencias
- Verificador de correspondencias

La figura 20 ilustra gráficamente los componentes de un descriptor local.

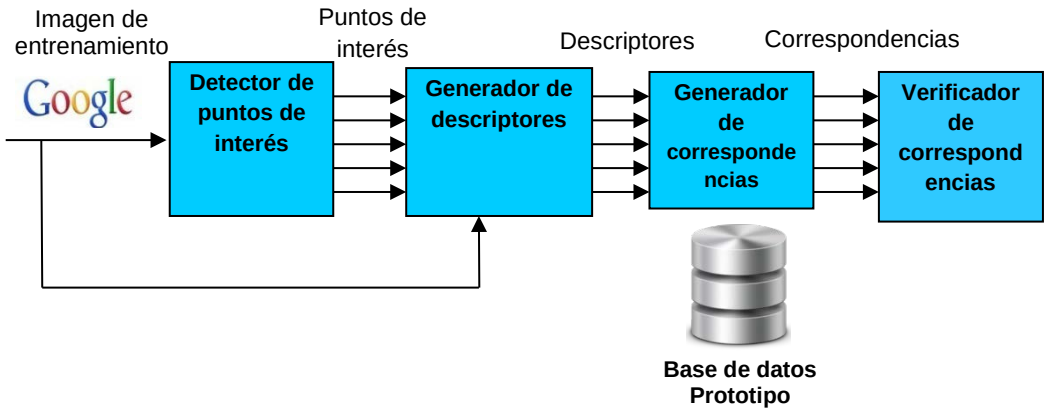

Figura 20: Componentes de un descriptor local

3.3.1.1 Detector de Puntos de Interés

Identifica aquellos puntos que sobresalen dentro de una imagen se los denomina indistintamente punto de interés, o feature. En esta etapa se calcula las posiciones de

los puntos de interés, y dependiendo del descriptor utilizado se lo hace de diferente manera, entre las formas más comunes se encuentra:

- Esquinas
- De máximos centro-contorno (manchas)
- Otros

Los Puntos de interés son, usualmente, máximos o mínimos locales de operadores diferenciales (filtros) aplicados sobre la imagen. Entre los descriptores de puntos de interés más utilizados y con mejores rendimientos se encuentran:

- Detector de Moravec (esquinas)
- Detector de Shi-Tomasi (esquinas)
- Detector de Harris (esquinas)
- SDoG (centro-contorno, multi-resolución)
- Harris-Laplace (esquinas, multi-resolución)
- SURF (multi-resolución)
- Otros específicos para aplicaciones

3.3.1.2 Generador de Descriptores

Como siguiente paso luego de la identificación de los puntos de interés se debe seleccionar una región de tamaño adecuado alrededor del punto, esta región recibe el nombre de “región de interés”.

Un descriptor caracteriza la región de interés de la imagen que rodea al punto de interés. Hay dos casos:

- Descriptores con 1 resolución
- Descriptores multi-resolución

A continuación se describen estos casos:

- Descriptores con 1 resolución

En este caso, el tamaño de todas las regiones invariantes es el mismo siempre y se elige “arbitrariamente”. La figura 21 ilustra un descriptor de una resolución.

Figura 21: Descriptores con una resolución

- Descriptores multi-resolución

En este caso, el tamaño de las regiones invariantes crece proporcionalmente con la escala y con el espacio en el cual se detectó el punto de interés

3.3.1.3 Generador de Correspondencias

Luego de la obtención de los puntos de interés y la generación de los descriptores de imagen se prosigue con el generador de correspondencias, cuyo objetivo es identificar correspondencias o similitudes entre los descriptores de la imagen entrante versus la imagen guardada en la base de datos. Las mejores correspondencias serán aquellos que tengan una similitud casi perfecta de la imagen entrante versus la imagen guardada en la base de datos.

3.3.1.4 Verificador de Correspondencias

Luego de generar las correspondencias se tiene como resultado un gran número de correspondencias entre puntos de la imagen entrante y la de la imagen guardada en la base de datos. Es aquí donde entra el verificador de correspondencias y su trabajo consiste en verificar si la imagen entrante está contenida en la base de datos, la figura 22 ilustra lo mencionado. Cada correspondencia se puede ver como una transformación geométrica que lleva un punto de interés de la imagen entrante y la imagen de la base de datos. La idea es encontrar un gran grupo de correspondencias que comparta la misma transformación lográndose así la identificación de la imagen.

Figura 22 : Proceso de reconocimiento de imágenes

3.4 DESCRIPTORES LOCALES SIFT, ASIFT, SURF y ORB

- SIFT

SIFT es un descriptor local creado por David Lowe en 1999, su característica primordial radica en ser invariante a deformaciones en la imagen como son escala, rotación y luminosidad.

Este descriptor basa su algoritmo de reconocimiento en la teoría de Gauss y trabaja con el espacio – escala Gaussiano y la diferencia Gaussiana para detectar puntos de interés y su respectiva orientación.

- ASIFT

ASIFT es un descriptor basado en el método SIFT, pero con la diferencia que simula tres parámetros: el zoom, el ángulo de la cámara en latitud y el ángulo de la cámara en longitud, y normaliza los otros 3 parámetros: la traslación, rotación y escala (Guoshen, 2009).

Básicamente lo que trata de hacer ASIFT es ser invariante casi al 100% a la deformación de la imagen, básicamente es el método SIFT pero con la mejora de zoom, y ángulo en longitud y latitud.

Diferencia Gaussiana: Función matemática que obtiene la diferencia de dos imágenes en la escala-espacio

- SURF

SURF (Speeded-Up Robust Features), fue presentado en el año 2006, como un detector y descriptor de puntos de interés para el reconocimiento de objetos. Siendo una de sus ventajas sobre otros métodos, el mejoramiento de la velocidad de cálculo y la robustez ante transformaciones de las imágenes. Estas mejoras se consiguen mediante la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos, mientras continúan siendo suficientemente característicos e igualmente repetitivos.

- ORB

ORB (Oriented FAST and Rotated BRIEF) es una mejora de SIFT, permite mejorar el rendimiento en tiempo real, entre las mejoras aportadas a SIFT este método incluye (Rublee. E, 2011):

- La adición de un componente de orientación rápida y precisa de FAST.
- El cálculo eficiente de características orientadas BRIEF.
- Análisis de la varianza y la correlación de orientación BRIEF.
- Un método de aprendizaje para la correlación de característica BRIEF en virtud de la invariancia rotacional, lo que lleva a un mejor rendimiento en las aplicaciones de vecino más cercano.

- Descriptores analizar

Luego de poseer una visión clara de los descriptores locales antes descritos, este proyecto descarta al descriptor ASIFT; porque es la utilización del descriptor SIFT varias veces en una imagen con el objetivo de analizar diferentes ángulos de la misma, esto provoca un incremento en el tiempo de procesamiento y un aumento en el uso de recursos, por tal motivo ASIFT no es aplicable en dispositivos móviles.

3.5 ESTADO DEL ARTE

TRABAJOS RELACIONADOS

En esta sección del proyecto se hará referencia a los artículos creados por los autores de los algoritmos SIFT, SURF Y ORB.

3.5.1 SIFT - Scale-Invariant Features

SIFT es un algoritmo de reconocimiento de imágenes, clasificado dentro de los descriptores locales y básicamente busca identificar las características sobresalientes de una imagen a partir de sus puntos de interés.

Estos puntos son invariantes frente a diferentes transformaciones como traslación, escala, rotación, iluminación y transformaciones afines (Boullosa, 2011).

SIFT fue publicado por David Lowe en 1999 (Lowe, 1999), en esta publicación Lowe propone un algoritmo para la extracción de características de una imagen, que permitan describir los objetos contenidos en ella (Lowe, 1999), denominado “Scale Invariant Feature Transform” (SIFT)

Este método permite obtener una serie de características invariables a la escala, rotación y en parte a cambios en la iluminación de una imagen.

El algoritmo descrito por lowe (Lowe, 1999) consta de 4 etapas definidas:

- Detección de extremos en la escala
- Localización de Punto de interés
- Asignación de la orientación
- Descriptor de los Puntos de interés

Estas etapas serán descritas a continuación:

3.5.1.1 Detección de extremos en la escala

La primera etapa del algoritmo realiza una búsqueda sobre las diferentes escalas y dimensiones de la imagen identificando posibles puntos de interés, invariantes a los cambios de orientación y escalado. Esto se lleva a cabo mediante la función DoG (Difference-of-Gaussian).

El objetivo de esta etapa es la obtención de puntos candidatos de la imagen que puedan ser identificados de forma repetida bajo diferentes vistas del mismo objeto.

El descriptor SIFT es construido a partir del espacio - escala Gaussiano de la imagen original, en el cual se pueden detectar de manera efectiva las posiciones de los puntos claves, invariantes a cambios de escala de la imagen.

El espacio - escala Gaussiano de una imagen $L(x, y, \sigma)$ es definido como la convolución de funciones 2D Gaussianas $G(x, y, \sigma)$ de diferentes valores σ con la imagen original $I(x, y, \sigma)$.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Dónde:

- L: scale-space
- G: Variable de escala Gaussiana
- I: Imagen de entrada
- x, y: Localización de coordenadas
- σ: Parámetro de escala
- $*$: Operador de convolución en x y y, entre la Imagen I y la Gaussiana G

El algoritmo utiliza la función DoG (Diferencia de Gaussiana) que se forma a partir de la derivada escalar de la Gaussiana escalada espacialmente. Esta función

DoG $D(x, y, \sigma)$ se obtiene mediante la sustracción de escalas posteriores en cada octava:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

Donde k es una constante multiplicativa del factor de escala y entendiéndose como Octava y escala:

- Octava: Conjunto de imágenes del espacio L con el mismo tamaño, pero con diferente valor de σ .
- Escala: Conjunto de imágenes del espacio L, con el mismo valor de σ , pero que con diferente tamaño

La función DoG es utilizada por varias razones. En primer lugar porque es una función eficiente en cuanto a coste computacional se refiere, las imágenes suavizadas $L(x, y, \sigma)$ son calculadas para la descripción de características en el espacio-escala (La figura 23 ilustra al espacio-escala), y por lo tanto, D puede obtenerse como una simple resta.

Al conjunto de las imágenes Gaussianas suavizadas junto con las imágenes DoG se le llama octava. El conjunto de las octavas es construido mediante el muestreo sucesivo de la imagen original por un factor de 2.

Cada una de las octavas es a su vez dividida en un número entero de subniveles o escalas s. Una vez se ha procesado una octava completa, la primera imagen de la

siguiente octava se obtiene mediante el muestreo de la primera de las imágenes de la octava predecesora con un valor de σ del doble respecto a la actual. Esto se traduce en una gran eficiencia del algoritmo para un número de escalas pequeño.

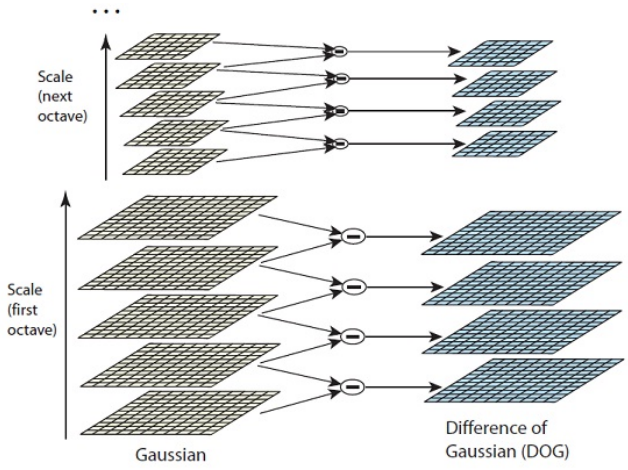

Figura 23: Creación del espacio – escala Gaussiano

Es importante tener en cuenta que la imagen original es expandida en el inicio del proceso para crear más puntos de muestreo que en la imagen original, por lo que la imagen resulta duplicada en tamaño antes de construir el primer nivel de la pirámide.

Dado que el espacio - escala L(x; y; σ) representa la misma información a diferentes niveles de escala, el modo particular del muestreo permite una reducción de la redundancia.

De esta manera se producen s + 3 imágenes por cada una de las octavas y por lo tanto s + 2 DoG imágenes donde se llevará a cabo la búsqueda de extremos. De

acuerdo con los resultados de Lowe, es el valor de s = 3 el que mejores resultados consigue. Las imágenes obtenidas son substraídas en parejas adyacentes para producir en las imágenes de la diferencia-de-gaussiana. Después de cada octava, las imágenes Gaussianas son muestreadas por un factor de 2, y se repite el proceso.

Los máximos y mínimos de las imágenes de la diferencia-de-gaussiana son detectados mediante la comparación de un pixel (marcado con X) con sus vecinos de las escalas actuales y adyacentes. Con esto se obtienen 6 imágenes Gaussianas suavizadas y 5 imágenes DoG por cada octava. Respecto del otro parámetro por determinar, σ referente al muestreo de la escala de suavizado.

Para detectar los máximos y los mínimos locales de cada punto de la imagen D(x; y; σ) se compara el valor de éste con el de los puntos vecinos, Si el valor resulta ser superior o inferior al de todos sus vecinos, se identifica el punto como máximo o mínimo local respectivamente. La figura 24 ilustra lo antes mencionado.

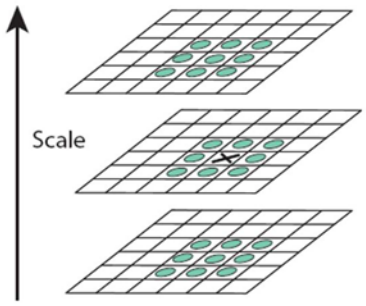

Figura 24: Localización de máximos y mínimos locales

3.5.1.2 Localización de Puntos de interés

Lo que busca esta etapa es seleccionar puntos de interés, de forma precisa, para lo cual aplica una medida de estabilidad sobre todos ellos para descartar aquellos que no sean adecuados.

Una vez los puntos clave candidatos han sido calculados, en la segunda etapa se realiza un estudio de su estabilidad. Los puntos no firmemente situados sobre los bordes o aquellos con bajo contraste son bastante vulnerables al ruido y por lo tanto no podrán ser detectados bajo pequeños cambios de iluminación o variación del punto de vista de la imagen.

Para solucionar este inconveniente Lowe utilizó estos criterios:

- Para eliminar los puntos con bajo contraste, se aplica un proceso de umbralización por el cual los puntos cuyo valor sea menor que dicho umbral D serán excluidos de la siguiente etapa por no considerarse suficientemente estables. En este proyecto se utiliza el valor de D = 0,03 recomendado por Lowe.

- Los puntos situados sobre bordes de manera difusa, conllevan un alto grado de inestabilidad incluso ante pequeños ruidos. Para llevar a cabo su eliminación, se utiliza la propiedad de la función DoG atendiendo a la gran curvatura que presenta en la dirección paralela al borde y la pequeña

curvatura que se observa en la dirección perpendicular. Estas respuestas tan características se pueden estudiar mediante el cálculo de la matriz del Hessiano sobre la localización y escala del punto en estudio:

$$\mathbf{H} = \begin{pmatrix} \mathbf{D_{xx}} & \mathbf{D_{yx}} \\ \mathbf{D_{xy}} & \mathbf{D_{yy}} \end{pmatrix}$$

Donde D es la imagen DoG D(x; y; σ) respecto de la escala s. Las derivadas se calculan mediante la resta del valor de los puntos vecinos. Se puede demostrar que la siguiente desigualdad permite la localización de los puntos en los bordes:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Por lo tanto aquellos puntos que no satisfagan dicha desigualdad serán descartados debido a su inestabilidad.

3.5.1.3 Asignación de la orientación

Esta etapa consiste en asignar una orientación a cada Punto de interés. Esta orientación provee de invarianza a la rotación de la imagen, basándose en las direcciones locales presentes en la imagen.

La característica principal de los puntos SIFT es que éstos son invariantes a una serie de transformaciones sobre las imágenes. La invarianza respecto de la rotación

se consigue mediante la asignación a cada uno de los puntos una orientación basada en las propiedades locales de la imagen y representando el descriptor respecto de esta orientación. Para cada uno de los puntos de interés se calcula la magnitud del gradiente, m, y su orientación, σ mediante las siguientes ecuaciones:

$$m(x,y) = \sqrt{(x+1,y) - L(x-1,y)^2 + \left(L(x,y+1) - L(x,y-1)\right)^2}$$

$$\theta(x,y) = \tan^{-1}\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}$$

A partir de las orientaciones del gradiente de todos los pixeles alrededor del Punto de interés, se forma un histograma de 36 contenedores (10 grados cada uno) para cubrir un total de $360°$. Cada muestra añadida al histograma, se pondera por la magnitud del gradiente y por una ventana circular de ponderación Gaussiana con $\sigma = 1.5$ veces mayor que la escala del Punto de interés, la figura 25 ilustra el proceso.

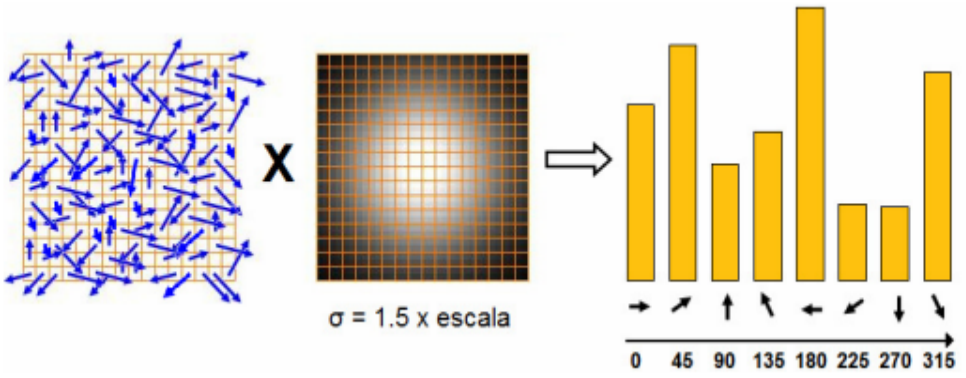

Figura 25: Zona de interés, gradiente e histograma

El contenedor cuyo valor es más alto se corresponde con la dirección dominante del gradiente y por lo tanto es elegido como orientación dominante. Sin embargo se ha de tener en cuenta la posibilidad de que exista más de una dirección dominante.

Es por ello que cualquier contenedor con un valor de más del 80% del valor de la magnitud principal se considerará también como dirección dominante. Los puntos que contengan más de una dirección dominante supondrán una mayor estabilidad al mismo.

Para una mayor precisión se utiliza una parábola para mediante la interpolación de los 3 valores más altos del histograma obtener el valor del pico.

Las orientaciones principales del histograma se asignan al punto de interés para que así el descriptor quede representado respecto de éstos.

3.5.1.4 Descriptor de los Puntos de interés

En las etapas previas, se ha asignado una localización, escala y orientación en la imagen para cada uno de los Puntos de interés. Estos parámetros forman un sistema de coordenadas 2D que describe localmente cada región de la imagen, y por lo tanto proporcionar invarianza a esos mismos parámetros.

El siguiente paso es, obtener un descriptor para cada zona de interés, lo cual aportara robustez a posibles variaciones de iluminación y cambios de puntos de vista 3D.

Para calcular el descriptor, se crea una ventana de 16x16 pixeles alrededor del Punto de interés. Esta ventana, es subdividida en subregiones de 4x4. La Figura 26 ilustra lo descrito.

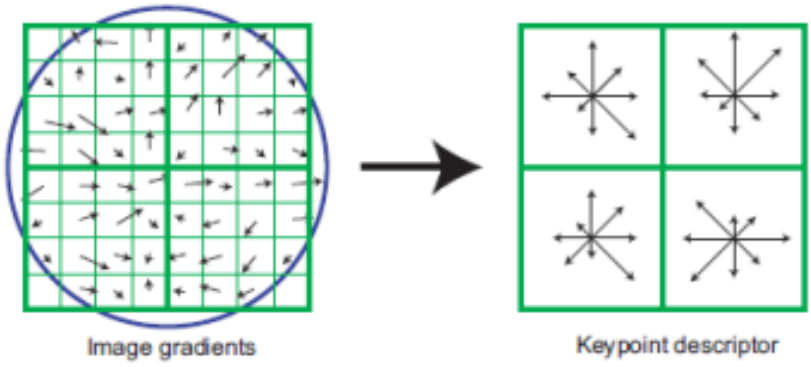

Figura 26: Ventana alrededor de un Puntos de interes

De cada subregión de 4x4, se calculan la magnitud y orientación del gradiente y a continuación se crea un histograma de orientación de 8 contenedores

El descriptor se forma a partir de un vector que contiene los valores de todas las orientaciones de las entradas del histograma, correspondientes a las longitudes de las flechas del lado derecho de la imagen.

Dado que son arreglos de 4x4 histogramas de orientación de 8 contenedores cada uno, tenemos que cada Punto de interés tiene un vector de características de 128 elementos. Para terminar el vector resultante se modifica para reducir efectos de cambios de iluminación.

Finalizadas las 4 etapas se prosigue con la comparación de los descriptores obtenidos:

3.5.1.5 Correspondencias entre descriptores (matching)

Dado las imágenes $\mathbf{I_1}$ e $\mathbf{I_2}$, se calcula sus correspondencias utilizando la estrategia del "vecino más próximo", la cual se define como el punto de interés con la distancia euclídea mínima del vector de características.

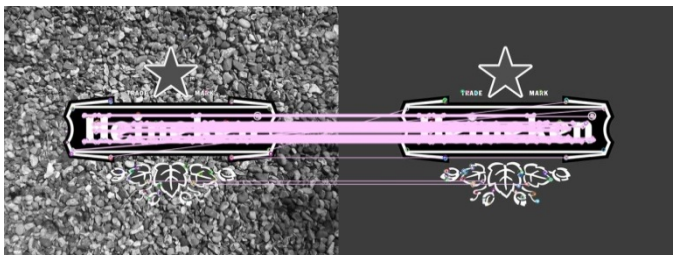

Figura 27: Correspondencia entre descriptores

Además se utiliza el criterio de máxima similitud. Este criterio establece que los mejores candidatos para realizar el matching con el punto clave p1 perteneciente a I1 cuyo vector de características es v1, son los puntos clave p01 y p02 pertenecientes a

I2 cuyos vectores de características v01 y v02 representan las distancias euclídeas mínimas d1 y d2 respectivamente respecto de v1.

Si la relación d1=d2 entre las distancias mencionadas es suficientemente pequeña, entonces se establece el matching entre los puntos p1 y p01 pertenecientes a cada una de las imágenes.

De acuerdo con Lowe, se establece un umbral de 0.76 para el ratio d1=d2. Esta estrategia de matching recibe el nombre de “el vecino más próximo”.

Finalmente la puntuación o score entre las dos imágenes se obtiene mediante una relación que tiene en cuenta el número total de puntos encontrados entre ambas imágenes. La figura 28 describe el flujo de procesos seguido por SIFT.

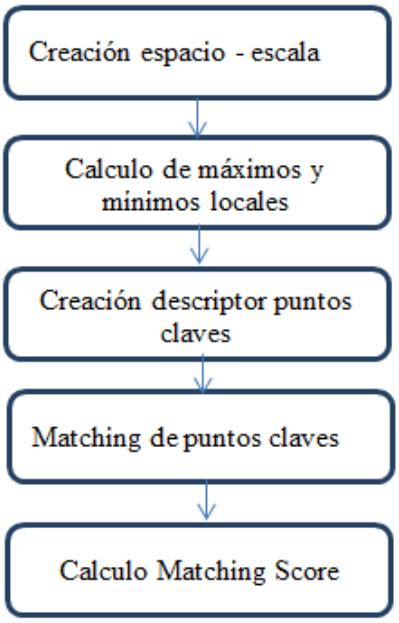

Figura 28: Flujo de procesos de SIFT

3.5.2 SURF: Speeded Up Robust Features

El descriptor SURF, cuyo acrónimo hace referencia al título, Speeded-Up Robust Features, fue desarrollado por Herbert Bay en el año 2006, por Herbert Bay, Tinne Tuytelaars y Luc Van Gool, como un detector y descriptor de puntos de interés para el reconocimiento de objetos. Siendo una de sus ventajas sobre otros métodos, el mejoramiento de la velocidad de cálculo y la robustez ante transformaciones de las imágenes. Estas mejoras se consiguen mediante la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos, mientras continúan siendo suficientemente característicos e igualmente repetitivos.

A continuación se detalla el algoritmo propuesto en SURF; el cual consta de las siguientes etapas.

- Detección de puntos de interés o Puntos de interes
- Asignación de la orientación
- Extracción de los descriptores

3.5.2.1 Detección de puntos de interés o Puntos de interés

El objetivo de esta etapa es el de detectar Puntos de interés, haciendo uso de una matriz Hessiana El motivo de usar la matriz hessiana es por su rendimiento en cuanto a la velocidad de cálculo y a la precisión.

- Imagen Integral

Antes de detallar las operaciones para localizar los Puntos de interés, es importante mencionar el concepto de la "imagen integral", la cual consiste en la sumatoria de las subregiones de una imagen.

Dado una imagen $I(x,y)$, la imagen integral $S(x,y)$, se calcula con la siguiente fórmula:

$$S(x,y) = \sum_{i=1}^{1\leq x}\sum_{j=1}^{j\leq y} I(x,y)$$

Uno de los beneficios de la imagen integral, es que permite hacer una convolución con un cuadrado (box filter) de cualquier tamaño con un costo computación bajo.

- Puntos de interés basados en la matriz Hessiana

Dado un punto $\mathbf{p} = (\mathbf{x},\mathbf{y})$ en una imagen $\mathbf{I}$, la matriz Hessiana $\mathbf{H}(\mathbf{p},\boldsymbol{\sigma})$, del punto p a una escala σ se define como:

$$\mathbf{H}(\mathbf{p},\boldsymbol{\sigma}) = \begin{bmatrix} \mathbf{L}_{xx}(\mathbf{p},\boldsymbol{\sigma}) & \mathbf{L}_{xy}(\mathbf{p},\boldsymbol{\sigma}) \\ \mathbf{L}_{xy}(\mathbf{p},\boldsymbol{\sigma}) & \mathbf{L}_{yy}(\mathbf{p},\boldsymbol{\sigma}) \end{bmatrix}$$

Donde $L_{xx}(p,\sigma)$ representa la convolución de la derivada parcial de segundo orden de la gaussiana $\frac{\partial^2}{\partial x^2} g(\sigma)$ con la imagen I en el punto p y del mismo modo para $L_{xy}(p,\sigma)$ y $L_{yy}(p,\sigma)$.

Las derivadas de segundo orden de la gaussiana son calculas con la imagen integral, lo cual hace que esta operación tenga un costo computación muy bajo. Las aproximaciones de las derivadas parciales se denotan por D_{xx}. D_{xy} y D_{yy}.

Para calcular el determinante de la matriz Hessiana (la cual representa una escala), se aplica la siguiente formula:

$$\det(H_{aprox}) = D_{xx}D_{yy} - (0.9D_{xy})^2$$

Donde el valor de 0.9 es la aproximación del filtro gaussiano.

- Representación del espacio de escala

Los Puntos de interés necesitan ser encontrados en diferentes escalas, debido a que las mayorías de correspondencias se hacen con imágenes de diferente escala.

El espacio de escala está divido en octavas las cuales están compuestas por un número fijo de escalas. Para construir las escalas se debe calcular varios pisos del $\det(H_{aprox})$ con filtros de varios tamaños.

El primer filtro que se aplica es de 9x9, esta es considerada como la máxima resolución espacial. Las siguientes capas se obtienen incrementando el valor del filtro en 6 y así sucesivamente hasta completar la octava. La figura 29 ilustra de forma gráfica lo antes mencionado.

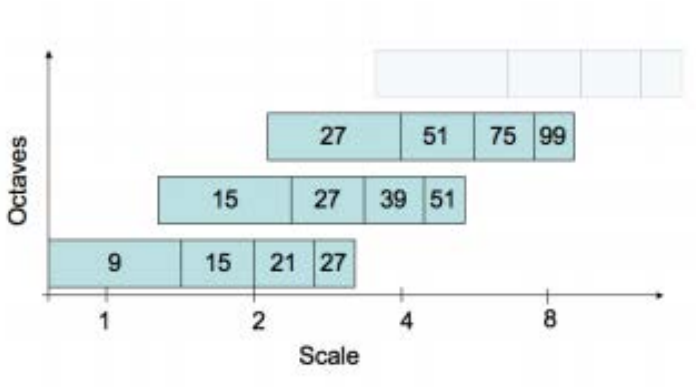

Figura 29: SURF escala y octavas

Como muestra en la figura 29, el incremento del filtro en la misma octava es el doble respecto a la octava anterior (desde 6, 12, 24, 48) y para el primer filtro de cada octava es el segundo de la octava anterior.

- Localización de puntos de interés:

Para la localización de los Puntos de interés en todas las escalas, se elimina los puntos que no cumplan la condición de máximo en un vecindario de $3x3x3$. De esta manera, el máximo $\det(H_{aprox})$ es interpolado en la escala y posición de la imagen.

3.5.2.2 Asignación de la orientación

En esta etapa se asigna una orientación a cada uno de los Puntos de interés obtenidos en la etapa previa, con el propósito que los Puntos de interés sean invariantes ante cambios de orientación de la imagen. Para esto, primero se calcula la respuesta de "Haar wavelet" en la dirección x e y, dentro de una área circular de radio $6s$ alrededor del Punto de interés, donde s es la escala del Puntos de interés detectado.

Una vez realizado el cálculo para todos los puntos vecinos, las respuestas son representadas como un vector en el espacio, colocando las respuestas horizontal y vertical en el eje de las abscisas y ordenadas respectivamente. A continuación se calcula la orientación domínate de cada sector sumando todos los resultados dentro de una ventana deslizante de orientación que cubre un ángulo de $\frac{\pi}{3}$ (60°). La figura 30 muestra la asignación de la orientación.

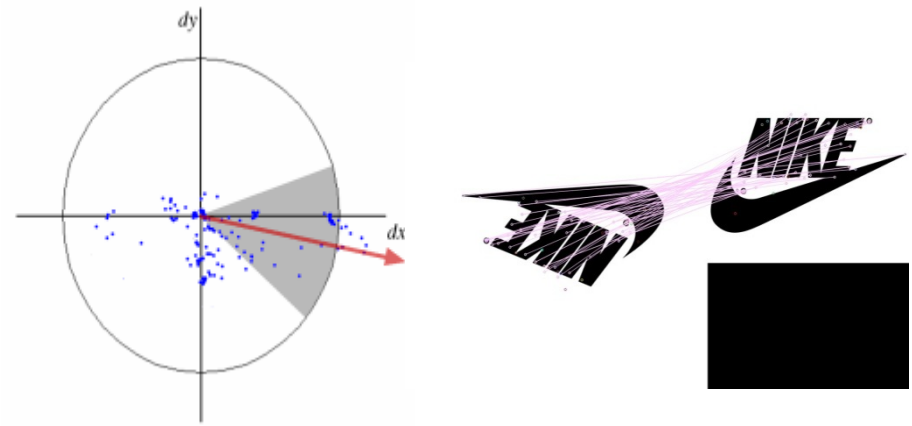

Figura 30: Asignación de Orientación

Finalmente la orientación final del Punto de interés, es aquella cuyo vector sea el más grande dentro de los 6 sectores ($\frac{360}{\pi/3} = \frac{360}{60} = 6$) en los que se ha divido el área circular alrededor del Punto de interés.

3.5.2.3 Extracción de los descriptores

Para calcular los descriptores de los Puntos de interés se utiliza una región cuadrada de $20s$ alrededor del Punto de interés y orientada en relación a la orientación obtenida en la etapa anterior.

Esta región posteriormente se la divide en subregiones de 4x4 y para cada una de estas se calcula la respuesta de Haar wavelet, en puntos con una separación de muestreo de 5x5 (en las direcciones horizontal y vertical).

Se considera como d_x y d_y a las respuestas de Harr en las direcciones horizontal y vertical respectivamente, la figura 31 muestra la extracción de un descriptor afectado por la variación de movimiento.

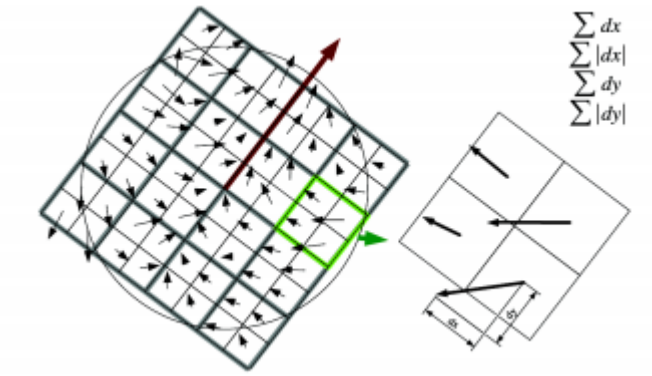

Figura 31: Extracción de descriptores

En cada una de las subregiones se suma las respuestas y se obtiene un valor representativo de $\mathbf{d_x}$ y $\mathbf{d_y}$, al mismo tiempo también se calculan los valores absolutos $|\mathbf{d_x}|$ y $|\mathbf{d_y}|$, y asi al final se tiene para cada subregión un vector $\mathbf{v}$, tal que:

$$v = \left(\sum d_x . \sum d_y . \sum |d_x| . \sum |d_y| .\right)$$

Finalmente el descriptor es un vector resultado de la concatenación de los vectores $\mathbf{v}$ de cada subregión de 4x4, por ende su dimensión es de 64 ($\mathbf{4x4x4 = 64}$). Al llegar a este puno se da por terminado el algoritmo SURF.

Finalizadas las 4 etapas se prosigue con la comparación de los descriptores obtenidos:

3.5.2.4 Correspondencias entre descriptores (matching)

En esta sección, al igual que en el caso de SIFT se calculan las correspondencias entre dos imágenes a partir de los vectores de características. La estrategia utilizada es la del “vecino más próximo”. El umbral establecido para la distancia euclídea mínima es de 0,7

3.5.3 ORB (Oriented FAST and Rotated BRIEF)

El descriptor local ORB (Rublee. E R. V., 2011) fue publicado en el ICCV (International Conference on computer Vision) del 2011 como una alternativa a los descriptores SIFT y SURF, siendo una de sus ventajas sobre otros descriptores la velocidad de cómputo.

Como lo indica su nombre, ORB está basado en los siguientes algoritmos.

- FAST (Features from Accelerated Segment Test): para la detección de Puntos de interes (Rosten, 2011).

- BRIEF (Binary Robust Independent Elementary Features): para la extracción de descriptores (Calonder M, 2011).

FAST y BRIEF son algoritmos que mejoran notablemente la velocidad de cómputo, sin embargo carecen de un operador de orientación e invarianza a la rotación respetivamente[38]. Para solventar estos dos problemas ORB propone:

- oFAST (FAST Punto de interés Orientation): Al algoritmo FAST se le agrega un componente de orientación para los Puntos de interés detectados.

- rBRIEF (Rotation-Aware Brief): Al algoritmo BRIEF le agrega invarianza a la rotación y mejora la velocidad de cálculo.

3.5.3.1 Detección de Puntos de interés utilizando Fast

En esta etapa se obtiene un conjunto de Puntos de interés utilizando el detector FAST y a continuación a cada uno de ellos se le asigna una orientación.

- FAST

Los autores de FAST (Rublee. E R. V., 2011) proponen agrupar 16 pixeles alrededor de un punto p (pixel candidato). Y posteriormente considerar al punto p como Punto de interés, siempre y cuando este cumpla con las siguientes consideraciones:

- Existen al menos n píxeles contiguos en el círculo que tengan una intensidad mayor que la intensidad del píxel candidato más un umbral t .
- Existen al menos n píxeles contiguos en el círculo que tengan una intensidad menor que la intensidad del píxel candidato menos el umbral t .

Por referencia de los autores se sabe que los valores óptimos de n y t son 9 y 20 respectivamente (Rublee. E R. V., 2011). A continuación se eliminan todos los Puntos de interés detectados que se encuentren a lo largo de los bordes utilizando el filtro de “Harris Corner ” (Harris, 1998).

Filtro de Harris Corner: Función que obtiene datos relevantes de una imagen a partir de sus esquinas

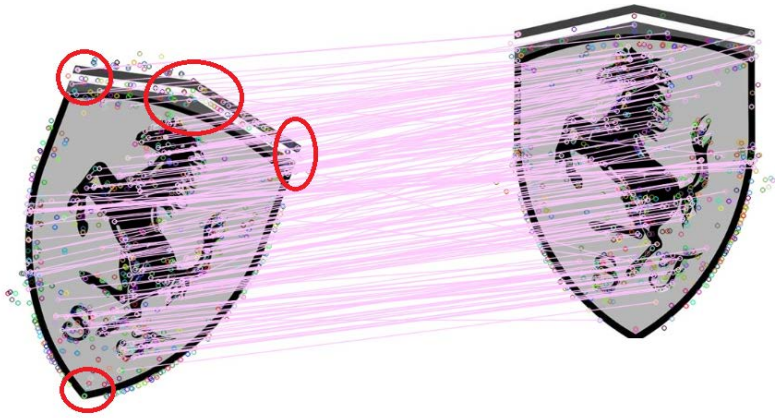

Figura 32: Aplicación del filtro de Harris Corner

- Espacio de escala con FAST

FAST no provee un espacio de escala, por lo que se crea una pirámide de imágenes para posteriormente obtener Puntos de interés de cada nivel de la pirámide aplicando las operaciones descritas en el apartado anterior, obteniendo así invarianza a la escala.

- Asignación de la orientación con FAST

Para asignar una orientación a cada uno de los Puntos de interés detectados, se usa el enfoque de la “intensidad del centroide” (Rublee. E R. V., 2011).

Basados en la suposición que, la posición del centroide de una esquina difiere de la posición central de esta, un vector orientación puede ser calculada desde el centro de la esquina al centroide para asignar una orientación al Punto de interés.

Así la orientación θ del Punto de interés está dada entonces por el ángulo de dicho vector y puede ser calculada como:

$$\theta = \text{atan}2(m_{01}, m_{10})$$

Con m_{pq} siendo el momento de la región de interés.

3.5.3.2 Extracción de descriptores con rBRIEF

Esta etapa, primero calcula un descriptor BRIEF dirigido y posteriormente se introduce una fase de aprendizaje para encontrar test binarios menos correlacionados, dando como resultado el descriptor denominado rBRIEF.

- BRIEF

Este descriptor está constituido por una cadena de bits, la cual es el resultado de un conjunto de test binarios calculados a partir de una imagen integral (Calonder M, 2011). Sea p, un parte de una imagen suavizada. Un test binario τ está definido por:

$$\tau(p;x,y) = \begin{cases} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{cases}$$

Donde $p(x)$ es la intensidad de p en el punto x. El descriptor está definido como un vector de n test binarios.

$$f_n(p) = \sum_{1 \le i \le n} 2^{i-1} \tau(p; x_i, y_i)$$

Para los test se utiliza la distribución Gaussiana alrededor del centro del patch. También se selecciona un vector de longitud $n = 256$.

Antes que se ejecuten los test la imagen es suavizada utilizando una imagen integral, donde cada punto de prueba es una sub-región de $5x5$ de una patch de pixeles de $31x31$ (Rublee. E R. V., 2011).

- BRIEF dirigido

Para obtener invarianza a la rotación se dirige el descriptor BRIEF de acuerdo a la orientación de los Puntos de interés. Para cualquier descriptor de n test binarios localizados en (x_i, y_i), se define la matriz de $2xn$.

$$S = \begin{pmatrix} x_1, \ldots, x_n \\ y_1, \ldots, y_n \end{pmatrix}$$

Con la orientación θ del patch y la correspondiente matriz de rotación R_θ, se construye una versión dirigida S_θ de S :

$$S_\theta = R_\theta \mathbf{S}$$

Así el operador BRIEF dirigido está dado por:

$(x_i, y_i) \in S_\theta$

Con el fin de obtener características discriminativas el autor (Rublee. E R. V., 2011) diseño la selección de punto pares con partes de una imagen alrededor del Punto de interés , de tal manera que el descriptor resultante no este correlacionado con el resultado de cada test binario.

Al finalizar la extracción se procede a comprar los descriptores, ORB al igual que SIFT utiliza la estrategia del vecino más cercano.

3.5.3.3 Correspondencias entre descriptores (matching)

Para el cálculo de correspondencias de dos imágenes cuyos descriptores fueron encontrados con ORB se utiliza la estrategia del “vecino más cercano” igual que SIFT y SURF pero con diferente algoritmo para cálculo de la distancia entre descriptores.

Las “distancia de haming” se aplica para encontrar la distancia entre los descriptores ya que sus valores de cadenas de bits (a diferencias de los valores de los descriptores de SIFT y SURF que son flotantes).

3.6 VENTAJAS Y DESVENTAJAS SIFT, SURF Y ORB

Como resumen final de este capítulo y con el objetivo de sustentar los conocimientos adquiridos sobre los descriptores SIFT, SURF Y ORB la tabla 5 plantea una síntesis de las ventajas y desventajas de los algoritmos antes mencionados.

Tabla 5: Ventajas y desventajas de los descriptores

	Ventajas	Desventajas
SIFT	Invarianza respecto de rotaciones, translaciones, escala y cambios de iluminación. repetibilidad.	Alto coste computacional. Tamaño del descriptor mucho mayor que los anteriores. Discretización de los filtrosgaussianos. Dependiente del tamaño
SURF	Mayor robustez y velocidad de cálculo respecto del descriptor SIFT.	Menor tamaño que el descriptor SIFT pero todavía incomparable respecto de los anteriores. Dependiente del tamaño de la imagen.
ORB	Mejor rendimiento en tiempo real, ideal para dispositivos de baja potencia sin aceleradores de GPU.	Dependiente del tamaño de la calidad de la imagen, demasiado sensible a la rotación de imágenes

Nota. Fuente: Tipantuña, L. Ñauñay M (2013)

CAPÍTULO 4

INVESTIGACIÓN
EVALUACIÓN DE DESCRIPTORES

4.1 INTRODUCCIÓN

Este capítulo detallará los ajustes realizados a cada descriptor así como también un estudio comparativo de los mismos en base al objetivo general planteado en el CAPÍTULO 1. Básicamente se realizará un análisis de eficiencia y eficacia de los algoritmos SIFT, SURF Y ORB. Finalmente se formulan conclusiones en base a los resultados obtenidos en el estudio comparativo y se selecciona el algoritmo con los mejores resultados. La figura 33 representa el diagrama de las etapas que componen este capítulo.

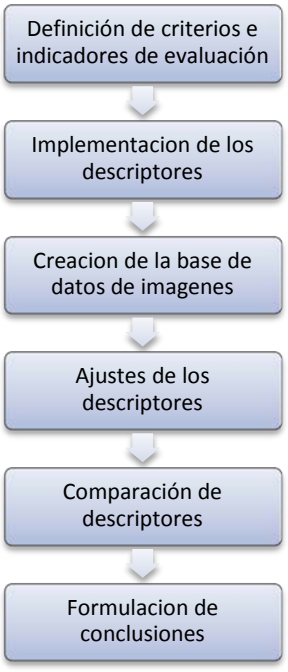

Figura 33: Diagrama de las etapas de evaluación

Para el desarrollo de la evaluación se utiliza un teléfono inteligente con sistema operativo Android versión 4.1 o Jelly Bean. La figura 34 resume las características técnicas de este dispositivo.

GALAXY NEXUS – CARACTERÍSTICAS

Sistema operativo	**CPU**
Android OS, v4.1 Jelly Bean	Procesador de doble núcleo a 1,2 GHz
Dimensiones	**RAM**
135.5 x 67.94 x 8.94 mm	1GB
Peso	**Memoria**
135g	16GB
Tamaño y tipo de pantalla	**Batería**
Super Amoled HD (1280 x 720 píxeles) de 4,65 pulgadas	Standard, Li-Ion 1750 mAh Hasta 17 h 40 min (2G) / Hasta 8 h 20 min (3G)

Figura 34: Características técnicas Galaxy Nexus

4.2 CRITERIOS E INDICADORES DE EVALUACIÓN

Para evaluar el desempeño de los descriptores SIFT, SURF y ORB, es necesario definir criterios e indicadores de evaluación que permitan analizar los resultados obtenidos en las pruebas técnicas.

Las conclusiones formuladas a partir de los resultados de la evaluación serán más completas y precisas en función de la independencia de los diferentes aspectos comparados. Por tal motivo, se ha decidido dividir los criterios e indicadores en dos grupos:

- Grupo de criterios e indicadores de eficiencia y
- Grupo de criterios e indicadores de eficacia.

Es importante aclarar que se tomaron en cuenta dichos criterios de evaluación debido a que la combinación de los mismos indica la efectividad de un algoritmo.

A continuación se describen los criterios e indicadores seleccionados:

4.2.1 Criterios e indicadores de eficiencia

Uno de los principales problemas que enfrentan los teléfonos inteligentes (independiente del sistema operativo que tenga: Android, IOS o Windows Phone) es la cantidad limitada de recursos de: CPU, memoria RAM y batería (Carroll, 2010) para

el uso por parte de las aplicaciones que residen en estos dispositivos. Por lo tanto, el uso eficiente de estos recursos se convierte en un elemento clave en la evaluación de una aplicación móvil especialmente si se trata de una aplicación que requiere un alto procesamiento computacional como juegos o aplicaciones de visión por computador (reconocimiento facial o reconocimiento de objetos).

A continuación, se detallan algunos criterios e indicadores que permitirán medir el uso de recursos cuando un descriptor se está ejecutando sobre un teléfono inteligente con sistema operativo Android.

4.2.1.1 Número de Puntos de interés

Se define la variable K_t, la cual representa el número promedio de Puntos de interes extraídos de una imagen de entrada. Esta métrica también nos permitirá verificar si los algoritmos cumplen con la propiedad de repetibilidad.

4.2.1.2 Tiempo de ejecución

Se define al tiempo de ejecución como el intervalo de tiempo que tarda el descriptor local en procesar una imagen hasta obtener sus características. Esta dada por la siguiente ecuación:

$$T_{total=} T_{detector} + T_{extractor}$$

Donde $T_{detector}$ representa el tiempo utilizado para detectar los puntos de interés y $T_{extractor}$ el tiempo utilizado para la extracción del vector de características.

4.2.1.3 Consumo de batería

En los teléfonos inteligentes la capacidad de la batería es muy limitada debido al tamaño y peso que poseen, esto implica que la eficiencia energética sea muy importante para su uso. Por lo tanto, una gestión óptima de consumo de energía es crítica en este tipo de dispositivos (Carroll, 2010) (MalikYasir, 2012).

Basados en la premisa anterior, se hace prioritario la medición del consumo de batería por parte de los descriptores locales, con el fin de determinar cuál de ellos hace mejor uso de este recurso.

Sea B_c el porcentaje de batería consumida por un descriptor local para procesar un conjunto de n imágenes de muestra, se define la siguiente ecuación:

$$B_t = \frac{B_c}{n}$$

Donde B_t es el porcentaje de batería que consume el descriptor local para procesar una imagen.

4.2.1.4 Pesos de los descriptores

Este aspecto es muy importante para aplicaciones que necesiten almacenar el vector de características generado por el descriptor para uso futuro. El tamaño de este vector varía dependiendo del descriptor utilizado y el número de características generadas. Esta métrica de define como:

$$P_t = KB\ necesarios\ para\ almacenar\ un\ vector\ de\ características$$

4.2.2 Criterios e indicadores de eficacia

La eficacia hace referencia al rendimiento o capacidad que posee un descriptor para identificar una imagen, el estudio comparativo de la eficacia sobre descriptores locales es una gran problemática, debido a que se trabaja sobre decisiones binarias .

Para solventar la problemática menciona existen varios criterios, entre los que sobresalen los siguientes: curvas ROC (F. Provost, 1998), curvas de coste (Holte, 2000) y precisión vs recall *(PR)* siendo esta ultima la sugerida como mejor alternativa por Mikolajczyk y Schmid en el artículos “A performance evaluation of local descriptors” (Schmid., 2005).

Decisiones binarias: Es una estructura de datos utilizada para representar una función booleana.

4.2.2.1 Precisión vs recall(PR)

Este enfoque se basa en el número de aciertos y fallos respecto de las correspondencias establecidas entre dos imágenes. A continuación se describe sus componentes:

- $\#\ correct\ matches$: son aquellas imágenes comparadas que, perteneciendo a la misma escena o grupo interrelacionado que la imagen original dentro de la categoría analizada, son identificadas de forma acertada como imágenes relacionadas.
- $\#false\ matches$: son aquellas imágenes que tras el proceso de comparación han sido identificadas como relacionadas con la imagen original de forma incorrecta, pues no pertenecen a la misma escena.
- $\#correspondences$: representa el número de imágenes que están relacionadas con la imagen original, es decir, que pertenecen a la misma escena.

4.2.2.2 Precisión (P)

Porcentaje de las imágenes que verdaderamente están relacionadas a la imagen original.

$$P = \frac{\#\ correct\ matches}{\#correct\ matches + \#false\ matches}$$

4.2.2.3 Recall (R)

El porcentaje de imágenes que fueron identificadas como relacionadas con la imagen original respecto al número total de imágenes analizadas.

$$R = \frac{\#correct\ matches}{\#correspondences}$$

4.2.2.4 Medida-F1 (F1)

Es un valor ponderado entre las dos métricas anteriores y será utilizada para determinar la precisión de los descriptores.

$$F1 = \frac{2PR}{P + R}$$

4.3 IMPLEMENTACIÓN DEL SISTEMA DE PRUEBAS

El primer paso para realizar las pruebas sobre los descriptores es la creación de una aplicación móvil que se ejecute en el sistema operativo Android, denominada "sistema de pruebas" la cual encapsula la implementación de los descriptores a evaluar: SIFT, SURF y ORB. La Figura 35 representa los componentes que posee la aplicación.

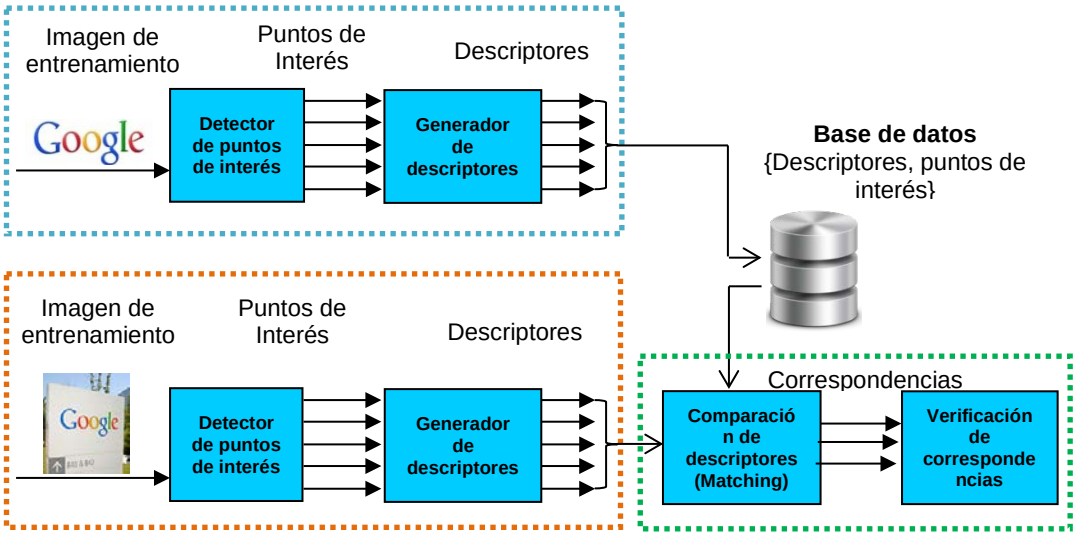

Figura 35: Componentes del sistema de pruebas

4.3.1 Descripción de los componentes del sistema de pruebas

- Base de datos prototipo: base de datos que almacena una tupla (secuencia ordena de objetos) compuesta por: un vector de características y un vector de puntos interés.
- Componente de entrenamiento: permite almacenar en una base de datos el vector de característica y los puntos de interés generados por unos de los descriptores (SIFT, SURF u ORB) a partir de una imagen de entrada.
- Componente de prueba: permite obtener el vector de característica y los puntos de interés generados por unos de los descriptores (SIFT, SURF u ORB) a partir de una imagen de entrada para su posterior comparación.

Tupla: Secuencia ordena de objetos

- Componente de búsqueda de correspondencias (matching): permite obtener un conjunto de correspondencias entre la tupla obtenida por el componente de prueba y una de las tuplas de la base de datos prototipo.

Finalmente para la implementación del sistema se utilizó las siguientes herramientas:

- Eclipse “Juno”
- Android SDK
- Android NDK y
- OpenCV4Android 2.4.4

Las herramientas fueron descritas en el capítulo 2 específicamente en la sección 2.9. Es importante mencionar que OpenCV en su versión 2.4.4 ya trae implementado los tres algoritmos estudiados en su versión para PC(s) más no en su versión para Android, donde únicamente tiene implementado el descriptor ORB. Por tal motivo, se creó una implementación de los descriptores SIFT y SURF basadas en la implementación para PC(s) de OpenCV.

4.3.2 Creación de la base de datos de imágenes

La base de imágenes para este proyecto está creada a partir de una galería de 200 imágenes (100 logotipos diferentes, es decir, dos imágenes por logotipo) de 640x480. El sistema de pruebas utiliza 100 logotipos diferentes para el entrenamiento y los 100 restantes para la fase de pruebas o comprobación.

Para validar el comportamiento de los descriptores frente a cambios que puede sufrir las imágenes en escala, rotación e iluminación, los 100 logotipos designados para la fase de comprobación se encuentran agrupadas de la siguiente manera:

- Grupo1 (G1): agrupa 25 logotipos, las cuales no tienen ninguna transformación. La Figura 36 ejemplifica lo mencionado.

Figura 36: Ejemplo de imágenes sin ninguna transformación.

- Grupo2 (G2): agrupa 25 logotipos, los cuales han sufrido diversos cambios de escala (ampliaciones o reducciones). La Figura 37 ejemplifica lo mencionado.

Figura 37: Ejemplo de imágenes afectadas por el cambio de escala

- Grupo3 (G3): agrupa 25 logotipos, las cuales ha sufrido cambios de rotación con diferentes grados (+/-15 SURF y ORB; +/- 30 SIFT). La Figura 38 ejemplifica lo mencionado.

Figura 38: Ejemplo de imágenes afectadas por el cambio de rotación.

- Grupo4 (G4): agrupa 25 logotipos, los cuales han sufrido cambios de iluminación (cambios de brillo y contraste). La Figura 39 ejemplifica lo mencionado.

Figura 39: Ejemplo de imágenes afectadas por el cambio de iluminación.

4.3.3 Ajustes de descriptores

El objetivo de esta sección es la selección de los valores óptimos para los parámetros de cada uno de los descriptores. Los valores seleccionados serán utilizados posteriormente por los descriptores en la etapa de evaluación. Es importante mencionar que este proceso de ajuste está asociado únicamente a la etapa de matching o comparación entre puntos claves, por lo que los parámetros (para la detección y extracción de características) de cada descriptor son tomados de los papers de los autores referenciados en el capítulo 2.

Para la optimización de los parámetros de cada descriptor se ha utilizado 100 logotipos que están divididos en 4 grupos: escala, rotación, iluminación e individuales.

4.3.3.1 Parámetro de ajuste

La etapa de matching o comparación para SIFT, SURF y ORB, consiste en el cálculo de la distancia basada en el vecino más próximo o cercano, la cual establece el matching entre aquellos puntos claves de ambas imágenes si, el ratio entre las distancias d_1 y d_2 es menor que un umbral→threshold .

$$\frac{d_1}{d_2} < threshold.$$

Paper: Artículo científico destinado a la publicación en revistas especializadas.
Threshold: Parámetro de umbral o radio para ajustar algoritmos de reconocimiento de imágenes.

En base a dicha definición, se puede deducir que el valor asignado al threshold condiciona el número de correspondencias encontradas entre puntos clave, por lo que, encontrar el valor óptimo de este umbral permitirá mejorar el desempeño en el reconocimiento de imágenes. Ya definido el parámetro que se ajustará se presenta a continuación un conjunto de cinco valores que este podrá tomar.

$$threshold = (0.6, 0.65, 0.7, 0.75, 0.8)$$

4.3.3.2 Selección de métricas

La etapa de ajuste se completa mediante el cálculo de las métricas de precisión y recall (PR) de los cuatro grupos de imágenes seleccionadas con los diferentes valores de threshold que puede tomar para la comparación. Al final se concluye con la selección del valor optimo del threshold para el descriptor analizado.

- **SIFT**

En capítulo 2 se explica el funcionamiento de los componentes del descriptor SIFT, de donde aparecen ciertos parámetros que ha sido establecido siguiendo las recomendaciones del paper de referencia (Guoshen, 2009). A continuación se resumen los parámetros y sus valores que serán utilizados para la evaluación del descriptor.

- $\sigma = 1.6$

  Factor de escala gaussiano.

- $s = 3$

  Representa el número de imágenes en cada octava.

- $|D| = 0.03$

  Representa el umbral relativo a la eliminación de puntos clave de bajo contraste en el proceso de localización de puntos claves.

- $r = 10$

  Representa el radio relativo a la eliminación de puntos clave en los bordes dentro del proceso de localización de puntos clave.

- $\frac{d_1}{d_2} < \ threshold$

  Radio relativo a la etapa de matching entre puntos clave.

Para obtener el valor de radio óptimo, se realizara el cálculo de correspondencias con los cuatro grupos de imágenes; además se utilizara los diferentes valores de **threshold**.

La tabla 6 muestra los resultados obtenidos por la etapa de ajuste para el descriptor SIFT.

En base a los resultados obtenidos se puede concluir que el valor óptimo de **threshold** es de 0.75, ya que la precisión de las pruebas de correspondencias con este valor fue el más alto con $\mathbf{F1 = 0.33}$.

Tabla 6: Resumen de valores P, R y F1 generados por la etapa de ajuste para el descriptor SIFT

Grupos	0.6			0.65			0.7			0.75			0.8		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Individual	0.81	0.46	0.58	0.80	0.47	0.59	0.80	0.50	0.62	0.82	0.50	0.62	0.77	0.50	0.60
Escala	0.36	0.22	0.27	0.35	0.22	0.27	0.35	0.22	0.27	0.35	0.23	0.28	0.33	0.27	0.30
Rotación	0.16	0.07	0.10	0.16	0.09	0.12	0.15	0.08	0.10	0.20	0.10	0.13	0.17	0.08	0.11
Iluminación	0.49	0.15	0.23	0.49	0.15	0.23	0.49	0.16	0.24	0.52	0.18	0.27	0.50	0.18	0.26
Media	0.45	0.22	**0.30**	0.45	0.23	**0.30**	0.45	0.24	**0.31**	0.47	0.25	**0.33**	0.44	0.25	**0.32**

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

- **SURF**

El descriptor SURF descrito en el capítulo 2 al igual que SIFT define algunos parámetros por defecto para su ajuste, estos parámetros son tomados del paper del autor (Lowe, 1999).

- $nOctaves = 4$

  Identifica el número de octavas que se utiliza en las búsquedas de puntos clave.

- $nOctaveLayers = 3$

  Representa el número de imágenes (capas) por octava.

- $extended = 0$

  Representa la longitud por defecto del vector de características de 64 elementos; en caso de establecer un valor de 1, la dimensión del vector será de 128.

- $\frac{d_1}{d_2} < \ threshold$

  Valor a encontrar.

Los resultados obtenidos de las pruebas son recogidos en la tabla 7

Tabla 7: Resumen de valores P, R y F1 generados por la etapa de ajuste para el descriptor SURF

Grupos	0.6			0.65			0.7			0.75			0.8		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Individual	0.89	0.39	0.54	0.89	0.40	0.56	0.89	0.50	0.64	0.87	0.48	0.62	0.85	0.52	0.65
Escala	0.71	0.33	0.45	0.71	0.35	0.47	0.71	0.37	0.49	0.71	0.39	0.50	0.70	0.40	0.51
Rotación	0.29	0.11	0.16	0.29	0.11	0.16	0.31	0.12	0.17	0.29	0.12	0.17	0.27	0.13	0.17
Iluminación	0.51	0.13	0.20	0.62	0.14	0.22	0.64	0.15	0.25	0.64	0.17	0.27	0.61	0.19	0.29
Media	0.60	0.24	**0.34**	0.63	0.25	**0.35**	0.64	0.28	**0.39**	0.63	0.29	0.39	**0.61**	0.31	**0.40**

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

A partir de los resultados obtenidos se concluye que el $threshold$ óptimo es de 0.8, por cuanto la precisión que alcanzan las pruebas con este umbral es de $F1 = 0.40$, siendo este el valor más alto de todos los calculados.

- **ORB**

Para el descriptor ORB se definen los siguientes parámetros por defecto obtenidos del paper de referencia[46], teniendo como valores:

- $nFeatures = 500$

  Representa el número máximo de puntos obtenidos por del detector.

- $scaleFactor = 1.2$

  Factor de escala utilizado para generar el *espacio de escala.*

- $\frac{d_1}{d_2} < \ threshold$

  Valor a encontrar.

En la tabla 8 se muestra los valores obtenidos por la etapa de ajuste para el descriptor ORB.

Tabla 8: Resumen de valores P, R y F1 generados por la etapa de ajuste para el descriptor ORB

Grupos	0.6			0.65			0.7			0.75			0.8		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Individual	0.89	0.46	0.60	0.90	0.47	0.61	0.88	0.48	0.62	0.86	0.49	0.63	0.88	0.50	0.64
Escala	0.79	0.20	0.32	0.79	0.23	0.36	0.78	0.27	0.40	0.75	0.32	0.45	0.75	0.35	0.48
Rotación	0.33	0.11	0.17	0.32	0.13	0.18	0.32	0.14	0.20	0.33	0.17	0.23	0.31	0.17	0.22
Iluminación	0.59	0.11	0.18	0.60	0.12	0.20	0.66	0.13	0.22	0.67	0.16	0.26	0.66	0.19	0.29
Media	0.65	0.22	**0.32**	0.65	0.24	**0.34**	0.66	0.26	0.36	0.65	0.29	**0.39**	0.65	0.30	**0.41**

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

Finalmente se concluye que el valor óptimo del $threshold$ para el descriptor ORB es de 0.8 el mismo valor que se obtuvo para el descriptor SURF. La precisión de las pruebas realizas con este valor de $threshold$ alcanzaron un valor de $F1 = 0.41$.

4.3.4 Comparación de descriptores

Una vez encontrados los valores óptimos de $threshold$ para cada uno de los descriptores, el siguiente paso es realizar una evaluación comparativa entre los descriptores para determinar cuál de ellos proporciona mejores resultados tanto en desempeño como en consumo de recursos.

Como se indicó en la sección de “Criterios e indicadores de evaluación”, las métricas utilizadas para la comparación se dividen en dos grupos: métricas de eficiencia (miden el consumo de recursos) y métricas de eficacia (miden el desempeño del descriptor).

Para obtener las métricas de eficiencia se ha utilizado el conjunto de imágenes de 100 logotipos destinados al entrenamiento del sistema de pruebas y para el cálculo de las métricas de eficacia se ha utilizado el conjunto de imágenes de 100 logotipos divididos en cuatro grupos diferentes, lo que permitirá evaluar el comportamiento de los descriptores en diferentes escenarios.

4.3.4.1 Consumo de recursos - eficiencia

La evaluación del consumo de recursos por parte de los descriptores se divide en diferentes apartados que tiene que ver con las métricas de eficiencia: tiempo de ejecución, peso utilizado por los vectores de características, porcentaje de batería y el número promedio de Puntos de interés extraídos.

Como se indica al inicio de esta sección, para la ejecución de las pruebas se utilizan los 100 logotipos destinados al entrenamiento los cuales serán ejecutados 10 veces, con el objetivo de observar si los algoritmos cambian su comportamiento según el estado del teléfono.

La tabla 9, muestra los resultados experimentales del descriptor SIFT, mientras que la tabla 10 muestra los resultados experimentales del descriptor SURF y por último la tabla 11 muestra los resultados experimentales obtenidos para el descriptor ORB.

Tabla 9: Pruebas de eficiencia con el descriptor SIFT

Métricas	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7	Prueba 8	Prueba 9	Prueba 10	Media
# Puntos de interes	222	222	222	222	222	222	222	222	222	222	**222**
Tiempo de ejecución (ms)	2908.10	2926.99	3049.34	3125.93	3014.20	3010.96	3090.49	2926.34	3036.73	2902.46	**2999.15**
Peso descriptor (KB)	102.30	102.30	102.30	102.30	102.30	102.30	102.30	102.30	102.30	102.30	**102.30**
Consumo de batería (%)	0.03	0.03	0.03	0.04	0.05	0.03	0.05	0.04	0.04	0.05	**0.04**

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

Tabla 10: Pruebas de eficiencia con el descriptor SURF

étricas	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7	Prueba 8	Prueba 9	Prueba 10	Media
# Puntos de interes	431	431	431	431	431	431	431	431	431	431	**431**
Tiempo de ejecución (ms)	991.04	936.91	971.80	892.09	914.25	929.20	985.17	907.05	974.24	900.87	**940.26**
Peso descriptor (KB)	385.02	385.02	385.02	385.02	385.02	385.02	385.02	385.02	385.02	385.02	**385.02**
Consumo de batería (%)	0.02	0.03	0.02	0.02	0.01	0.02	0.02	0.01	0.01	0.01	**0.02**

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

Tabla 11 Pruebas de eficiencia con el descriptor ORB

Métricas	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5	Prueba 6	Prueba 7	Prueba 8	Prueba 9	Prueba 10	Media
# Puntos de interes	432	432	432	432	432	432	432	432	432	432	432
Tiempo de ejecución (ms)	159.62	155.63	159.57	160.52	155.26	155.01	161.07	153.02	152.65	157.50	156.98
Peso descriptor (KB)	49.66	49.66	49.66	49.66	49.66	49.66	49.66	49.66	49.66	49.66	49.66
Consumo de batería (%)	0	0	0.01	0	0.01	0.01	0	0.01	0	0.01	0.01

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

Como conclusión se presenta la tabla 12, que agrupa los resultados obtenidos por cada descriptor mediantes las medias ponderadas.

Tabla 12: Comparación de eficiencia entre descriptores

Descriptor	# Puntos de interés	Tiempo de ejecución (ms)	Peso descriptor (KB)	Consumo de batería (%)
SIFT	222	2999.15	102.30	0.04
SURF	431	940.26	385.02	0.02
ORB	432	156.98	49.66	0.01

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

Con los resultados obtenidos se concluye que ningún algoritmo cambio su estado al ser probado 10 veces, más bien se afirma que el comportamiento de los descriptores es constante y efectivamente cumple con la propiedad de repitibilidad.

A continuación se detallan los resultados de las métricas de eficiencia analizadas:

- **Número de Puntos de interés**

En la figura 40 se resumen el número promedio de Puntos de interés detectados por descriptor en una imagen de entrada. Se observa que el descriptor ORB y el descriptor SURF son los que más Puntos de interés detectan 432 y 431 respectivamente, en cambio el descriptor SURF se encuentra rezagado detectando apenas 222 puntos claves.

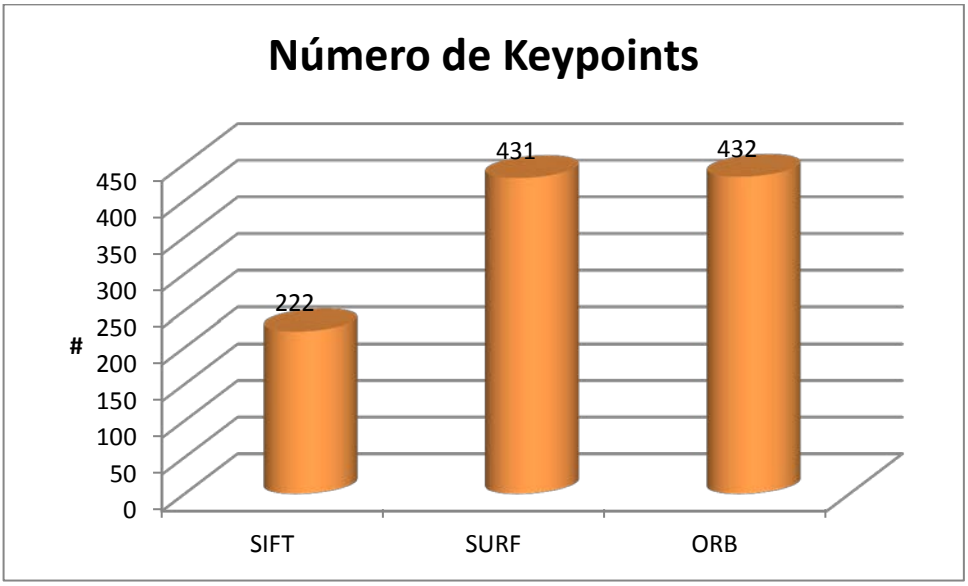

Figura 40 Comparación del número de Puntos de interés (K_t) detectados entre los descriptores SIFT, SURF y ORB

- **Tiempo de ejecución**

En la figura 41 se observa que el tiempo de computo empleado por el descriptor SIFT es de 3.0s superando así ampliamente a los tiempos empleados por el descriptor SURF y ORB con 2.06 (s) y 2.84 (s) respectivamente. Esto nos conduce a la conclusión, que el descriptor SIFT no es una buena opción para aplicaciones en tiempo real.

Todo lo contrario se afirma sobre el descriptor ORB el cual necesita un tiempo promedio de 0.16 (s) para procesar una imagen, convirtiéndolo en la mejor opción de entre los tres descriptores para aplicaciones de tiempo real. Si bien el tiempo promedio de ejecución del descriptor SURF es de 0.94 (s), sigue siendo lento si se lo compara con el descriptor ORB.

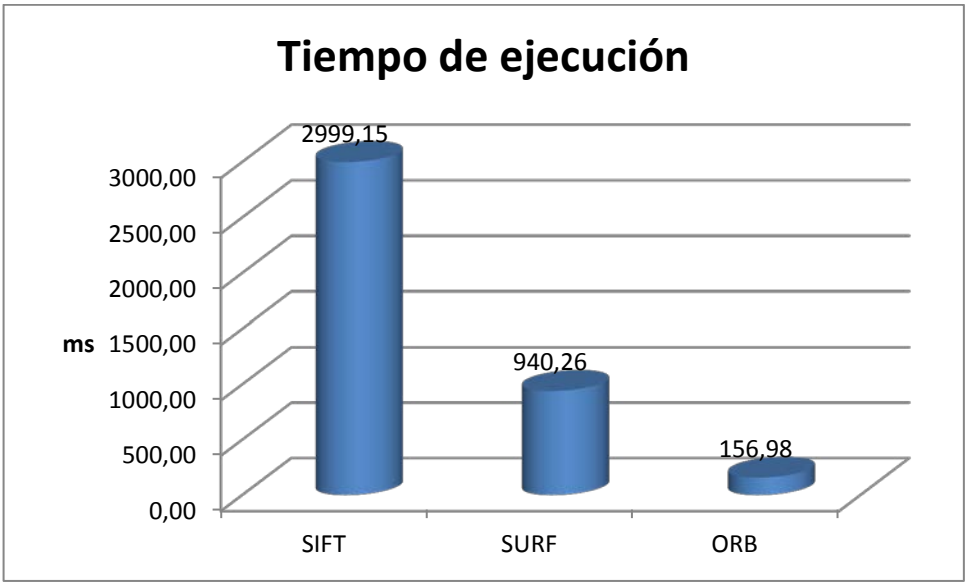

Figura 41 Comparación de tiempo de ejecución ($\boldsymbol{T_{total}}$) entre los descriptores SIFT, SURF y ORB

- **Consumo de batería**

En lo que se refiere al consumo de batería la figura 42 muestra los niveles consumidos por cada descriptor. Observándose que el porcentaje de batería utilizado para procesar una imagen es proporcional al tiempo de ejecución analizado en el apartado anterior.

Así, se obtuvo que el porcentaje de batería consumida por ORB es de apenas el 0,01% del total disponible, en cambio SURF consume el doble (0.02%) y finalmente SIFT consume cuatro veces más con (0.04%).

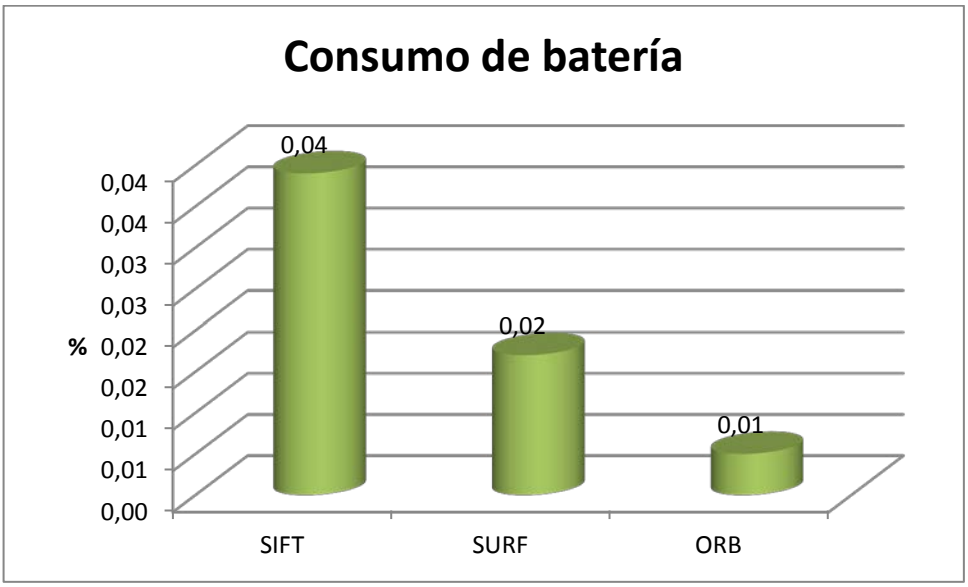

Figura 42 Comparación del (%) de batería consumido ($\boldsymbol{B_t}$) por los descriptores SIFT, SURF y ORB

- **Peso del descriptor**

Con respecto al peso que necesitan los descriptores para almacenar sus vectores de características, la figura 43 muestra que tanto SIFT como ORB son los que

menos peso tiene con 102.30KB y 49.66KB respectivamente. En cambio el vector de características generado por el descriptor SURF necesita un promedio de 385.02KB casi 8 veces el peso del vector generado por ORB.

Esto es consecuencia del tipo de dato almacenado en el vector de características, es así que SURF y SIFT almacenan datos de tipo double, mientras que ORB almacena datos tipo char.

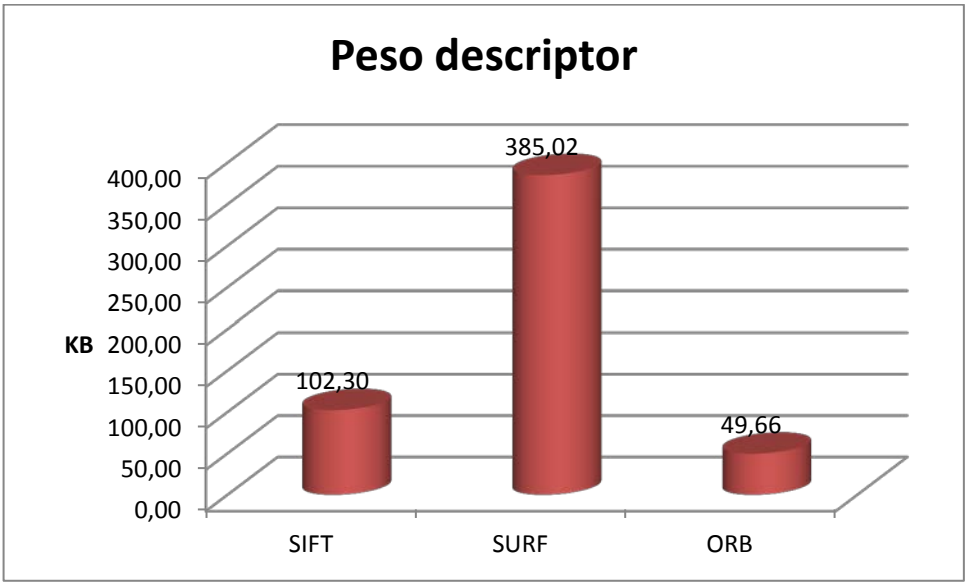

Figura 43: Comparación del peso/tamaño (P_t) que necesitan los descriptores SIFT, SURF y ORB para almacenar sus vectores de características

4.3.4.2 Consumo de recursos – Desempeño

Para evaluar el desempeño de cada uno de los descriptores se ha utilizado la métrica F1 descrita en la sección 3. Las pruebas se ejecutaron con una base de 100 imágenes de logotipos, divididas en cuatro grupos con las siguientes

transformaciones: escala, iluminación, rotación e individuales. En Tabla 13 agrupa los resultados de las pruebas por cada descriptor.

Tabla 13 Comparación de desempeño entre descriptores

Grupos	SIFT			SURF			ORB		
	P	R	F1	P	R	F1	P	R	F1
Individual	0.82	0.50	0.62	0.85	0.52	0.65	0.88	0.50	0.64
Escala	0.35	0.23	0.28	0.70	0.40	0.51	0.75	0.35	0.48
Rotación	0.20	0.10	0.13	0.27	0.13	0.17	0.31	0.17	0.22
Iluminación	0.52	0.18	0.27	0.61	0.19	0.29	0.66	0.19	0.29
Media	0.47	0.25	**0.33**	0.61	0.31	**0.40**	0.65	0.30	**0.41**

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

- **Individuales**

Las imágenes pertenecientes al grupo de logotipos que no sufrieron ninguna transformación, no deberían representar ninguna dificultad para los descriptores debido a que son casi identificas a las imágenes originales o de entrenamiento.

La figura 44 muestra los resultados experimentales de grupo de imágenes individuales, donde el descriptor SURF es el que mejor desempeño muestra ante este tipo de imágenes con un 65% de precisión seguido muy de cerca por el descriptor ORB con una precisión del 64% y finalmente el descriptor SIFT con una 62%.

La variación más grande entre los descriptores (SURF vs SIFT) es de apenas tres puntos porcentuales, lo que lleva a concluir que no existe una diferencia bien marcada entre los descriptores comparados.

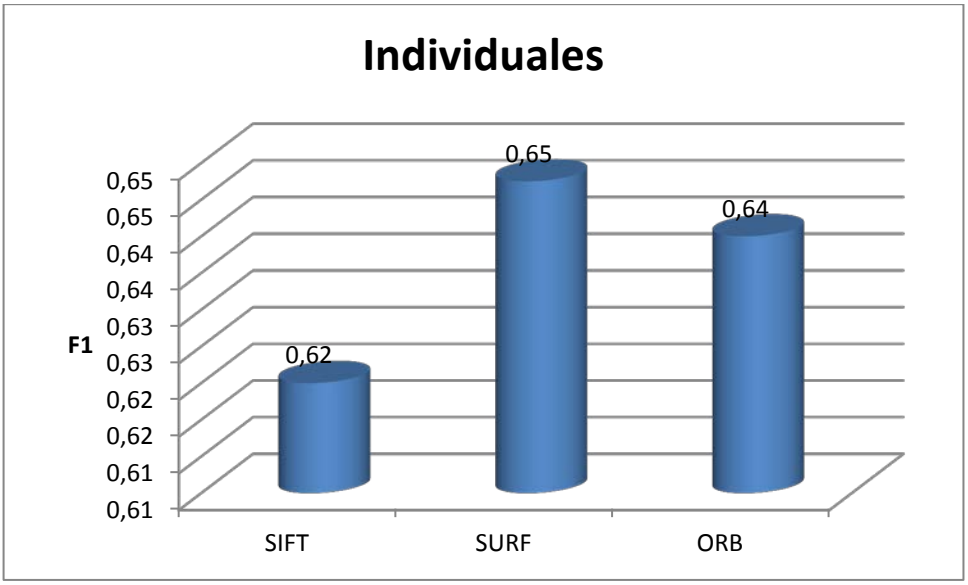

Figura 44 Comparación de descriptores con imágenes sin ninguna transformación

- **Cambios de escala**

Lo resultados obtenidos para la detección de imágenes que han sufrido una variación de escala respecto de la imagen original se las recoge en la figura 45. Al igual que con el grupo de imágenes individuales, los descriptores SURF y ORB son los que mayor precisión presentan con 51% y 48% respectivamente. El descriptor SIFT presenta una precisión de 28%, es decir 23% menos que SURF y 20% menos que ORB. Lo que nos indica que SIFT no tiene un buen desempeño con imágenes de diferentes escalas.

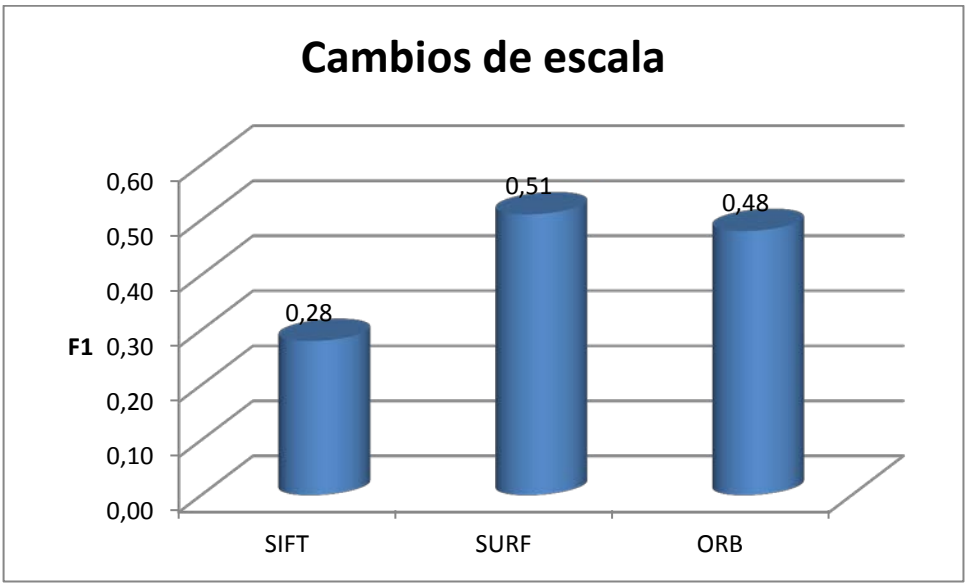

Figura 45 Comparación de descriptores con imágenes que ha sufrido cambios de escalas

- **Cambios de iluminación**

La figura 46 recoge los resultados de la comparación de los descriptores con imágenes que han sufrido cambios de brillo y contraste. Estas transformaciones afectan directamente a los pixeles de las imágenes, cambiándolos de color y en algunos casos modificando la morfología de las imágenes.

Los resultados muestran que tanto SURF y ORB tienen la misma precisión 29%, en cambio SIFT se encuentra con un 2% menos. Se puede apreciar que los resultados obtenidos en este grupo siguen la tendencia de los grupos anteriores, con SURF y ORB con las precisiones más altas pero sin una diferencia marcada entre ellos.

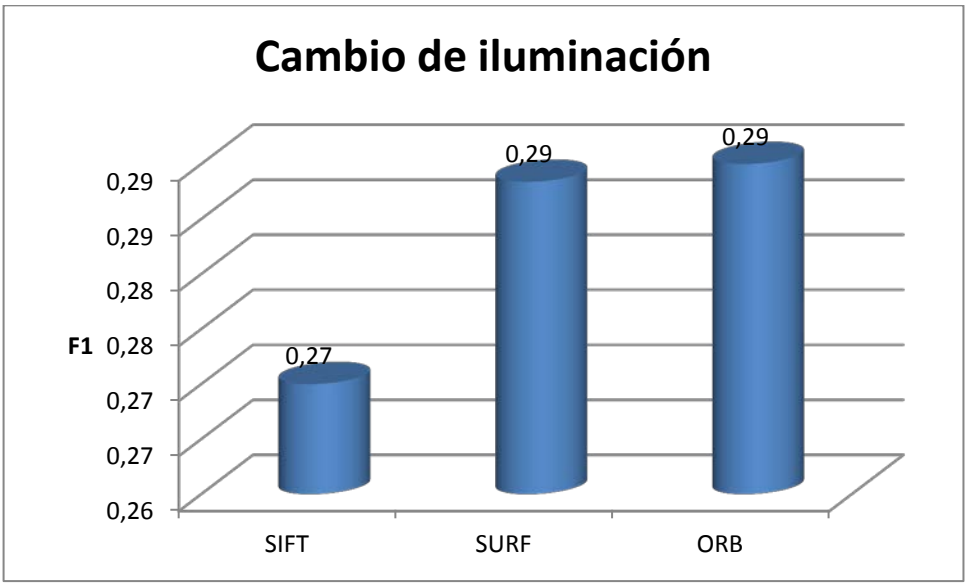

Figura 46 Comparación de descriptores con imágenes que ha sufrido cambios de iluminación

- **Cambios de rotación**

Este tipo de transformación se caracteriza por el cambio de ángulo de la imagen de pruebas en relación a la imagen original, lo que causa la aparición o desaparición de ciertas regiones. Los tres descriptores comparados cuentan con algún mecanismo para solventar este problema.

Los resultados recogidos de esta prueba se resumen en la Figura 47, en esta se aprecia de forma clara, que la precisión alcanzada por el descriptor ORB (28%), supera a SURF con 5% y a SIFT con un 9%.

A diferencia de los 3 grupos anteriores donde la variación de la precisión entre SURF y ORB no supera el 3%, en esta se ve un diferencia de 5% de ORB sobre SURF.

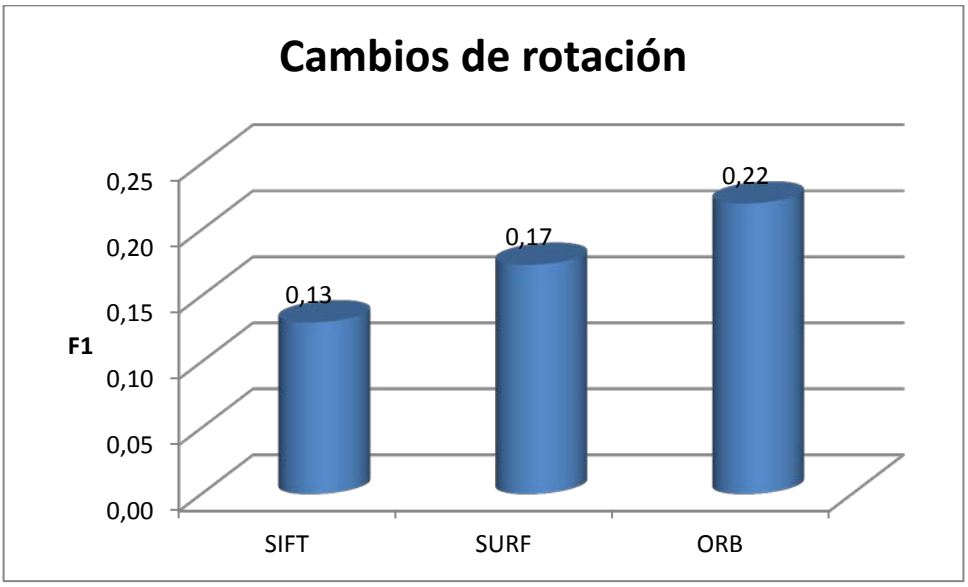

Figura 47 Comparación de descriptores con imágenes que ha sufrido cambios de rotación

4.4 RESULTADOS OBTENIDOS

En base al análisis de cada una de las pruebas comparativas realizadas en relación al uso eficiente de recursos, se determina que, el descriptor ORB es el descriptor más eficiente al necesitar únicamente 0.16s para detectar 432 puntos claves de una imagen de entrada y obtener el vector de características de la misma con un peso promedio de 49.66 KB, consumiendo un 0.01% de batería disponible por el teléfono.

En cuanto al desempeño se refiere se plantean las siguientes conclusiones:

El descriptor SURF tiene el mejor desempeño en imágenes que no han sufrido transformaciones y en imágenes con cambios de escala, alcanzando un 65% y 57% respectivamente en cada grupo.

Si bien SURF tiene la precisión más alta en los grupos de imágenes: individuales y con cambios de escala, la diferencia con la precisión obtenida por ORB para los mismos grupos no es abismal, apenas del 1% para el grupo de imágenes individuales y 3% para el grupo de cambio de escala.

El descriptor SIFT en los cuatro grupos de imágenes analizados obtiene las precisiones más bajas. Imágenes individuales 62%, cambios de escala 28%, cambios de iluminación 27% y cambios de rotación 13%.

Al calcular el promedio entre los resultados de la métrica F1 (tabla 13) de los grupos de imágenes analizados, se obtiene que el descriptor ORB posee una precisión promedio del 41% siendo esta la más alta entre los descriptores evaluados. En la Figura 48 se muestra el promedio de las precisiones calculadas para cada descriptor.

Se comprobó que ninguno de los tres descriptores analizados tiene un buen desempeño con logotipos, puesto que este tipo de imagen no ofrece una alta cantidad de características como si lo hacen las imágenes complejas. Esta conclusión se asemeja a la expuesta en la investigación denominada reconocimiento e

identificación de logotipos en imágenes con trasformada SIFT (Juárez, 2011) en la cual se hace un análisis del descriptor SIFT en un ambiente desktop con el mismo tipo de imágenes utilizados en este análisis

Como conclusión final se determina que el descriptor ORB es el más eficiente y eficaz entre los descriptores evaluados sobre el sistema operativo Android, para el reconocimiento de imágenes (específicamente logotipos), como lo muestra la figura 48.

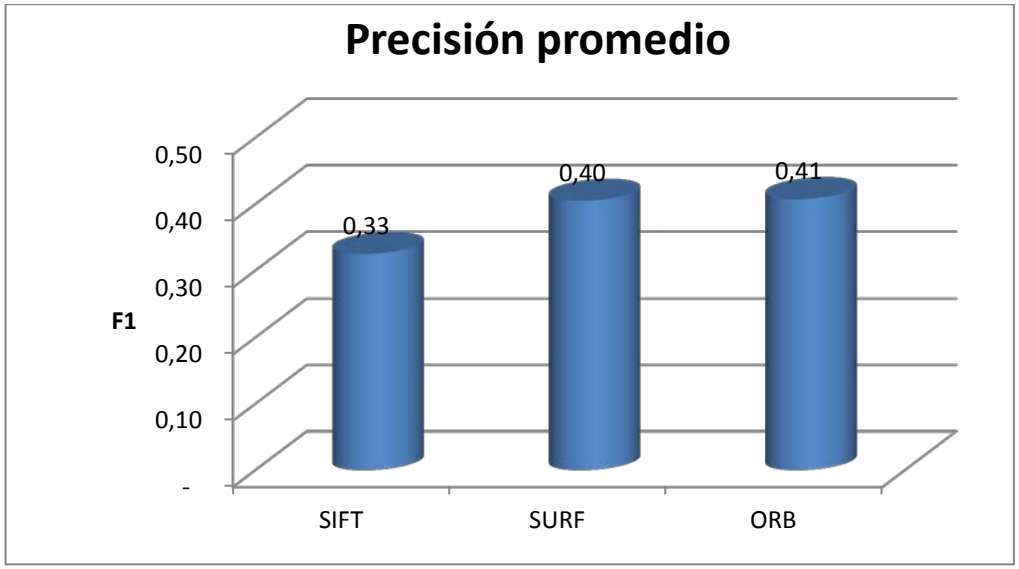

Figura 48 Comparación entre descriptores

CAPÍTULO 5

DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO

5.1 INTRODUCCIÓN

Este capítulo abarca el desarrollo del aplicativo denominado “LogoFinder”, capaz de reconocer logotipos a partir de una imagen dada mediante la utilización del algoritmo ORB y mostrar información relacionada a la misma.

Al tratarse de un prototipo, resultado de una investigación, los requerimientos para la construcción del sistema han sido definidos por los autores de este proyecto.

5.1.1 Requerimientos funcionales de LogoFinder

- Una aplicación móvil que permita buscar logotipos en base a una imagen tomada desde la cámara de un teléfono inteligente. Si el sistema encuentra alguna coincidencia debe mostrar la siguiente información:

    - Imagen que se consulta
    - Nombre del logotipo
    - Descripción
    - URL de referencia.

- Una página web que permita agregar, editar, eliminar y consultar logotipos.
- Además, de la información del logotipo que se muestra al usuario se debe guardar los datos necesario para comparar las imágenes de los mismos.
- Un logotipo puede tener asignado varias imágenes para su entrenamiento
- Para el reconocimiento de los logotipos el sistema utiliza el descriptor ORB el cual fue seleccionado entre los descriptores estudiados en este proyecto.
- La base de datos donde se almacenan los logotipos se encuentra en un servidor remoto.

5.2 MODELO DE DESARROLLO

El objetivo principal de la ingeniería de software es construir una solución software eficiente que satisfaga las necesidades requeridas por el cliente. Para cumplir con este objetivo existen diferentes enfoques o paradigmas de desarrollo, tales como el modelo en cascada, modelo por prototipos, modelo de la espiral entre otros. El modelo que mejor se ajusta a las necesidades del proyecto es el modelo por prototipos, por ende este modelo será utilizado para la construcción de todos los componentes del prototipo en cuestión.

Modelo por prototipos

Los prototipos nacieron como un método para acelerar la definición de los requisitos de software por construir. La idea es crear un modelo de toda la aplicación o de algunas de sus partes y presentársela al cliente. El cliente hará sus observaciones

acerca de lo que ve en el modelo y el desarrollador en base a estas observaciones modificara ese modelo. El proceso se repite hasta cubrir con todos los requerimientos del producto. La figura 49 detalla el ciclo seguido por el modelo de prototipos.

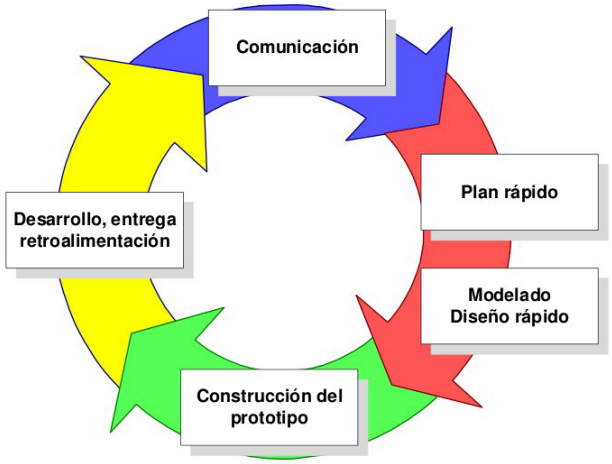

Figura 49: Paradigma de la construcción de prototipos

A continuación se detallan las etapas utilizadas en el desarrollo del aplicativo:

5.2.1 Plan rápido

Para que el desarrollo del sistema concluya con éxito, es de vital importancia que antes de codificar se tenga una completa y plena compresión de los requisitos del software.

En el anexo 1 se adjunta el documento de especificación de casos de uso y los diagramas de clases que fueron generados producto del análisis realizado.

5.2.2 Diseño

Basados en los requerimientos que debe cumplir el sistema propuesto y aplicando la estrategia de diseño “dividir para vencer”. Se define una arquitectura de n-capas como se detalla en la figura 50.

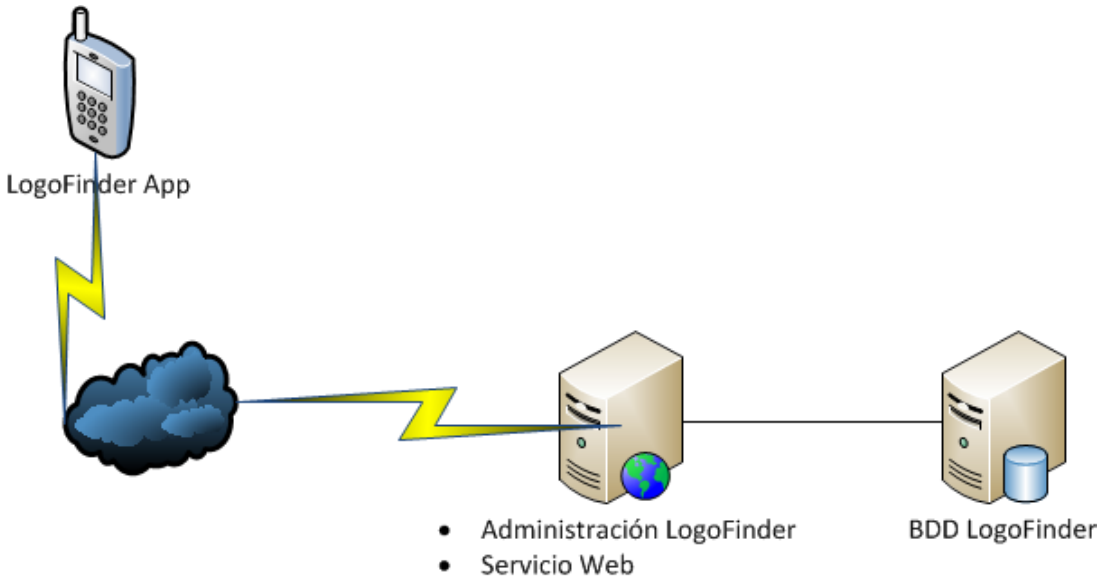

Figura 50: Diagrama de arquitectura de LogoFinder

Los componentes de la arquitectura se detallan a continuación:

- ***LogoFinder App*:** Comprende el aplicativo móvil sobre el sistema operativo Android.
- ***Administración de LogoFinder*:** es el sitio web que permite la admiración de los logotipos (Entrenamiento de logotipos)

- ***Servicio Web*:** permite realizar consultas sobre logotipos que fueron registrados en la base de datos.
- ***Base de datos (BDD) LogoFinder*:** medio de almacenamiento de logotipos

Como se puede apreciar el desarrollo se divide en dos grupos bien diferenciados:

- **Back-end:** Comprende las partes que se encuentra en un servidor remoto.
- **Front-end:** Comprende el aplicativo móvil

5.2.2.1 Diseño de Back-End

- Diseño de datos

El diseño de datos se realiza en base a la información que se pretende mostrar al usuario luego de que este realice una consulta desde un teléfono inteligente hacia el sistema en busca de algún logotipo. La figura 51 muestra el modelo entidad relación de la base de datos del sistema.

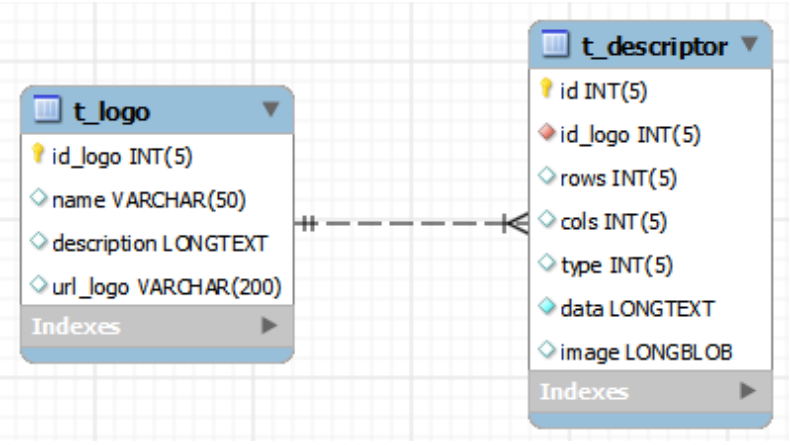

Figura 51: Modelo entidad relación de la base de datos LogoFinder

- Diseño del administrador de logotipos

La arquitectura utilizada para el desarrollo del administrador es de 3-capas como se muestra en la figura 52.

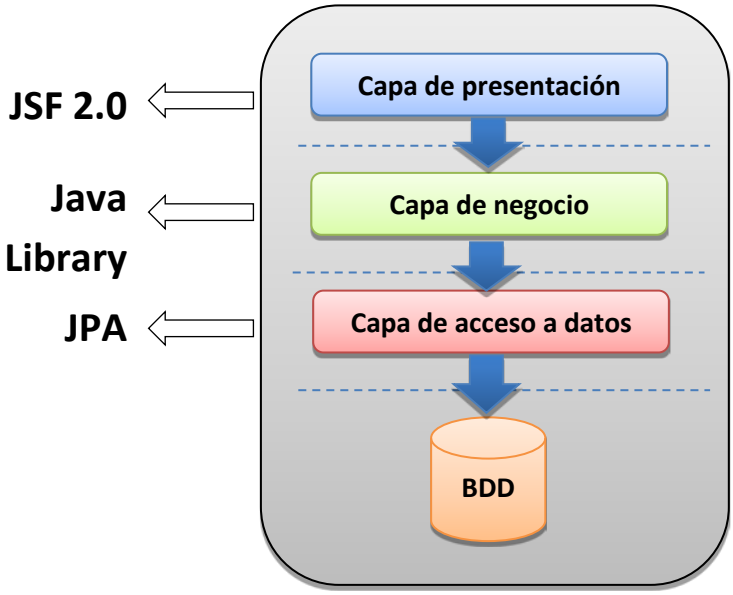

Figura 52: Arquitectura del administrado de logotipos

A continuación se presenta una breve descripción de cada una de las capas presentadas en la gráfica anterior, por medio de la tabla 14:

Tabla 14 Capas del Back/End

Capa de presentación	Esta capa es responsable de mostrar información al usuario e interpretar sus acciones. Se utiliza la tecnología JavaServer Faces (JSF) para este fin.
Capa de negocio	Representa la lógica misma que permite realizar las distintas operaciones de negocio requeridas por el usuario. Contendrá diferentes módulos funcionales y otras clases reusables en todos los módulos del componente. En esta capa se realiza el entrenamiento de las imágenes de los logotipos que serán almacenadas en la base de datos para futuras consultas.
Capa de acceso a datos	Representa la lógica para acceder al motor de base de datos. Para el acceso a datos utilizaremos Java Persistence API (JPA) que es el api de persistencia desarrollado para la plataforma JAVA EE.

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

- Diseño de la interfaz de usuario

En esta sección se detalla el diseño de la interfaz de usuario en la etapa de Back–End:

➢ Pantalla principal:

Figura 53: Pantalla principal del administrado de logotipos

- Agregar Logotipo:

Figura 54: Pantalla para agregar un logotipo en administrado de logotipos

- Editar Logotipo:

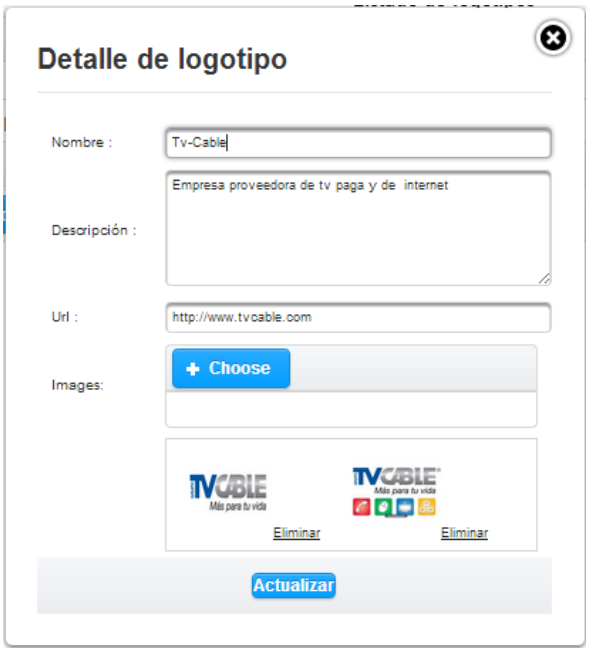

Figura 55: Pantalla para editar un logotipo en el administrado de logotipos

- Diseño del servicios web

Al tener una arquitectura divida en Back-end y Front-end surge la necesidad de tener una interfaz que permita la comunicación entre las dos partes. Para esto se hace uso de la tecnología de “servicios web” la cual permite el intercambio de datos entre aplicaciones.

El front-end, que se encuentra representado por la aplicación móvil realizará consultas al back-end por medio del servicio web, para determinar si la imagen que el usuario capturó con la cámara de su smartphone pertenece a un logotipo que se encuentra en la base de datos. Si se encuentran coincidencias el servicio web retornara la información del logotipo encontrado. La figura 56 se muestra la arquitectura que tendrá el servicio web.

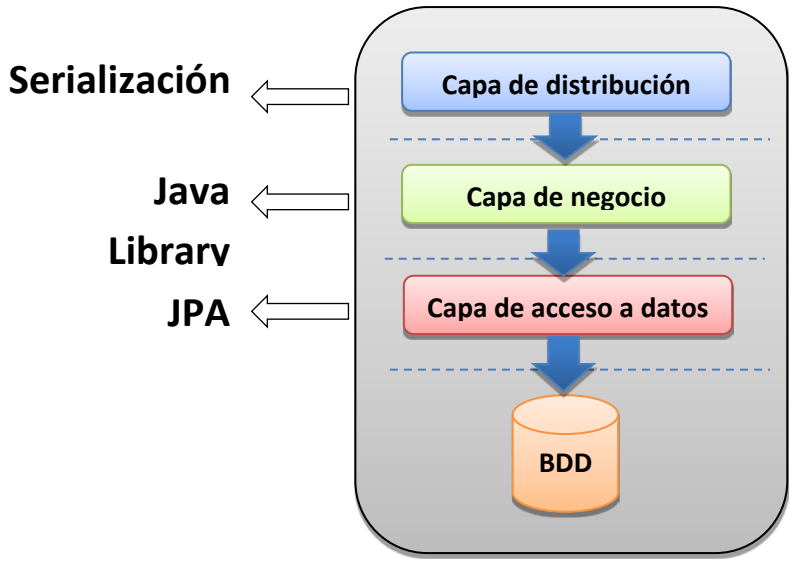

Figura 56: Arquitectura Servicio Web

A continuación se presenta una breve descripción cada una de las capas presentadas en la gráfica anterior, mediante la tabla 15:

Tabla 15 Capas del Servicio Web

Capa de presentación	Representa los servicios web que el componente publica para que sean invocados por consumidores externos. El formato para el intercambio de datos es JSON.
Capa de negocio	Representa la lógica misma que permite realizar las distintas operaciones de negocio requeridas por el usuario. Contendrá diferentes módulos funcionales y otras clases reusables en todos los módulos del componente. En esta capa se realiza el entrenamiento de las imágenes de los logotipos que serán almacenadas en la base de datos para futuras consultas.
Capa de acceso a datos	Representa la lógica para acceder al motor de base de datos. Para el acceso a datos utilizaremos Java Persistence API (JPA) que es el api de persistencia desarrollado para la plataforma JAVA EE.

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

5.2.2.2 Diseño de Front-End

El aplicativo móvil al igual que los otros subsistemas, está organizado utilizando el patrón de arquitectura en capas como se aprecia en la figura 57.

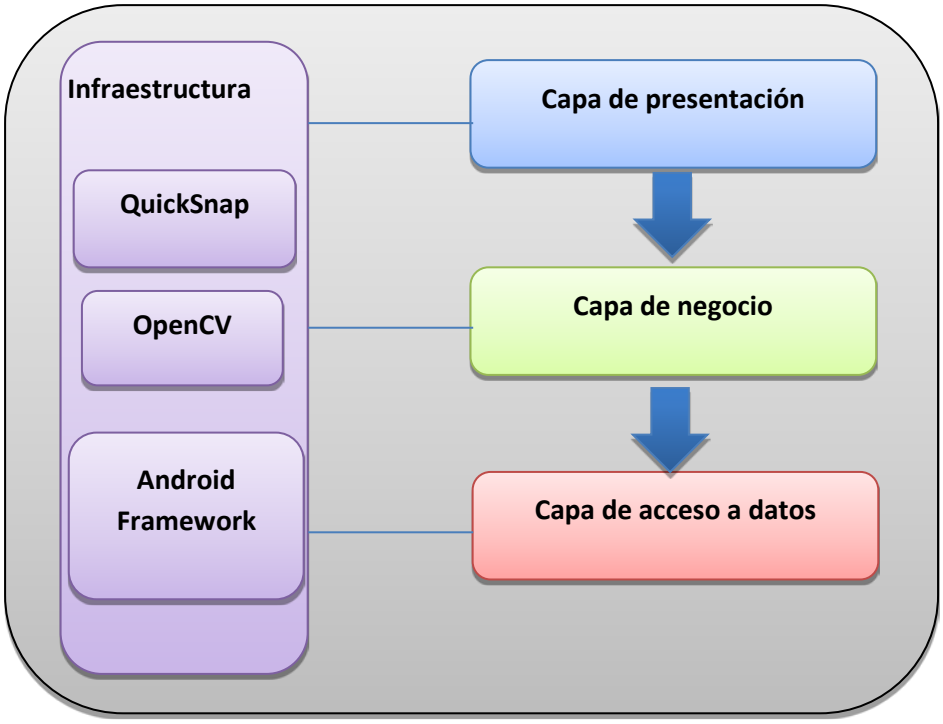

Figura 57: Arquitectura LogoFinder App

A continuación se presenta una breve descripción de cada una de las capas presentadas, mediante la tabla 16:

Tabla 16 Capas del Front/End

Capa de presentación	Implementa la lógica requerida para que los usuarios puedan interactuar con el sistema. Es responsable de mostrar información al usuario e interpretar sus acciones. Android utiliza archivos xml para la GUI.
Capa de negocio	Representa la lógica misma que permite realizar las distintas operaciones de negocio requeridas por los usuarios.
Capa de acceso a datos	Representa la lógica para acceder a los servicios provistos por sistemas externos a través de un cliente HTTP.

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

- **Infraestructura**

Esta capa proporciona capacidades técnicas genéricas que dan soporte al resto de capas. Estas capacidades son provistas en forma de servicios de infraestructura o librerías.

- QuickSnap

  Permite tener la cámara del dispositivo

- OpenCV4Android

  Librería OpenCV para dispositivos Android.

- Android framework

  Marco de trabajo para aplicaciones sobre el sistema operativo Android provisto por Google.

- **Diseño de interfaces**

En esta sección se detalla el diseño de la interfaz de usuario en la etapa de Front–End:

- Cámara

La figura 58, ilustra la interfaz gráfica de la cámara al capturar el logo a analizar

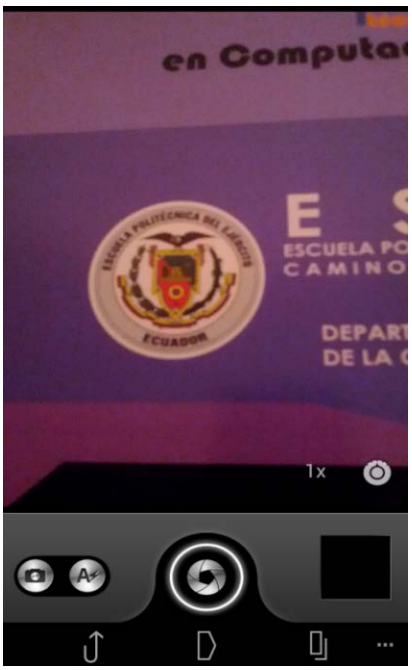

Figura 58: Interfaz de la cámara

- Búsqueda de logotipos

La figura 59, ilustra la interfaz gráfica de la aplicación al consultar un logo.

Figura 59: Interfaz de la aplicación al realizar una búsqueda

5.2.3 Construcción del prototipo

5.2.3.1 Construcción del Back-End

Luego de las etapas de análisis y diseño es el momento de construir el prototipo, para esto se utilizan las siguientes herramientas:

- Java como lenguaje de programación
- Netbeans7.3 como entorno de desarrollo
- Mysql como sistema de gestión de base de datos relacional
- Apache Tomcat 7.0.34 como servidor Web.

5.2.3.2 Construcción del Front-End

Para la construcción del aplicativo móvil se utilizan las siguientes herramientas:

- Android SDK
- Android NDK y
- Eclipse Juno como entorno de desarrollo

5.2.4 Pruebas del prototipo

Existen diferentes enfoques para la ejecución de pruebas de un sistema de software, las cuales se aplican dependiendo de lo que se desee verificar. Así, tenemos:

- Pruebas unitarias
- Pruebas de integración
- Pruebas del sistema
- Pruebas de aceptación
- Pruebas de instalación

Para este proyecto en particular, donde el objetivo de construir el prototipo es únicamente para demostrar el funcionamiento de algoritmos de reconocimiento de imágenes en dispositivos móviles, se ha decido utilizar “las pruebas del sistema” las cuales son detallas a continuación.

5.2.4.1 Pruebas del sistema

El objetivo de esta etapa es probar al sistema como un todo. Se basa en los requerimientos generales y abarcan todas las partes combinadas del sistema.

La tabla 17 resume las pruebas ejecutadas al prototipo según los requerimientos del mismo y los resultados obtenidos.

Tabla 17: Pruebas del sistema " LogoFinder"

ID	Requerimiento	Pre-condición (dado que)	Resultado esperado	Paso	Fallo
1	Administración de logotipos	Registrar el logotipo: ▪ Nombre: ESPE ▪ Detalle: Escuela politécnica del Ejercito ▪ Url: http://www.espe.edu.ec ▪ Imágenes: 1 imágenes	El logotipo ha sido registrado exitosamente	X	
		Editar el logotipo: ▪ Nombre: ESPE ▪ Detalle: Universidad de las fuerzas armadas ESPE. ▪ Url: http://www.espe.edu.ec ▪ Imágenes: 2 imágenes	El logotipo ha sido actualizado exitosamente	X	
		Eliminar logotipo: ▪ Nombre: ESPE	El logotipo ha sido eliminado exitosamente	X	
2	Búsqueda de logotipos	Dado una imagen que pertenece a un logotipo registrado en el sistema	Se visualiza la siguiente información del logotipo encontrado: ▪ Imagen de consulta. ▪ Titulo ▪ Descripción ▪ URL	X	
		Dado una imagen que no pertenece a ningún logotipo registrado en el sistema.	No se encontraron resultados	X	

Nota. Fuente: Tipantuña, L. Ñauñay M. (2013)

CAPÍTULO 6

CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES

Teniendo en cuenta los inconvenientes que acarrea la limitada cantidad de recursos de un teléfono móvil inteligente y la afectación que esto conlleva sobre las aplicaciones móviles de tiempo real; esta tesis concluye que el mejor algoritmo de reconocimiento de imágenes sobre el sistema operativo Android es ORB, debido a que en la etapa de pruebas técnicas descritas en la sección 4.3.4, este algoritmo fue el que mejor desempeño mostro tanto en eficiencia, como en eficacia.

Luego de haber finalizado la investigación, descrita en el capítulo 4, se llegó a la conclusión que: los descriptores locales analizados no tienen un buen desempeño con logotipos, es así que el mejor descriptor (ORB) apenas obtuvo un 41% de desempeño con este tipo de imágenes, esto se debe a que los algoritmos estudiados trabajan mejor con imágenes complejas, entendiéndose como una imagen compleja aquella imagen que cuenta con muchas texturas y encierra muchos objetos dentro de ella.

Durante el desarrollo del prototipo se encontró que un factor a tener en cuenta es la “selección del mecanismo de búsqueda de correspondencias” ya que de este dependerá la fiabilidad de los resultados entregados al usuario. Este tema al no estar

dentro del alcance del proyecto no fue investigado, pero se recomienda un análisis de estos mecanismos como trabajo futuro.

En lo referente al sistema operativo se concluye que Android aporta solidez al proceso de reconocimiento de imágenes, de hecho en todas las pruebas realizadas se observó que el sistema nunca colapso, denotando así su poderío a pesar de estar optimizado para trabajar en baja potencia y con poca memoria.

En cuanto a la hipótesis planteada en el capítulo 1, se concluye la misma como verdadera, dado que la adecuada selección de un algoritmo de reconocimiento de imágenes, sí mejora la eficiencia y eficacia de una aplicación que corre sobre el sistema operativo Android.

6.2 RECOMENDACIONES

- Se recomienda el uso de los algoritmos SURF y ORB en el reconocimiento de imágenes complejas, entendiéndose como una imagen compleja aquella imagen que posee una gran cantidad de características.

- Se promueve como trabajo futuro investigar la mejora de los algoritmos estudiados en lo referente al reconocimiento de logotipos o en su defecto la búsqueda de nuevos algoritmos.

- Se sugiere el análisis de eficiencia y eficacia de los algoritmos SIFT, SURF y ORB, pero en sistemas operativos móviles como iOS, Windows phone o BlackBerry.

- Se recomienda estudiar la posibilidad de combinar los algoritmos SIFT, SURF y ORB, con el objetivo de obtener lo mejor de los tres y colocarlo en uno solo.

- Se promueve la combinación de los descriptores locales estudiados con otros algoritmos de reconocimiento de imágenes como por ejemplo el Histograma de color, para así mejorar el proceso de reconocimiento de imágenes.

- Con respecto al prototipo diseñado se recomienda la investigación de nuevas tecnologías como OCR para mejorar los tiempos de respuesta en la búsqueda de imágenes

BIBLIOGRAFÍA

DevelopersAndroid. (2012). Recuperado el 15 de Marzo de 2013, de Service: http://developer.android.com/reference/android/app/Service.html

Actualidadg. (2012). *Actualidadg*. Recuperado el 1 de Marzo de 2013, de Que es android, historia conceptos y evolución del sistema operativo para dispositivos móviles: http://actualidadg.com/2012/10/que-es-android-historia-conceptos-y-evolucion-del-sistema-operativo-para-dispositivos-moviles/

ALEGSA. (1998 - 2013). *Definición de smartphone.* Recuperado el 05 de 2013, de Diccionario de informática: http://www.alegsa.com.ar/Dic/smartphone.php

AndroidDevelopers. (2012). *Developers Android*. Obtenido de Activity: http://developer.android.com/reference/android/app/Activity.html

Belatrix. (2012). Recuperado el 15 de Marzo de 2013, de Desarrollo Aplicaciones Móviles Sobre Celulares y Smartphones: http://www.belatrixsf.com/index.php/spdesarrollosmoviles

Boullosa, Ó. (2011). Estudio comparativo de descriptores visuales para la detección de escenas cuasi-duplicadas. *Articulo Tecnico*.

Calonder M, L. V. (2011). BRIEF: Binary Robust Independent Elementary Feature.

Carroll, A. (2010). *AnAnalysis of PowerConsumption in a Smartphone.*

Chen, R. Z. (2007). A Multi-scale Phase Method for Content Based Image Retrieval. *Fourth International Conference on Image and Graphics ICIG*, (págs. 795 - 800).

Cisco. (2012). Global Mobile Data Traffic. *Articulo tecnico* .

Condesa. (4 de Julio de 2011). *Androideity*. Recuperado el 1 de Marzo de 2013, de Arquitectura de Android: http://androideity.com/2011/07/04/arquitectura-de-android/

Delgado, B. V. (2012). Descubriendo la anatomía de una aplicación sobre Android. *Articulo Tecnico*.

DevelopersAndroid. (2012). Recuperado el 1 de Abril de 2013, de http://developer.android.com/intl/es/sdk/index.html

DevelopersAndroid. (2012). Obtenido de Emulator Android: http://developer.android.com/intl/es/tools/help/emulator.html

DevelopersAndroid. (2012). Recuperado el 1 de Abril de 2013, de Herramientas SDK y NDK: http://developer.android.com/tools/sdk/ndk/index.html

DevelopersAndroid. (2012). *Content Provider*. Obtenido de http://developer.android.com/intl/es/reference/android/content/ContentProvider.html [consultada 15 de Marzo del 2013]

Dwarakanath D, E. A. (2012). Evaluating Performance of Feature Extraction Methods forPractical 3D Imaging Systems. *Conference on Image and Vision Computing* , (págs. 250 - 255). New Zeland .

F. Provost, T. F. (1998). The case against accuracy estimation for comparing induction algorithms. *Proceedings of the Fifteenth International Conference on Machine Learning*, (pág. vol. 445).

García, C. H. (2010). Recuperado el 8 de Marzo de 2013, de Desarrollo en aplicaciones Android: http://es.scribd.com/doc/47567357/Desarrollo-de-Aplicaciones-en-Android

Geospatial. (2012 - 2013). *Geospatial*. Obtenido de Tipos de aplicaciones móviles: http://geospatialtraininges.com/recursos-gratuitos/tipos-de-aplicaciones-moviles/

Gonzales, A. (2011). *xatakandroid*. Recuperado el 13 de 02 de 2013, de http://www.xatakandroid.com/sistema-operativo/que-es-android

Guoshen, Y. J. (2009). *ASIFT: A New Framework for Fully A ne Invariant Image Comparison.* Citeseer.

Harris, M. a. (1998). A combined corner and edge detector. *In Alvey Vision Conference*, (págs. 147–151).

Holte, C. D. (2000). Explicitly representing expected cost: An alternative to roc representation. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, (págs. 198–207).

J. Huang, S. R.-J. (1999). Spatial color indexing and applications. *International Journal of Computer Vision*.

Juárez, C. (2011). Reconocimiento e identificación de logotipos en imágenes con trasformada SIFT. *Articulo Tecnico*.

Lowe, D. G. (1999). Reconocimiento de objetos locales de escala invariante características. *Conferencia Internacional sobre la Visión por Computador*.

MalikYasir, M. (2012). *Power Consumption Analysis of a Modern Smartphone.*

Martinez, L. (2011). *Aplicaciones para dispositivos móviles*. Obtenido de http://riunet.upv.es/bitstream/handle/10251/11538/Memoria.pdf?sequence=1

Monteagudo, A. C. (2005). *Tecnologías móviles con Java.* Obtenido de http://www.iti.es/media/about/docs/tic/08/articulo1.pdf

Moreno, D. (2011). mcBrief: un descriptor local de features para imágenes color. *Articulo Tecnico*.

Object Technology International, I. (2003). *Eclipse Platform Technical Overview*. Obtenido de http://eclipse.org/whitepapers/eclipse-overview.pdf

OpenCV. (2013). Obtenido de http://opencv.org/

OpenCVWiki. (2013). Obtenido de http://code.opencv.org/projects/opencv/wiki

PCWorld. (2012). Android e iOS se disputan el mercado de smartphones este año. *PCWorld*.

Ramisa A, V. A. (2011). Evaluation of the SIFT ObjectRecognition Method in Mobile Robots. *Articulo Tecnico*.

Redes&Telecom. (2012). Especial de redes WLAN. *Redes&Telecom*.

Rodriguez, A. (2012). *Módulos de la librería OpenCV*. Obtenido de http://digitimagen.blogspot.com/2012/08/modulos-de-la-libreria-opencv.html

Rosten, E. D. (2011). Machine learning for high-speed corner detection.

Rublee. E, R. V. (2011). ORB: an eficient alternative to SIFT or SURF.

Salas, D. (2011). Recuperado el 24 de Febrero de 2012, de la historia y los comienzos de Android, el sistema operativo de google: http://www.elandroidelibre.com/2011/08/la-historia-y-los-comienzos-de-android-el-sistema-operativo-de-google.html

Schmid., K. M. (2005). A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*

TechRepublic. (2010). Steve Jobs proclaims the post-PC era has arrived. *TechRepublic*.

Theverge. (2012). 850k Android activations a day. *Theverge*.

Tofel, K. (2012). *BloombergBusinessweek*. Obtenido de Time for Google to Take Control of Android: http://www.businessweek.com/articles/2012-05-09/time-for-google-to-take-control-of-android

Torres, R. C. (2011). Diseño e implementacion de un sistema remoto de monitoreo de cámaras de seguridad mediante el desarrollo de una aplicación para un dispositivo móvil celular. *Articulo Tecnico*.

Wikipedia. (2010). Obtenido de http://es.wikipedia.org/wiki/Eclipse_(software)

Wikipedia. (2012). *Wikipedia*. Recuperado el 13 de Febrero de 2012, de es.wikipedia.org/wiki/Android?

Ziegler, A. C. (2012). Locally Uniform Comparison Image Descriptor. *Articulo Tecnico*.