Universidad Nacional Autónoma de México

Facultad de Ciencias

REALIDAD AUMENTADA - UNA APLICACIÓN PARA EL SISTEMA OPERATIVO ANDROID

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
SEBASTIÁN GARCÍA ANDERMAN

DIRECTOR DE TESIS:
DR. JOSÉ DE JESÚS GALAVIZ CASAS

2011

Hoja de datos del jurado

1. Datos del alumno
Sebastián
García
Anderman
56 81 35 73
Universidad Nacional Autónoma de México
Ciencias de la Computación
301696151

2. Datos del tutor
Dr.
José de Jesús
Galaviz
Casas

3. Datos del sinodal 1
Dra.
Elisa
Viso
Gurovich

4. Datos del sinodal 2
L. en C.C.
Francisco Lorenzo
Solsona
Cruz

5. Datos del sinodal 3
Dra.
Amparo
López
Gaona

6. Datos del sinodal 4
Mat.
Salvador
López
Mendoza

7. Datos del trabajo escrito
Realidad Aumentada - Una Aplicación Para El Sistema Operativo Android
114 páginas
2011

A mi padre

Índice general

Índice de figuras

[1]Punto de Interés que se mostrará en el dispositivo.

Índice de código

Índice de Algoritmos

Capítulo 1

Introducción

1.1. Realidad aumentada

La realidad aumentada es una variación de la Realidad Virtual; en las tecnologías de Realidad Virtual se introduce al usuario en un ambiente completamente virtual donde no se tiene contacto con nada del mundo real a su alrededor. En cambio, la realidad aumentada permite al usuario ver el mundo real y sobrepone o compone objetos virtuales sobre él. De este modo, la realidad aumentada no reemplaza la realidad, la complementa. Idealmente, la realidad aumentada hace que al usuario le parezca que los objetos reales y virtuales coexisten en el mismo espacio.

Se cree que el término fue usado por primera vez en 1990 por Tom Caudell, un investigador de Boeing, para describir un dispositivo que guiaba a los trabajadores al ensamblar cables en los aviones que ahí construían [21]. Esa temprana definición de realidad aumentada era una intersección entre el mundo virtual y físico, donde imágenes digitales se mezclaban con el mundo real para aumentar nuestra percepción. En la figura 1.1 podemos ver un ejemplo donde se muestra a una persona sosteniendo una hoja que posee un marcador detectado por una cámara, que reemplaza la imagen de la hoja por una calle y pone un automóvil sobre ella.

La definición de realidad aumentada requería, inicialmente, del uso de un HMD[1]. Posteriormente se quita este requerimiento para no limitar la definición de realidad aumentada a tecnologías específicas, con lo que la realidad aumentada debe tener tres características [13]:

[1] Siglas del inglés Head Mounted Display. Dispositivo de visualización similar a un casco que permite reproducir imágenes creadas por computadora sobre una pantalla o similar muy cercano a los ojos o directamente sobre la retina.

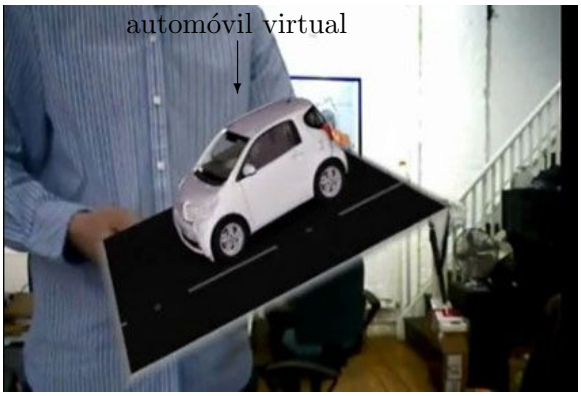


Figura 1.1: Persona sosteniendo una hoja con un automóvil virtual agregado con realidad aumentada

1. Combinar el mundo real y virtual.
2. Interaccionar en tiempo real.
3. Mostrarse en 3D.

Esta definición permite otras tecnologías además de HMD, manteniendo los componentes principales de realidad aumentada. No se incluyen, por ejemplo, efectos especiales en películas donde los objetos virtuales se mezclan con un ambiente real en 3D pero no son interactivos. Los objetos virtuales mostrados deben tener un fuerte vínculo con el ambiente real, ya sea información u otra cosa lo que estemos desplegando. Este vínculo es más que nada una relación espacial entre los objetos virtuales y el mundo real [26].

Para este trabajo omitiremos el tercer punto de la definición de realidad aumentada, mostrar los objetos virtuales en 3D, ya que nuestro objetivo será el proporcionar al usuario información adicional sobre lo que está viendo en el mundo real, por lo que no es necesario que nuestros objetos virtuales parezcan formar parte del mundo real.

Lo esencial de la realidad aumentada es la interacción entre el mundo real y nuestro contenido virtual agregado a él, por lo que nuestros objetos virtuales deben colocarse en relación al mundo real. Esto no presentaría problemas en sistemas fijos, pero la mayoría de las aplicaciones de realidad aumentada se hacen en dispositivos que pueden moverse y por lo tanto cambia la dirección del mundo real hacia la que enfocan y debemos reubicar nuestros objetos virtuales. En realidad aumentada se le llama *rastreo* al proceso de

Figura 1.2: *Head Mounted Display* HMD [17]

seguimiento de las coordenadas de la escena de objetos en movimiento en tiempo real. Generalmente se realiza el rastreo con un HMD o un dispositivo móvil. Se han usado principalmente dos métodos para este rastreo:

- Rastreo óptico.
- Rastreo con sensores GPS[2], acelerómetro multieje[3], magnetómetro[4]).

El rastreo óptico se realiza detectando algún objeto dentro del campo de visión del dispositivo de realidad aumentada y rastreando su movimiento. En muchos casos se utiliza un marcador particular llamado *fiduciario*, estos marcadores generalmente son imágenes en blanco y negro, no simétricas y particulares, de tal modo que el dispositivo de realidad aumentada pueda rastrearlo fácilmente y diferenciarlo de otros objetos en la escena. Lo que se hace con estos marcadores es ubicarlos y colocar nuestro contenido virtual sobre ellos como podemos ver en la figura 1.3, pudiendo seguir su movimiento y así mantiene la parte interactiva. Una escena puede tener más de un marcador fiduciario siempre y cuando sean diferentes, con lo que se puede colocar distinto contenido virtual sobre cada uno de ellos.

[2]Siglas del inglés Global Positioning System. Sistema que permite conocer la posición de un objeto móvil gracias a la recepción de señales emitidas por una red de satélites.

[3]Dispositivo que mide la aceleración. Los acelerómetros multieje detectan la magnitud y la dirección de la aceleración como una cantidad vectorial y pueden utilizarse para determinar orientación.

[4]Aparato que mide la intensidad, y algunas veces también la dirección, de un campo magnético.

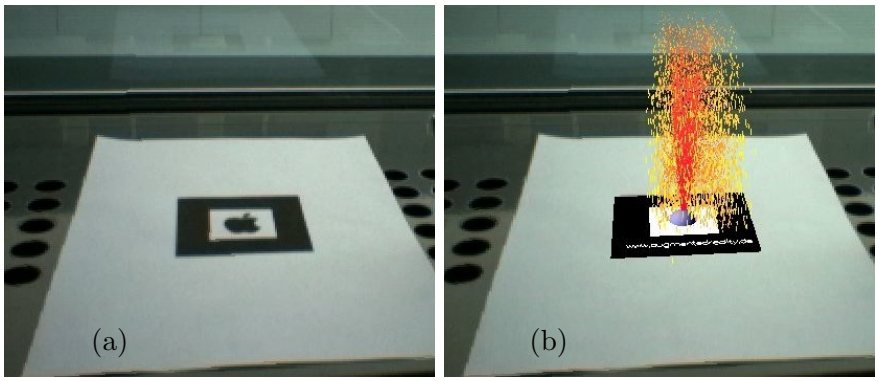


Figura 1.3: (a) Marcador fiduciario sobre una hoja de papel (b) Contenido virtual colocado sobre el marcador fiduciario

El rastreo por medio de sensores (el cual usaremos en este trabajo), funciona detectando la posición y orientación del dispositivo de realidad aumentada por medio de sus distintos sensores, sin la ayuda de elementos ópticos. Al necesitar la posición y orientación del dispositivo, podemos representar esto con seis variables independientes (tres coordenadas de traslación y tres ángulos de rotación). Este tipo de sistemas se llaman sistemas de rastreo con seis grados de libertad o 6dof[5] [15]. Las primeras tres coordenadas las obtenemos gracias al GPS, y las segundas tres por medio del magnetómetro y acelerómetro multieje. Es importante notar que con este sistema debemos también saber las coordenadas de los objetos reales con los que vamos a interactuar; en general será suficiente con tener solamente su ubicación.

Dado que las aplicaciones de realidad aumentada deben generar el contenido virtual en tiempo real, los sistemas deben rastrear los objetos a una tasa de al menos 30Hz para que se ajuste a la percepción humana.

1.2. Utilidad de la realidad aumentada

¿Por qué la realidad aumentada es un tema interesante? ¿Por qué es útil el combinar objetos reales y virtuales?

- La realidad aumentada aumenta la percepción y la interacción del mundo real al usuario.

[5]Siglas del inglés 6 Degrees of Freedom.

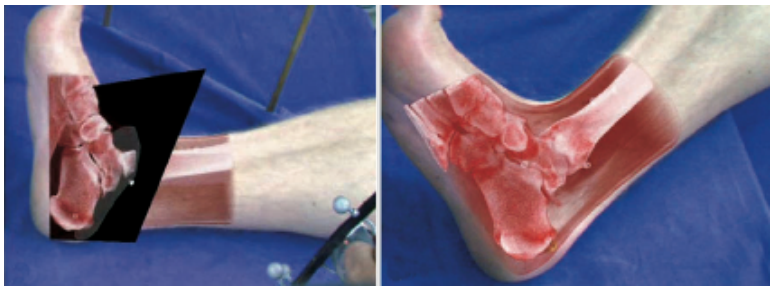


Figura 1.4: Sistema médico de realidad aumentada utilizado con un HMD[18]

- Los objetos virtuales muestran información que el usuario no puede detectar con sus propios sentidos.
- La información obtenida por el objeto virtual ayuda al usuario a realizar tareas del mundo real.

Actualmente la tecnología es usada en innumerables áreas para ayudarnos a realizar tareas, proporcionarnos información, etc. Pero en general el mundo real y el mundo virtual están separados, y son nuestros dispositivos los que permiten vincularlos. La realidad aumentada nos permite mezclar estos mundos y ver la información obtenida por computadora en el mundo real a nuestro alrededor.

Se han explorado varias clases de aplicaciones potenciales de realidad aumentada. Algunas de éstas son [13]: médicas, visualización, mantenimiento y reparación, información, planeación de rutas para robots, entretenimiento y militares.

1.3. Realidad aumentada en teléfonos móviles

En los últimos años, los teléfonos móviles se han convertido en una plataforma ideal para la realidad aumentada. Los teléfonos de últimas generaciones tienen pantallas a color, cámara integrada, procesadores veloces, GPS y sensores como acelerómetros y magnetómetros. Además, la gran adopción de los teléfonos móviles hace que puedan ser una de las plataformas dominantes para realidad aumentada.

Tradicionalmente los sistemas de realidad aumentada se realizaban sobre HMD, siendo uno de los primeros *Feiner's Touring Machine* [14] como podemos ver en 1.5.

Figura 1.5: Prototipo de sistema de información de realidad aumentada. El usuario lleva una mochila, un HMD y sostiene una computadora de mano

Al usar un HMD el usuario queda con las manos libres para interactuar con el contenido virtual, ya sea directamente o con algún dispositivo como un ratón o similar. Sin embargo, para dispositivos de realidad aumentada como teléfonos celulares, el usuario mira la pantalla del dispositivo y necesita, al menos, una mano para sostenerlo y puede interactuar con el contenido virtual con la otra, normalmente a través de una pantalla táctil en el dispositivo.

1.4. Objetivo general

En este trabajo se construirá una aplicación de realidad aumentada para un teléfono móvil con el sistema operativo Android. Esta aplicación utilizará el sensor GPS del dispositivo para detectar dónde se encuentra, el magnetómetro para detectar la dirección en la que apunta, así como el acelerómetro multieje para detectar el ángulo en que está inclinado el dispositivo. Con esta información, la aplicación podrá obtener información relevante de distintas fuentes en línea, como Wikipedia, para entonces utilizar realidad aumentada y colocar información adicional sobre lo que se está viendo en la pantalla del dispositivo, que es una vista en vivo de la cámara.

Un ejemplo de uso sería estar en el Zócalo de la Ciudad de México, abrir la aplicación y apuntar el dispositivo a la Catedral Metropolitana; la

aplicación entonces pondría una marca encima de ella con su nombre, y al hacer click sobre esta marca en la pantalla podríamos ver la entrada de la Wikipedia para este edificio.

1.5. Descripción general del trabajo

Después de la introducción sobre la realidad aumentada, se abarcarán los siguientes temas:

- Android (sección 2): Describe la historia de la plataforma Android así como su arquitectura.
- Diseño de la aplicación (sección 3): Describe el objetivo de la aplicación, su arquitectura, componentes internos y de interfaz gráfica.
- Sensores (sección 4): Explica el funcionamiento del magnetómetro y el acelerómetro multieje del dispositivo, así como la manera de usarlos para calcular la dirección en la que apunta el teléfono. También se describen y comparan algunos métodos de filtrado de ruido para suavizar los datos obtenidos de estos sensores.
- GPS (sección 5): Describe el funcionamiento de un sensor GPS, así como el método utilizado para calcular distancias entre dos puntos sobre la superficie terrestre.
- Sobreposición de imágenes sobre la cámara (sección 6): Describe los métodos empleados para calcular la posición de los elementos a dibujar sobre la pantalla para que correspondan con lo que la cámara del dispositivo está desplegando.

Capítulo 2

Android

Android es una plataforma para dispositivos móviles como teléfonos celulares, computadoras portátiles y táctiles. Andy Rubin[1] lo describe como:

> "Android es la primer plataforma para dispositivos móviles realmente abierta y completa. Incluye un sistema operativo, interfaz de usuario y aplicaciones. Todo el software para ejecutar en un teléfono móvil, pero sin los obstáculos propietarios que han detenido la innovación móvil." [19]

2.1. Historia

Históricamente, los desarrolladores, generalmente programando en `C` o `C++`, han tenido que entender el hardware específico en el que trabajan, siendo muchas veces un solo dispositivo o un rango de dispositivos de un fabricante. Con el desarrollo de tecnologías de hardware móvil este enfoque ha ido cambiando. Recientemente plataformas como Symbian[2] han sido creadas para darle a los desarrolladores una audiencia más amplia, ya que al programar para alguno de esos sistemas tienen la ventaja de que su aplicación funcionará en muchos dispositivos diferentes.

Estas plataformas ofrecen acceso al hardware del dispositivo, pero requieren que los desarrolladores escriban código complejo en `C/C++` y usen interfaces de programación de aplicaciones (API[3]) propietarias, que gene-

[1] Vicepresidente de ingeniería de Google y supervisor del desarrollo de Android

[2] Sistema operativo móvil desarrollado por Nokia. http://www.symbian.org/

[3] Siglas del inglés Application Programming Interface. Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

ralmente son complicadas de usar. Esta dificultad es todavía mayor en aplicaciones que deben funcionar en diferentes implementaciones de hardware y, sobre todo, en las que usan alguna característica específica del hardware, como el GPS.

Posteriormente, uno de los grandes avances en el desarrollo para teléfonos móviles fue la introducción de MIDlets[4] [28]. Estos son ejecutados en una JVM[5], abstrayendo el hardware y permitiendo a los desarrolladores crear aplicaciones que funcionen en una gran cantidad de dispositivos que soporten el entorno de ejecución Java. Lamentablemente esta facilidad hace que se tenga un acceso muy restringido al hardware del dispositivo.

Android se encuentra dentro de una nueva gama de sistemas operativos móviles que han surgido recientemente, diseñados para trabajar con hardware móvil mucho más poderoso. iOS[6], Windows Phone 7[7], proveen ambientes de desarrollo simplificados pero con mayores capacidades. Sin embargo, a diferencia de Android, estos sistemas operativos son propietarios. Al darle prioridad, a veces, a las aplicaciones nativas sobre las creadas por terceros, restringen la comunicación entre las aplicaciones y los datos del teléfono y restringen o controlan la distribución de aplicaciones de terceros en sus plataformas.

Las API y herramientas disponibles para desarrollar aplicaciones móviles son demasiado restrictivas y muchas veces parecen estar detrás de las *infraestructuras* (frameworks) de escritorio. Aquí es donde Google vio una oportunidad y en 2005 adquiere la compañía Android Inc. para iniciar el desarrollo de la plataforma Android, la cual prometía ser abierta, asequible, código de fuente abierta y una infraestructura de desarrollo sofisticada.

En 2007 un grupo de líderes de la industria de dispositivos móviles se reunieron alrededor de la plataforma Android para formar la Open Handset Alliance[8]. Algunos de los miembros de esta alianza son:

[4]Un MIDlet es una aplicación que utiliza el perfil MIDP (Mobile Information Device Profile) y la especificación CLDC (Connected Limited Device Configuration), ambos para Java ME.

[5]Siglas del inglés Java Virtual Machine, o máquina virtual de Java. Es un máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (Java bytecode), el cual es generado por el compilador del lenguaje Java.

[6]Sistema operativo móvil desarrollado por Apple para ejecutarse en el iPhone, está basado en Mac OS X. http://developer.apple.com/iphone/

[7]Sistem operativo móvil desarrollado por Microsoft. http://www.windowsphone7.com/

[8]Alianza abierta para los teléfonos móviles. http://www.openhandsetalliance.com

- Sprint Nextel
- T-Mobile
- Motorola
- Samsung
- Sony Ericsson
- Toshiba
- Vodafone
- Google
- Intel
- Texas Instruments

Una de las metas de esta alianza es innovar rápidamente y responder de mejor manera a las necesidades de los consumidores y su primer resolución fue usar la plataforma Android. Además, los miembros se han comprometido a liberar una cantidad significativa de propiedad intelectual a través de la licencia de software libre Apache License, Version 2.0[9].

En septiembre de 2008 Google anunció la disponibilidad del SDK[10] 1.0 e hizo el código fuente de la plataforma Android disponible a través de la licencia de código abierto de Apache. También en esta fecha T-Mobile[11] anunció el G1, primer teléfono con Android en el mercado.

Después de esa primera versión han habido otras cinco actualizaciones a la plataforma, siendo la última la versión 2.3 [1] lanzada en diciembre de 2010.

También ha crecido enormemente la cuota de mercado de teléfonos inteligentes con Android en los últimos dos años, teniendo en agosto de 2010 17.2 %[2] y 160,000 nuevos usuarios cada día[22]. A mediados del 2010 se estimaba que había más de 40 teléfonos en el mercado de más de 10 compañías con Android[29] convirtiéndose así en una excelente plataforma para los desarrolladores que quieran tener una gran base de usuarios para sus aplicaciones.

El código fuente de Android es completamente abierto, los detalles de su distribución se publican en http://source.android.com y se puso a disponibilidad del público en Octubre de 2008.

El código fuente de Android y todos sus proyectos están manejados por el sistema de control de versiones Git[12]. La lista completa de los proyectos de Android en su repositorio de Git se encuentra en http://android.git.kernel.org/

[9] Licencia de software libre. http://www.apache.org/licenses/LICENSE2.0.html

[10] Siglas del inglés Software Development Toolkit, o kit de desarrollo de software. Es generalmente un conjunto de herramientas de desarrollo que le permiten a un programador crear aplicaciones para un sistema concreto.

[11] Operadora alemana de telefonía móvil con subsidiarias en Estados Unidos.

[12] Sistema de control de versiones distribuido con énfasis en la velocidad. Fue originalmente diseñado y desarrollado por Linus Torvalds. http://git-scm.com/

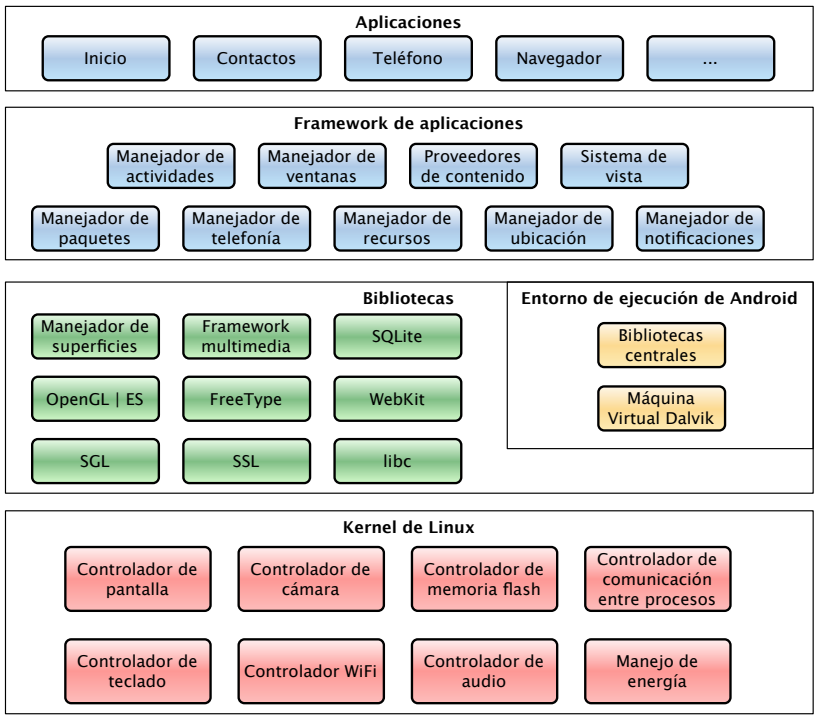


Figura 2.1: Arquitectura del Sistema Android

2.2. Arquitectura

La plataforma Android adopta la idea de cómputo general para dispositivos móviles. Está basado en un sistema operativo Linux para manejar los dispositivos, memoria y procesos. Las bibliotecas de Android abarcan telefonía, video, gráficos, diseño de interfaces gráficas y otros aspectos del dispositivo. En la figura 2.1 podemos ver el esquema de la arquitectura de Android.

Android depende de la versión 2.6 de Linux para sus componentes principales como seguridad, manejo de memoria, manejo de procesos, pila de red y modelo de controladores. El kernel también actúa como una capa de abstracción entre el hardware y el resto del software.

Encima del kernel está la siguiente capa que contiene las bibliotecas nativas de Android. Éstas están programadas en C o C++ y compiladas para la arquitectura particular del dispositivo. Todas ellas tienen clases de `Java` que las encapsulan y permiten a los programadores usarlas. Además

Android también permite que se desarrollen aplicaciones usando su kit de desarrollo nativo NDK[13].

Algunas de las bibliotecas nativas más importantes son:

- **Manejador de superficie:** Android utiliza un manejador de ventanas similar a Vista o Compiz, pero mucho más sencillo. En lugar de dibujar directamente sobre el buffer[14] de pantalla, los comandos de dibujo van a mapas de bits que se combinan con otros para formar lo que el usuario ve en la pantalla. Gracias a esto el sistema permite crear distintos tipos de efectos.
- **Gráficos 2D y 3D:** Elementos en dos y tres dimensiones pueden combinarse en una sola interfaz de usuario con Android. La biblioteca utiliza hardware de aceleración 3D en caso de que el dispositivo la posea; de lo contrario, utiliza un procesador gráfico de software. Incluye las bibliotecas SGL[15] y OpenGL ES[16] para trabajar con gráficos.
- **Codecs[17] de medios:** Android puede reproducir y grabar audio y video en varios formatos incluyendo AAC, AVC (H.264), H.263, MP3 y MPEG-4.
- **Base de datos SQL[18]:** Android incluye el motor de bases de datos SQLite [6].
- **Motor de navegador web:** Para mostrar contenido HTML[19] Android utiliza la biblioteca WebKit[20].

[13]Siglas del inglés Native Development Kit. Es un kit de desarrollo para poder programar directamente sobre la plataforma de cómputo que se está usando y no usar la máquina virtual.

[14]Región de memoria utilizada temporalmente para almacenar datos mientras se mueven de un lugar a otro.

[15]Siglas del inglés Scalable Graphics Library. Es el subsistema gráfico utilizado por Android.

[16]Siglas del inglés Open Graphics Library for Embedded System. Es un subconjunto de la API de gráficos en 3D OpenGL diseñado para sistemas embebidos como teléfonos celulares, PDA y consolas de videojuegos. http://www.khronos.org/opengles/

[17]Siglas del inglés COmpressor-DEcompressor o, más comúnmente, COder-DEcoder. Es un dispositivo o programa capaz de codificar y/o decodificar un flujo de datos digitales o una señal.

[18]Siglas del inglés Structured Query Language. Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas

[19]Siglas del inglés HyperText Markup Language. Es el lenguaje de marcado predominante para la elaboración de páginas web.

[20]Motor de disposición diseñado para permitir a navegadores web procesar páginas web. Es utilizado por navegadores como Safari y Chrome. http://webkit.org/

El SDK de Android soporta la mayoría de las bibliotecas de Java, excepto AWT[21] y Swing[22]. En lugar de ellos, Android tiene su propia sistema para realizar interfaces gráficas. Al realizarse la programación en Java, se necesita una JVM que se encarga de interpretar el bytecode[23] de Java. Una JVM provee optimizaciones para ayudar a Java a alcanzar un buen desempeño, tratando de ser comparable con lenguajes compilados como C y C++. Android tiene su propia Máquina Virtual, llamada Dalvik VM[24], optimizada para ejecutar las clases compiladas de Java tratando de contrarrestar las limitaciones de los dispositivos portátiles como memoria, velocidad de procesador y energía.

Máquina Virtual Dalvik

Para Android, Google se tomó mucho tiempo en hacer optimizaciones para dispositivos móviles de poca capacidad. Estos dispositivos están muy por detrás de los sistemas de escritorio, tanto en memoria como en velocidad. Tienen muy poco poder de procesamiento, su memoria RAM puede ser tan pequeña como 64MB y su espacio disponible para aplicaciones de 20MB. Además, también deben considerarse optimizaciones para maximizar el rendimiento de la batería.

> **Nota:**
>
> El primer teléfono con Android, el G1 de T-Mobile que salió a finales de 2008, posee 192MB de RAM, una tarjeta de memoria microSD de 1GB y un procesador Qualcomm MSM7201A de 528MHz. El Droid de Motorola que salió a finales de 2009 posee 256MB de RAM, una tarjeta de memoria microSD de 16GB y un procesador Arm Cortex de 550MHz. Comparemos esas características con las de una laptop Dell de modelo económico, que posee un procesador de 2GHz de doble núcleo y 4GB de RAM.

Debido a estos requerimientos de desempeño en los dispositivos móviles impuestos por sus limitaciones de hardware, las optimizaciones son esen-

[21] Siglas del inglés Abstract Window Toolkit. Es el primer conjunto de herramientas de interfaz gráfica de Java.

[22] Swing es el principal conjunto de herramientas de interfaz gráfica de Java.

[23] El bytecode es un código intermedio más abstracto que el código de máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código de máquina.

[24] La máquina virtual Dalvik fue diseñada e implementada por Dan Bornstein quien la bautizó *Dalvik* en honor a un pueblo de Islandia donde vivieron algunos de sus antepasados.

Tipo de archivo	Tamaño (bytes)	Porcentaje
Archivo jar sin comprimir	470312	100 %
Archivo jar comprimido	232065	49 %
Archivo dex sin comprimir	209248	44 %

Tabla 2.1: Tamaño del navegador Web de Android[20]

ciales. Si vemos la lista de paquetes de Android, podemos ver que ocupan alrededor de 20MB [28], incluso con su JVM optimizada. Es por esto que Google tuvo que replantearse la implementación de la JVM estándar en muchos aspectos.

La máquina virtual Dalvik toma los archivos *class* generadas por Java y las combina en uno o más archivos del formato Dalvik Executable (.dex), el cual reusa información duplicada de los diferentes archivos *class*, con lo que reduce a la mitad aproximadamente el tamaño de los archivos sin comprimir como podemos ver en el cuadro 2.1.

Google ajustó la recolección de basura en la máquina virtual Dalvik y en la versión 2.2[25] de Android incluyó compilación en tiempo de ejecución (conocida como JIT[26]) para ayudar con el desempeño de las aplicaciones. Gracias a la implementación de la compilación JIT, la versión 2.2 presenta un desempeño de 2 a 5 veces mayor en tareas que dependen directamente del CPU a comparación de la versión 2.1 [23]. Esta mejora es para aplicaciones programadas en `Java`, que son la mayoría de las desarrolladas por programadores externos a Google.

La máquina virtual Dalvik utiliza un tipo diferente de generación de código de máquina, en el cuál se usan registros como unidades primarias de almacenamiento de datos en lugar de utilizar la pila de memoria. Con esto Google ha lograda tener 30 % menos instrucciones [20] en comparación con las implementaciones tradicionales. Cada aplicación de Android corre en su propio proceso, con su propio ejemplar de la máquina virtual Dalvik. Ésta fue diseñada para que el dispositivo pueda ejecutar varias máquinas virtuales de manera eficiente. La máquina virtual ejecuta clases compiladas por el compilador de Java que se han transformado en el formato .dex por la herramienta "dx" incluida en el kit de desarrollo de Android.

[25] La versión 2.2 de Android con nombre Froyo fue liberada el 20 de mayo de 2010.

[26] Siglas del inglés Just In Time. La compilación en tiempo de ejecución es una técnica para mejorar el rendimiento de sistemas de programación que compilan a bytecode; consiste en traducir el bytecode a código de máquina nativo en tiempo de ejecución.

Capítulo 3

Diseño del programa

3.1. Objetivo

El objetivo del proyecto es desarrollar un programa de realidad aumentada para la plataforma Android que proveerá al usuario con información adicional sobre su entorno. Para esto mostraremos en la pantalla del dispositivo la imagen en vivo de la cámara del mismo, y sobrepondremos la información adicional sobre esta imagen. Definiremos el término PI como el punto que mostrará el programa sobre la imagen.

El programa utilizará el sensor GPS del dispositivo para conocer su ubicación y así saber cuáles PI hay a su alrededor. Se usará el magnetómetro y el acelerómetro multieje para calcular la dirección a la que apunta el dispositivo y de este modo colocar la información en la pantalla sobre el PI que vemos en la imagen de la cámara del dispositivo. En la figura 3.1 podemos ver cómo el usuario está en el Zócalo de la Ciudad de México apuntando el dispositivo a la Catedral Metropolitana. El programa detecta la posición geográfica del dispositivo así como hacia dónde está apuntando y sobrepone un marcador para indicar que hay un punto de interés en ese lugar y muestra cual es el nombre de ese PI.

Figura 3.1: Diseño de indicación de un PI. Se muestra una imagen de Zócalo de la Ciudad de México con la Catedral Metropolitana señalada por el programa como un punto de interés

El programa también nos permitirá obtener más información del punto de interés que estemos observando al tocar con el dedo alguno de los PI que vemos en la pantalla. Cada PI posee las siguientes características:

- Coordenadas geográficas.
- Nombre del PI.
- Dirección de consulta, esto es un URL con más información sobre el PI.
- Elevación (de estar disponible).

A partir de estos elementos y los datos del dispositivo, podremos calcular los siguientes datos de cada PI:

- Distancia a la que se encuentra.
- Acimut[1], en la figura 3.2 podemos ver un ejemplo.

[1]En náutica, el acimut es el ángulo o longitud de arco entre el punto cardinal norte en sentido horario de 0° a 360° y otro punto.

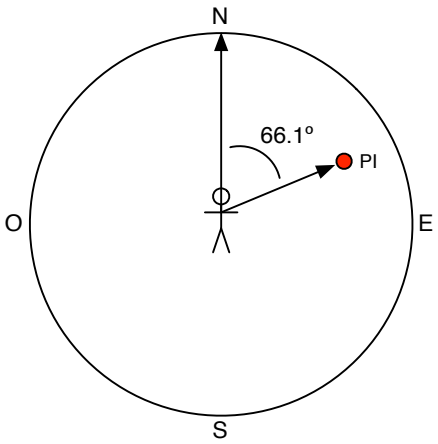


Figura 3.2: El acimut para el PI que vemos es de 66.1°. Esto lo utilizamos para calcular la posición del PI en la pantalla

- Inclinación del dispositivo al PI, si el PI no tiene altitud, la inclinación por omisión es 0.
- Posición en la pantalla.

3.2. Fuentes de información

Toda la información la descargará el programa de distintas fuentes en línea en el momento en que se ejecute, haciendo una consulta con su ubicación y un radio para indicar cuál es la distancia máxima desde el dispositivo que deben tener los PI que se muestren. Estas fuentes serán direcciones web a las cuales se les podrá hacer una consulta y enviarán al dispositivo los datos en JSON[2] o XML[3].

La principal fuente de información que ocuparemos para este proyecto es el servicio GeoNames[4], que es una base de datos que contiene alrededor

[2]Siglas del inglés JavaScript Object Notation u objeto de notación de javascript. Es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.http://www.json.org/.

[3]Siglas del inglés eXtensible Markup Language o lenguaje de marcas extensible. Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C) http://www.w3.org/XML/.

[4]Base de datos geográfica gratuita y accesible a través de Internet bajo una licencia Creative Commons. http://www.geonames.org/

de 10 millones de nombres geográficos. Este servicio cuenta con una API con la cual se pueden hacer consultas de entradas de la Wikipedia[5] que se encuentren a un cierto radio de las coordenadas provistas.

En este servicio podemos hacer una consulta a la dirección:

http://api.geonames.org/findNearbyWikipedia?

pudiendo usar los siguientes parámetros:

- lat: latitud.
- lng: longitud.
- radius: radio en el que queremos que estén los resultados, se da en `km`.
- lang: código de idioma, es decir, *"es"* para español, *"en"* para inglés, etc. Por omisión *"en"*.
- maxRows: cantidad de filas máxima en los resultados. Por omisión 5.
- country: código de país. Por omisión todos los países.
- username: nombre de usuario de la página.

De este modo si realizamos la siguiente consulta, para ver los primeros cien lugares de interés de la Wikipedia que estén a 1`km` del Zócalo de la Ciudad de México, cuyas coordenadas son, latitud 19.4344, longitud -99.1331, tendremos el siguiente URL:

`http://ws.geonames.org/findNearbyWikipedia?lat=19.4344&lng=-99.1331&lang=es&radius=1&maxRows=100`

y obtendremos como resultado lo que se presenta en el código 1, donde vemos que los primeros dos resultados son la Catedral Metropolitana y el Centro Cultural España. Esta consulta devuelve 45 resultados en español y 70 en inglés, teniendo entonces bastantes PI a solo 1`km` a nuestro alrededor estando ubicados en el Zócalo de la Ciudad de México.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames>
  <entry>
    <lang>es</lang>
```

Código 1: Los primeros dos resultados de lugares de interés a 1`km` a la redonda del Zócalo de la Ciudad de México (1/3)

[5]Enciclopedia libre y políglota de la Fundación Wikimedia. http://www.wikipedia.org/

```
    <title>
      Catedral Metropolitana de la Ciudad de México
    </title>
  <summary>
    La Catedral Metropolitana de la Ciudad de México
    es la sede de la Arquidiócesis Primada de México
    ubicada frente a la Plaza de la Constitución, en
    el centro histórico de la Ciudad de México. Las
    medidas aproximadas de este templo son 59 metros
    de ancho por 128 de largo y una altura de 60
    metros (...)
  </summary>
  <feature>landmark</feature>
  <countryCode>MX</countryCode>
  <population>0</population>
  <elevation>0</elevation>
  <lat>19.4344</lat>
  <lng>-99.1331</lng>
  <wikipediaUrl>
    http://es.wikipedia.org/wiki/Catedral_Metropoli
    tana_de_la_Ciudad_de_M%C3%A9xico
  </wikipediaUrl>
  <thumbnailImg/>
  <rank>98</rank>
  <distance>0.0067</distance>
</entry>
<entry>
  <lang>es</lang>
  <title>
    Centro Cultural de España en México
  </title>
  <summary>
    El Centro Cultural de España en México se
    encuentra localizado en el Centro Histórico de
    la Ciudad de México, México. El centro se enfoca
    al arte contemporáneo (principalmente
```

Código 1: (2/3)

```
    iberoamericano) teniendo como base los siguientes
    objetivos: apoyar a la recuperación del Centro
    Histórico de la Ciudad de (...)
  </summary>
  <feature/>
  <countryCode>MX</countryCode>
  <population>0</population>
    <elevation>0</elevation>
    <lat>19.435</lat>
    <lng>-99.1326</lng>
    <wikipediaUrl>
    http://es.wikipedia.org/wiki/Centro_Cultural_de_Es
    pa%C3%B1a_en_M%C3%A9xico
  </wikipediaUrl>
    <thumbnailImg/>
    <rank>27</rank>
    <distance>0.0825</distance>
  </entry>
</geonames>
```

Código 1: (3/3)

En el código 1 podemos ver que GeoNames nos devuelve varios datos para cada entrada, pero sólo estaremos interesados en el título, las coordenadas geográficas, la altitud y el URL con información adicional. No utilizamos la distancia pues en algunas ocasiones ha resultado no ser muy precisa, así que es mejor calcularla con nuestra posición y la del PI. También vemos que estos dos PI dicen tener una elevación o altitud de 0, lo que no quiere decir que estén a nivel del mar, sino que no se tiene esa información. En estos casos suponemos que el PI está a la misma altitud que el dispositivo.

Para poder descargar los datos desde las distintas fuentes de información que tengamos, creamos la clase abstracta `FuentePI` con algunos métodos auxiliares para cada fuente de información y el método abstracto `obtenerPI( double latitud, double longitud)` que recibe como parámetros la latitud y longitud del dispositivo.

No debemos empezar a descargar esta información inmediatamente puesto que si la precisión del GPS es muy mala en el momento de empezar a descargar, obtendremos PI que no corresponden a nuestra ubicación actual.

Es por eso que en el escucha de cambio de ubicación hacemos una verificación, si la precisión es de 50m o mejor, entonces comenzamos la descarga de información. Con esta precisión obtendremos PI que estén a nuestro alrededor aunque el GPS obtenga una mejor precisión después.

La descarga de información no debe hacerse en el hilo de ejecución principal ya que esto haría que la interfaz gráfica dejara de responder todo el tiempo que dure esta operación. Android proporciona una manera sencilla de hacer este tipo de tareas en un hilo separado, solo tenemos que extender la clase `AsyncTask` y sobrecargar el método `doInBackground`, todo lo que pongamos en este método se hará en un hilo aparte. En el código 2 podemos ver la implementación de esta clase que obtiene la información para nuestra fuente GeoNames, con una clase `FuentePIGeonames` que extiende a `FuentePI`.

Cada una de las clases que extiendan a `FuentePI` utilizan la clase `HttpConnection` para poder comunicarse con el servidor que utilicen. Esta clase tiene métodos para enviar distintos tipos de peticiones http, como `get`, `post` y `delete`, y devuelven la respuesta del servidor.

3.3. Arquitectura

El programa consiste de una clase principal `Main` que se encarga de ejecutar el programa y cargar cada una de las capas de éste como podemos ver en el código 3. Las capas que agregamos son las siguientes:

- `cameraPreview`: Previsualización de la cámara -línea 5 (véase 3.4.1)-.
- `arLayer`: Capa que contiene la información de la realidad aumentada, es decir, los PI -línea 6 (véase 3.4.2)-.
- `radar`: Capa que contiene la vista de radar de los PI que tenemos alrededor -línea 9 (véase 3.4.3)-.
- `verticalRangeSeekBar`: Contiene la barra de rango para ajustar la distancia de la que queremos que aparezcan los PI -línea 12 (véase 3.4.4)-.
- `AccuracyDisplay`: Muestra la precisión de la localización del dispositivo -línea 23 (véase 3.4.5)-.

```
/**
 * Clase para descargar los PI de las fuentes de información
 * en un hilo separado
 */
class DownloadPOIData extends AsyncTask<Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... voids) {
        double latitud = ubicacionActual.getLatitude();
        double longitud = ubicacionActual.getLongitude();
        FuentePI geo = new FuentePIGeonames();
        geo.obtenerPI(latitud, longitud);
    }

    @Override
    protected void onPostExecute(Void s) {
        //Aquí ponemos lo que queramos que se
        //realize después de que se ejecuta
        //la tarea en el fondo
    }
}
```

Código 2: Clase para obtener los PI de nuestra fuente de información en un hilo separado

```
/**
 * Inicializamos las capas del programa.
 */
private void initLayers() {
  setContentView(cameraPreview);
  addContentView(arLayer, new ViewGroup.LayoutParams(
      ViewGroup.LayoutParams.FILL_PARENT,
      ViewGroup.LayoutParams.FILL_PARENT));
  addContentView(new Radar(), new ViewGroup.LayoutParams(
      ViewGroup.LayoutParams.WRAP_CONTENT,
      ViewGroup.LayoutParams.WRAP_CONTENT));
  addContentView(new VerticalRangeSeekBar(),
```

Código 3: Inicialización de las diferentes capas que despliegan información en el programa (1/2)

```
13        new FrameLayout.LayoutParams(
14        FrameLayout.LayoutParams.WRAP_CONTENT,
15        FrameLayout.LayoutParams.WRAP_CONTENT,
16        Gravity.RIGHT | Gravity.CENTER_VERTICAL));
17    FrameLayout.LayoutParams accuracyDisplayLayoutParams =
18        new FrameLayout.LayoutParams(
19        FrameLayout.LayoutParams.WRAP_CONTENT,
20        FrameLayout.LayoutParams.WRAP_CONTENT,
21        Gravity.RIGHT | Gravity.TOP);
22    accuracyDisplayLayoutParams.setMargins(0,2,2,0);
23    addContentView(ARLayer.accuracyDisplay,
24        accuracyDisplayLayoutParams);
25  }
```

Código 3: (2/2)

3.4. Interfaz gráfica

3.4.1. Imagen de la cámara

La interfaz gráfica muestra la imagen de la cámara del dispositivo, la cual tiene una velocidad de 15 cuadros por segundo en ciertos teléfonos y 30 cuadros por segundo en otros. Ponemos esto como primer capa de la interfaz gráfica para después colocar todos los demás elementos sobre ella.

En el código 4 vemos cómo iniciamos la cámara y desplegamos la previsualización sobre un objeto de la clase `SurfaceView` [8]. Esta clase nos permite dibujar en ella muy rápidamente en un hilo de ejecución distinto al de la interfaz gráfica, de modo que la aplicación no necesita esperar a que la jerarquía de vistas esté lista para dibujar.

```
1  public class CameraPreview extends SurfaceView
2    implements SurfaceHolder.Callback {
3
4    SurfaceHolder surfaceHolder;
5    Camera camera;
```

Código 4: Implementación de la superficie donde se mostrará la previsualización de la cámara (1/2)

```
6     CameraPreview() {
7       super(Main.context);
8       //Agregamos un callback para que se  nos notifique
9       //cuando la supreficie sea creada o destruida
10      surfaceHolder = getHolder();
11      surfaceHolder.addCallback(this);
12      surfaceHolder.setType(
13        SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
14    }
15
16    /**
17      * Cuando se crea la superfice adquirimos la cámara
18      * y le decimos que dibuje sobre nuestra superficie.
19      */
20    public void surfaceCreated(SurfaceHolder holder) {
21      camera = Camera.open();
22      try {
23        camera.setPreviewDisplay(holder);
24      } catch (IOException exception) {
25        camera.release();
26        camera = null;
27      }
28    }
29
30    /**
31      * Cuando se destruya nuestra superfice debemos liberar
32      * la cámara pues es un recurso compartido.
33      */
34    public void surfaceDestroyed(SurfaceHolder holder) {
35      if (camera != null) {
36        camera.stopPreview();
37        camera.release();
38        camera = null;
39      }
40    }
41  }
```

Código 4: (2/2)

También calculamos el tamaño idóneo de la previsualización de la cámara basado en sus especificaciones y en el tamaño de la pantalla.

3.4.2. Puntos de interés

La parte central del programa son los PI que estén alrededor del usuario. Éstos se mostrarán en la pantalla sobre el objeto real visto a través de la cámara del dispositivo cuando aquel sea visible desde el punto donde se ubica el usuario. Se marcarán con un disco de color rojo con el nombre del PI debajo de él como podemos ver en la figura 3.1.

Indicar los PI es útil, pero la verdadera utilidad del programa viene en poder obtener más información sobre el PI que nos interese. Para esto utilizamos una ventana adicional de información que mostraremos cuando el usuario toque algunos de los PI en la pantalla. Entonces se deslizará hacia arriba una ventana mostrando el nombre y distancia hacia el PI y cargará información al respecto; está información será contenido `HTML` cargado generalmente desde una página web, aunque también puede estar guardado este contenido de forma local, dependiendo de la fuente de información que se esté utilizando. En la figura 3.3 podemos ver la versión para dispositivos móviles de un artículo de la wikipedia con información sobre la Catedral Metropolitana de la Ciudad de México, así como la distancia a la que se encuentra del dispositivo. Cuando acabemos de consultar la información, basta con tocar la `'X'` en la esquina superior derecha para que se cierre la ventana y volvamos a la vista normal de la aplicación.

Como vimos en la sección 3.2, estando en el centro histórico tenemos 45 PI a `1km` de nosotros; esto puede resultar en que se amontonen los PI en la pantalla y no podamos distinguirlos ni tampoco seleccionarlos. Para evitar este problema haremos una detección de colisiones para evitar que dos puntos se traslapen. En la sección 6.2 veremos cómo se resuelve este problema.

3.4.3. Radar

Al ser los PI la parte central de la aplicación, necesitamos una forma de identificar rápidamente dónde hay varios de ellos, o dónde no los hay, sin tener que girar el dispositivo 360°. Para esto se incluye una especie de radar en el que se muestran todos los PI alrededor del usuario en el rango seleccionado.

En la figura 3.4 se ve cómo el programa muestra los PI alrededor del usuario marcándolos con puntos rojos. Estos se encuentran alejados del centro

Figura 3.3: Ventana de información adicional de un PI mostrando un artículo de la wikipedia y la distancia a la Catedral Metropolitana de la Ciudad de México

según su distancia. Al dibujarlos en el radar, utilizamos una escala proporcional a la distancia a la que se encuentran del dispositivo, para que los puntos que se encuentren muy cerca no se aglomeren en el centro de la imagen y se distinga más fácilmente la distancia entre ellos. Así, para dibujar los puntos del radar la distancia a la que se encuentran del centro se multiplica por la escala obtenida en:

$$\text{escala} = \frac{\text{radio}}{\text{rango}}$$

Los elementos que sean visibles en la dirección en la que se encuentra el dispositivo se encuentran en el área sombreada. También se muestra en la parte superior la dirección en grados a la que está apuntando el dispositivo. Tanto los PI como los grados cambian a una frecuencia de alrededor de 30 cuadros por segundo, conforme el usuario mueve el dispositivo a una nueva dirección.

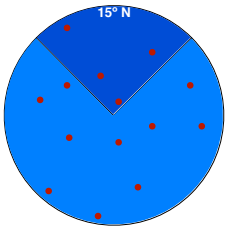


Figura 3.4: Vista del radar que mostrará los PI alrededor del dispositivo en el rango seleccionado

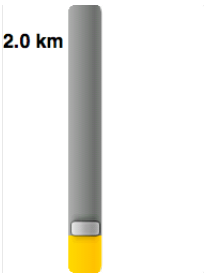
3.4.4. Selección de rango

Para mostrar los PI alrededor del dispositivo necesitamos establecer un rango, puesto que tenemos capacidad de memoria y almacenamiento limitada, y también porque el usuario puede querer limitar la información sólo a lo que esté más cerca de él. Así, cada vez que ejecutemos el programa debemos realizar una consulta a los proveedores de datos utilizando tanto la ubicación del dispositivo como un rango para limitar los PI que devolverá.

Especificar el rango también es útil para delimitar la cantidad de PI que vemos, ya que en algunos lugares puede haber una gran cantidad de ellos y el rango que normalmente usamos puede ser muy grande. También quizá estemos en cierto lugar donde podamos ver PI a una gran distancia y querremos aumentar el rango en el que se despliegan los mismos.

Para esto utilizamos una barra de desplazamiento vertical colocada al lado derecho de la pantalla. Lamentablemente Android sólo proporciona barras de desplazamiento horizontales, por lo que fue necesario implementar una nueva clase que proporcionara esta funcionalidad.

El indicador de la barra puede ser desplazado con el dedo y nos muestra el rango seleccionado, como podemos ver en la figura 3.5. La distancia que movemos el indicador es proporcional de manera cuadrática al rango seleccionado, similar a lo que sucede en el radar, para que cuando estemos en el área inferior podamos seleccionar con mayor precisión un rango pequeño y al estar en la parte superior no hace falta tener mucha precisión para seleccionar un rango amplio. La barra de selección devuelve un valor de 0 a `MAX`, que en este caso especificamos como 1000, dependiendo de la posición en la



(a) Seleccionamos 2.0 km

(b) Seleccionamos 80.1 km

Figura 3.5: Barra de selección de rango

que se encuentre el indicador. Para obtener un rango de 0 a 100`km` usamos la fórmula:

$$\text{rango} = \frac{\text{posición}^2}{10},$$

con lo que el rango va de $\frac{0^2}{10} = 0$ `m` hasta $\frac{1000^2}{10} = 100,000$ `m`.

3.4.5. Indicador de precisión de GPS

Al utilizar un GPS para determinar la ubicación del dispositivo estamos sujetos a distintos problemas ya que la señal no siempre es ideal y estos dispositivos no funcionan bien en interiores. Lo ideal para un dispositivo GPS es estar al aire libre, pero aun así la precisión de nuestras mediciones puede variar ampliamente.

Los chips GPS usados actualmente en los dispositivos móviles tienen una precisión de 6m el 95 % del tiempo, o mejor cuando se usan al aire libre y hay buena recepción de satélites GPS [27]. En las pruebas realizadas al aire libre con un teléfono Droid Eris[6] esto probó ser cierto, teniendo una buena precisión la mayoría del tiempo, marcando casi siempre una precisión de 2`m`.

Aún así, 6m pueden hacer que los PI que dibujamos en la pantalla no se muestren donde deben, particularmente los que están muy cerca del dispositivo; es por eso que el programa muestra al usuario en la esquina superior derecha de la pantalla, cuál es la precisión actual de las mediciones realizadas por el GPS, con lo que el usuario sabe que si tiene una precisión muy baja, la información que se muestre no será precisa.

[6]Teléfono marca HTC con Android 2.1. Procesador Qualcomm® MSM7600™ de 528MHz. Memoria RAM 288MB. http://www.htc.com/us/products/droid-eris-verizon.

Para adquirir la ubicación por medio del GPS, Android primero obtiene una medición muy vaga y la va refinando poco a poco hasta obtener una más precisa. Esto se hace porque obtener una medición precisa puede tardar hasta 1 minuto, que es demasiado para un usuario que está esperando a la aplicación. Entonces, con la medición poco precisa se pueden empezar a realizar cálculos y mostrar información, sin tener que esperar a que se tenga la precisión máxima. Así, cuando el programa se inicia es probable que tengamos poca precisión, pero irá mejorando poco a poco. Generalmente, en el transcurso de 20 segundos obtendremos una precisión aceptable en nuestra ubicación. Teniendo una precisión de 200 metros, ya podemos hacer una petición de los PI que tengamos a nuestro alrededor para guardarlos de manera local y ajustar su posición en la pantalla conforme se refine nuestra ubicación.

En el código 5 vemos cómo el escucha de cambio de ubicación cambia la precisión del indicador del GPS. Esto se hace cada vez que cambie la ubicación, lo que ocurre conforme mejore la precisión. También cada vez que cambie nuestra ubicación se afectará la posición de los PI que veamos en la pantalla, ya sea que cambie por aumento de precisión o porque nos movamos de lugar.

```
1 private final LocationListener locationListener;
2 locationListener = new LocationListener() {
3   public void onLocationChanged(Location location) {
4     currentLocation = location;
5     locationChanged = true;
6     //Cambiamos el indicador de precición de GPS
7     accuracyDisplay.changeAccuracy(location.getAccuracy());
8   }
9 }
```

Código 5: Escucha para el cambio de ubicación

3.4.6. Preferencias de la aplicación

Uno de los objetivos de la aplicación es que sea de muy fácil uso, por lo que, nos limitaremos a tener preferencias para seleccionar las distintas fuentes de información de las cuáles queremos que se muestren PI. Solo hay que presionar el botón de “Menu” del dispositivo y después seleccionar preferencias. Éstas se mostrarán como en la figura 3.6.

Figura 3.6: Pantalla de preferencias de la aplicación

3.4.7. Activación de GPS

Muchos teléfonos celulares actuales incluyen un sensor GPS, pero también tienen otras formas de calcular su ubicación. Una es a través de las torres con las que obtienen su señal telefónica, al realizar una triangulación y pueden calcular su posición con una precisión muy variable. La segunda es a través de redes WiFi. Existen compañías como Skyhook [5] que se dedican a rastrear en redes WiFi en el mapa. Así, con la información de una red que esté cerca del dispositivo se puede consultar esta base de datos para obtener una ubicación aproximada. Ninguno de estos dos métodos nos da la precisión que necesitamos, por lo que para que el programa funcione correctamente, el dispositivo debe tener un GPS. Estos sensores utilizan una gran cantidad de batería, por lo cuál muchos usuarios los mantienen apagados mientras no los usan; entonces, el programa al iniciarse realiza una comprobación del estado del GPS y en dado caso de estar desactivado le pide al usuario activarlo y lo lleva a la pantalla de configuración del dispositivo donde puede encenderse. Para esto utilizamos un diálogo nativo de Android que podemos ver en la figura 3.7.

Figura 3.7: Diálogo para solicitar al usuario que habilite el GPS de su dispositivo

Capítulo 4

Sensores

Los teléfonos móviles modernos son mucho más que simples dispositivos de comunicación con conexión a internet. Con la proliferación de sistemas microelectromecánicos, llamados MEMS[1], más pequeños y que utilizan menos energía, los teléfonos móviles cada vez van agregando más sensores, que a su vez se vuelven más precisos.

La mayoría de los teléfonos con Android tiene varios sensores, siendo los más comunes: el acelerómetro multieje, el magnetómetro y el dispositivo GPS. En la tabla 4.1 podemos ver todos los sensores que soporta actualmente Android.

De cada sensor se pueden obtener varios atributos, entre los que están:

- Nombre del sensor
- Consumo de corriente en mA
- Resolución
- Rango máximo
- Fabricante
- Versión

El sensor de orientación de un Nexus One[2] muestra las siguientes características [25]:

- Nombre: AK8973 Orientation Sensor
- Consumo de corriente: 7.0 mA
- Resolución: 1.0 grado

[1]Siglas del inglés Micro-Electro-Mechanical Systems.

[2]Teléfono marca HTC con Android 2.3. Procesador Qualcomm QSD 8250 Snapdragon de 1 GHz. Memoria RAM 512MB. http://www.google.com/phone/detail/nexus-one.

Tipo de Sensor	Descripción
TYPE_ACCELEROMETER	Acelerómetro de tres ejes que mide la aceleración en $\frac{m}{s^2}$.
TYPE_GYROSCOPE	Mide la orientación basándose en un momento angular en tres ejes, lo mide en grados.
TYPE_LIGHT	Mide la luz del ambiente en lux.
TYPE_MAGNETIC_FIELD	Mide el campo magnético en microteslas μT
TYPE_ORIENTATION	Mide la orientación del dispositivo en tres ejes.
TYPE_PRESSURE	Mide la presión del aire en kilopascales.
TYPE_PROXIMITY	Mide la distancia entre el dispositivo y un objeto en centímetros.
TYPE_TEMPERATURE	Mide la temperatura en grados Celsius.

Tabla 4.1: Sensores disponibles en el SDK de Android

- Rango máximo: 360 grados

Para poder saber hacia dónde está apuntando nuestro teléfono utilizaremos dos sensores, el magnetómetro y el acelerómetro multieje. Con el magnetómetro sabremos la dirección de nuestro dispositivo con respecto al norte magnético, sabiendo entonces a dónde apuntamos; el acelerómetro multieje lo utilizaremos para obtener la inclinación de nuestro dispositivo con respecto a un eje paralelo al suelo. De este modo tendremos dos ángulos de rotación, como podemos ver en la figura 4.1. Sólo son necesarios dos ejes puesto que la aplicación está diseñada para utilizarse con el dispositivo de forma horizontal con lo que garantizamos que nuestro tercer ángulo sea fijo.

4.1. Acelerómetro

4.1.1. Funcionamiento

Los acelerómetros utilizados en los dispositivos móviles actuales generalmente funcionan gracias a la piezoelectricidad. Esto es un fenómeno que se presenta en ciertos materiales sólidos, más comúnmente en cristales, que al ser sometidos a tensiones mecánicas exhiben ciertas cargas eléctricas. Así,

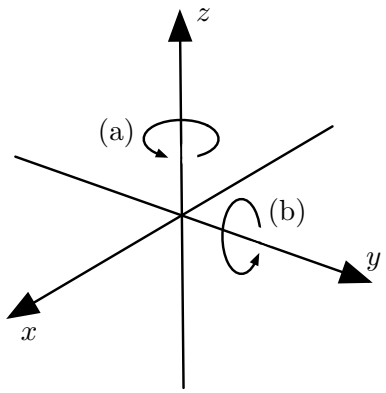


Figura 4.1: Ejes de rotación. (a) Eje obtenido por el sensor magnético. (b) Eje obtenido por el acelerómetro

con sensores piezoeléctricos en cada uno de los tres ejes del dispositivo, podemos medir su aceleración en el espacio.

Imaginemos el acelerómetro como una caja con una esfera en el centro, como en la figura 4.2, que está en un lugar aislado de cualquier fuerza, incluida la de gravedad. La esfera entonces flota estática en el centro de la caja. Definamos tres ejes: x horizontal, y perpendicular a la superficie de la página y z vertical. La esfera entonces podría desplazarse hacia la derecha $x+$, hacia la izquierda $x-$, hacia arriba $z+$, hacia abajo $z-$, fuera de la página hacia el lector $y+$ o hacia atrás $y-$ o combinando estos movimientos elementales de manera arbitraria.

Si movemos rápidamente la caja a la izquierda, sometiéndola a una aceleración de, digamos, $1g = 9.81\frac{m}{s^2}$, entonces la esfera se desplazará a la pared $x-$ como se puede ver en la figura 4.3. Si medimos la presión que la esfera ejerce sobre esta pared, obtendremos entonces el valor $-1g$ en el eje x justamente.

Notemos que el acelerómetro detecta una fuerza en la dirección opuesta al vector de aceleración, a la que llamamos comúnmente fuerza ficticia [16]. Así, el acelerómetro mide la aceleración indirectamente a través de la fuerza que es aplicada a una de sus paredes, en los acelerómetros reales esto es un sensor piezoeléctrico o un resorte.

Ahora, si tomamos nuestro modelo y lo ponemos en la tierra, entonces la esfera caerá en la pared $z-$ y aplicará una fuerza de $1g$ a la pared de abajo como vemos en la figura 4.4. Aquí, la caja no se está moviendo pero de igual manera tenemos una lectura de $-1g$ en el eje z. Entonces, incluso cuando el

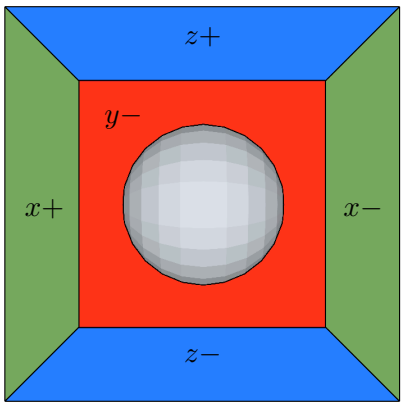


Figura 4.2: Inmóvil sin gravedad.
$X = 0g$, $Y = 0g$, $Z = 0g$

teléfono esté en reposo, tendremos esta lectura por parte del sensor.

La utilidad real de estos acelerómetros es cuando utilizamos más de un eje: si rotamos nuestra caja 45° a la derecha, entonces la esfera tocará dos paredes, $z-$ y $x-$ como podemos ver en la figura 4.5, donde tenemos que el vector gravedad se descompone en $\sqrt{\frac{1}{2}}$ para cada componente debido al ángulo de rotación de la caja. Esto lo podemos ver también como un vector de aceleración, el cual descomponemos en las proyecciones de cada eje como vemos en la figura 4.6; esos son los valores que nos dará el acelerómetro del dispositivo.

4.1.2. Uso

En el código 6 (página 41) podemos ver cómo utilizamos el manejador de sensores para inicializar el sensor de orientación, que es el sensor magnético, y el acelerómetro. En ambos casos registramos un escucha para estos sensores que se encargará de hacer los cálculos necesarios cada vez que haya una actualización en los valores de estos sensores. Para determinar la velocidad con la que se realizará la actualización de los sensores debemos proporcionar uno de tres valores posibles que en este caso fue `SensorManager.SENSOR_DELAY_GAME`, que es el valor intermedio. Utilizando un Droid Eris, que es un teléfono de gama media baja, los sensores nos dan 32 actualizaciones por segundo en promedio, siendo esto suficiente para mantener nuestros 30Hz y que la interfaz de usuario se vea bien.

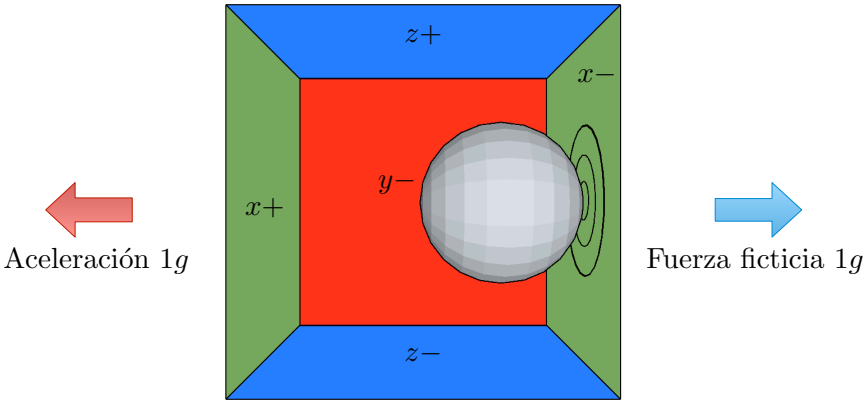


Figura 4.3: Aceleración de $1g$ a la izquierda.
$X = 1g$, $Y = 0g$, $Z = 0g$

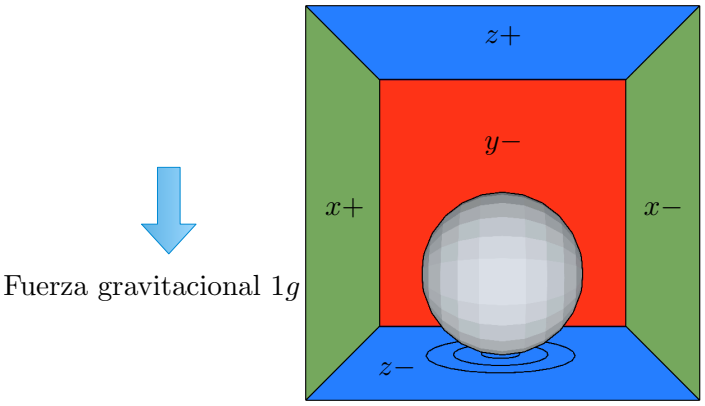


Figura 4.4: Inmóvil con gravedad, fuerza gravitacional $1g$.
$X = 0g$, $Y = 0g$, $Z = -1g$

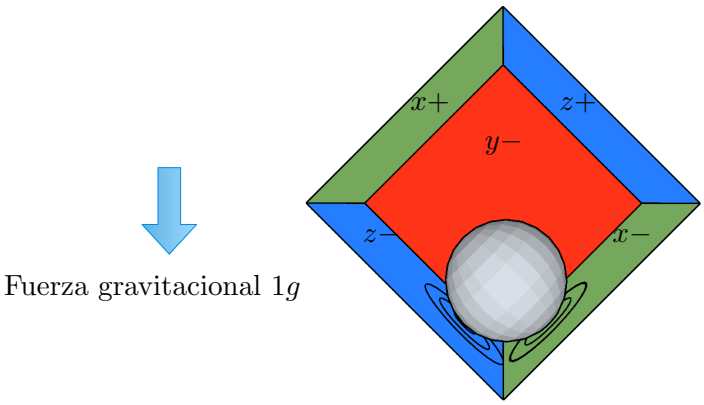


Figura 4.5: Inmóvil con gravedad, caja rotada 45°. Fuerza gravitacional $1g$. $X = -\sqrt{\frac{1}{2}}g$, $Y = 0g$, $Z = -\sqrt{\frac{1}{2}}g$

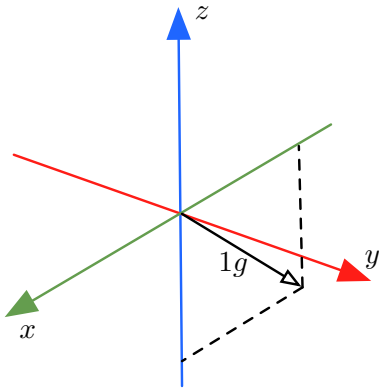


Figura 4.6: Vector de aceleración con sus proyecciones sobre el eje x e y

```
1  /**
2   * Iniciamos los sensores, de orientación y el acelerómetro.
3   */
4  private void initSensors() {
5    sensorManager = (SensorManager) Main.context
6      .getSystemService(Context.SENSOR_SERVICE);
7    sensorManager.registerListener(orientationListener,
8      sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
9      SensorManager.SENSOR_DELAY_GAME);
10   sensorManager.registerListener(orientationListener,
11     sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
12     SensorManager.SENSOR_DELAY_GAME);
```

Código 6: Inicialización de los sensores del dispositivo

Cada vez que el dispositivo detecte un cambio en alguno de los sensores llamará al escucha indicado, pasando un objeto `Event`, del cual extraeremos la información de los cambios en el sensor. En el código 7 podemos ver cómo a partir de `Event` es posible extraer el tipo de sensor que llamó al escucha (líneas 6 y 13), así como los valores del sensor (líneas 7 y 15).

```
float[] valoresAcelerometro;
float[] valoresMagnetometro;
final SensorEventListener orientationListener
 = new SensorEventListener() {

  public void onSensorChanged(SensorEvent event) {
  if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
   valoresMagnetometro = event.values;
   direccion = SensorOptimalFilter.filtrarDireccion(
         calcularDireccion());
  }

  if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
   inclinacion = SensorOptimalFilter.filtrarInclinacion(
          calcularInclinacion(
   event.values[0], event.values[2]));
   valoresAcelerometro = event.values;
  }

  if (cambioLaUbicacion) {
    actualizarLayoutPI(currentLocation);
    cambioLaUbicacion = false;
  } else {
    actualizarLayoutPI(null);
  }
  //Ponemos la dirección del dispositivo en el radar
  Radar.setDireccion(direccion);
  //Con esta llamada se redibujan todos los elementos
  //de la pantalla
  postInvalidate();
};
```

Código 7: Escucha para cambios en el acelerómetro y el sensor de orientación

El acelerómetro nos devuelve un arreglo con tres valores, los cuáles vemos en la figura 4.7:

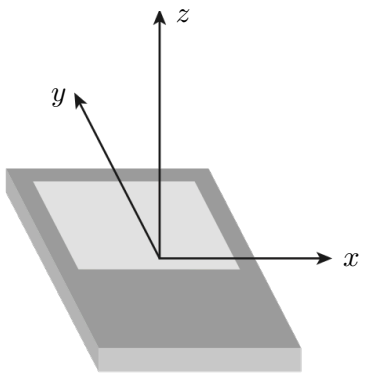


Figura 4.7: Ejes del dispositivo [28]

1. **eje x (lateral)**: Aceleración hacia los lados (izquierda o derecha); los valores positivos representan movimientos hacia el lado derecho del dispositivo y los valores negativos indican movimiento hacia la izquierda.

2. **eje y (longitudinal)**: Aceleración hacia adelante o hacia atrás; la aceleración hacia adelante es un valor positivo.

3. **eje z (lateral)**: Aceleración hacia arriba y hacia abajo; los valores positivos representan movimiento hacia arriba, como cuando levantamos el dispositivo. Estando en reposo, como en la figura 4.7, este eje es el que registrará la aceleración de la gravedad como $-9.81\frac{m}{s^2}$.

La aplicación está diseñada para funcionar con el dispositivo en posición horizontal, es decir, con su lado más largo alineado con el suelo para poder mostrar mayor cantidad de PI ya que en general se encontrarán a la derecha o la izquierda de otro PI y no arriba y abajo. Con esta limitante, sólo necesitamos los valores que nos dará el acelerómetro para x y z, ya que la y será fija.

De este modo, para calcular la inclinación que tiene el dispositivo simplemente calculamos el ángulo del vector formado por los valores del acelerómetro en el eje x y z respecto al eje y. Esto lo calculamos con la fórmula:

$$\text{inclinación} = \arctan\left(\frac{x}{z}\right)$$

Verificamos primero que z no sea cero, y calculamos la inclinación según el algoritmo 1. En caso de que z sea 0 entonces sólo utilizamos el eje x para calcular la inclinación, la cual será $\frac{\pi}{2}$ en caso de que x sea negativa y $\frac{3\pi}{2}$ en caso de ser positiva. Después convertimos nuestro resultado a grados y por último compensamos la posición del dispositivo ya que éste se encuentra girado 90°; así sumamos o restamos 90 según sea necesario.

Algoritmo 1 Cálculo de la inclinación del dispositivo

```
if z ≠ 0 then
  inclinación ← arctan(x/z)
else if x < 0 then
  inclinación ← π/2
else if x ≥ 0 then
  inclinación ← 3π/2
end if

inclinación ← inclinación 360/2π {convertimos a grados}

if inclinación < 0 then {Compensamos la posición del dispositivo}
  inclinación ← inclinación + 90
else
  inclinación ← inclinación − 90
end if
```

4.2. Sensor magnético

4.2.1. Funcionamiento

Un magnetómetro es un dispositivo que sirve para medir la fuerza y/o dirección del campo magnético en una vecindad del instrumento. Hay varios tipos de magnetómetros que funcionan de distintas maneras. En los teléfonos y dispositivos móviles se utiliza un magnetómetro de estado sólido de Efecto Hall [29] que es un sensor construido completamente con materiales sólidos que reacciona al Efecto Hall. El Efecto Hall es la producción de una diferencia de voltaje en un conductor eléctrico, transversal a la corriente eléctrica en el conductor y un campo magnético perpendicular a la corriente.

Este fenómeno fue descubierto por Edwin Hall en 1897 [10] y permite medir diferentes campos magnéticos alrededor del sensor, entre ellos el campo magnético de la tierra.

El magnetómetro que se incluye en estos dispositivos, al igual que el acelerómetro, es de tres ejes, dándonos un vector con la magnitud del campo magnético de la tierra relativo a la posición en que se encuentra el teléfono.

4.2.2. Uso

Android incluye un sensor de orientación llamado `TYPE_ORIENTATION`, como vimos en la tabla 4.1, el cuál nos da muy fácilmente la dirección a la que está apuntando el dispositivo en grados respecto al norte magnético. Sin embargo, a partir de la versión 2.3 este sensor ha sido marcado como obsoleto *(deprecated)* [9] y se recomienda calcular esta orientación utilizando el sensor magnético, el acelerómetro y una matriz de rotación. En realidad esto es lo que hace Android cuando usamos el sensor de orientación, pero dado que en realidad no existe este sensor en el teléfono, se ha optado por no usarlo directamente. Además hay una ventaja al calcular de esta manera la orientación, y es que así podemos cambiar el sistema de referencia de la orientación para reasignar los ejes X, Y y Z para que se ajusten a la posición en que estará el dispositivo al usarlo con la aplicación.

Si usamos el sistema de referencia estándar, la orientación del dispositivo se da en tres dimensiones como muestra la figura 4.8 [28].

Para calcular la orientación del dispositivo utilizaremos el sensor magnético y el acelerómetro, así que guardamos los datos de cada uno de estos sensores en un arreglo `valoresAcelerometro` y `valoresMagnetometro` como podemos ver en los códigos 7 y 8. Después calculamos la matriz de rotación con el método `getRotationMatrix` (línea 6, código 7) que transforma un vector del sistema de coordenadas del dispositivo al sistema de coordenadas de la tierra que podemos ver en la figura 4.9; el método utiliza los datos del acelerómetro para calcular en qué posición se encuentra el dispositivo; tiene los siguientes parámetros:

- `R` Arreglo de nueve números de tipo `float` que contienen la matriz de rotación `R` cuando este método regresa. `R` es la matriz identidad cuando el dispositivo está alineado con el sistema de coordenadas de la tierra.

- `I` Arreglo de nueve números de tipo `float` que contiene la matriz de rotación `I` cuando el método regresa. `I` es la matriz de rotación que

transforma el vector dado por los valores del magnetómetro en el mismo sistema de coordenadas que la gravedad (el sistema de coordenadas de la tierra).

- `gravedad` Arreglo de tres números de tipo `float` que contiene el vector de gravedad expresado en las coordenadas del dispositivo, que se obtiene de los valores que devuelve el sensor de tipo `TYPE_ACCELEROMETER`.
- `geomagnetico` Arreglo de tres números de tipo `float` que contiene el vector de cantidades magnéticas expresado en las coordenadas del dispositivo, que se obtiene de los valores que devuelve el sensor de tipo `TYPE_MAGNETIC_FIELD`.

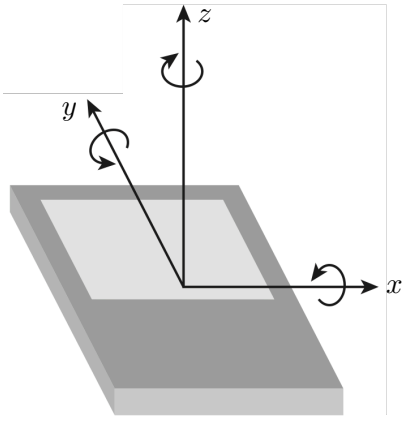


Figura 4.8: Ejes de orientación del dispositivo

Luego reasignamos los ejes de coordenadas para que correspondan a la posición del dispositivo. Para esto usamos el método `remapCoordinateSystem` (código 8), el cual rota una matriz para que esté expresada en un sistema de coordenadas distinto, recibe los siguientes parámetros:

- `entradaR`: La matriz de rotación a ser transformada.
- `X`: Define en cuál de los ejes coordenados de la tierra se asignará el eje X del dispositivo.
- `Y`: Define en cuál de los ejes coordenados de la tierra se asignará el eje Y del dispositivo.

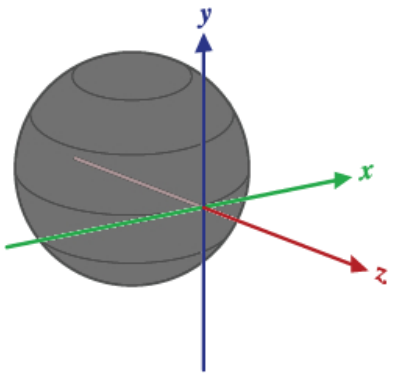


Figura 4.9: Ejes de coordenadas del mundo

- `salidaR`: La matriz de rotación transformada.

Notemos que como el sistema de coordenadas es ortonormal sólo necesitamos dar dos ejes para establecer el nuevo sistema de coordenadas. Para reasignar el sistema de coordenadas utilizaremos `AXIS_X` en el parámetro X y `AXIS_Y` en el parámetro Y (código 8, línea 8).

Por último, utilizamos el método `getOrientation` (código 8, linea 12), el cual nos da la orientación del dispositivo según la matriz de rotación que le proporcionemos; recibe los siguientes parámetros:

- `R` Matriz de rotación.
- `valores` Arreglo de tres números de tipo `float` que contiene el resultado.

Este método nos devuelve la orientación del dispositivo en sus tres ejes alineado con las coordenadas de la tierra cuando el dispositivo está en posición horizontal. En el primer índice del arreglo `valores`, se encuentra la orientación con respecto al eje Z de la figura 4.9, que es la dirección con respecto al norte. Esta dirección se da en radianes y la convertimos a grados (código 8, línea 15).

```
1  private float calcularDirecion() {
2    float[] valores = new float[3];
3    float[] matrizDeRotacion = new float[9];
4    SensorManager.getRotationMatrix(matrizDeRotacion, null,
5        valuesAccelerometer, valuesMagneticField);
6
7    float[] matrizDeRotacion2 = new float[9];
8    SensorManager.remapCoordinateSystem(matrizDeRotacion,
9        SensorManager.AXIS_X, SensorManager.AXIS_Z,
10       matrizDeRotacion2);
11
12   SensorManager.getOrientation(matrizDeRotacion2, valores);
13
14   //Convertimos de radianes a grados
15   direccion = (float) Math.toDegrees(valores[0]);
16   if (direccion < 0) {
17     direccion = 360 + direccion;
18   } else if (direccion >= 360) {
19     direccion = direccion - 360;
20   }
21   return direccion;
22 }
```

Código 8: Inicialización de los sensores del dispositivo

4.3. Filtrado de ruido

Un gran problema al utilizar los sensores del dispositivo es la cantidad de ruido que tienen sus mediciones. En el acelerómetro idealmente tendríamos una medición de $1g$ repartida en los ejes que afecta según el ángulo de inclinación del dispositivo; en el caso de la dirección, deberíamos poder calcular una dirección de 0° a 360° que cambiara al ritmo en que giramos el dispositivo. Lamentablemente esto no sucede así, ya sea porque el dispositivo está arriba de una alta montaña donde la gravedad varíe del $9.81\frac{m}{s^2}$ estándar, porque haya algún otro dispositivo cerca con un gran campo magnético (como una bocina), o porque la mano del usuario tiemble mucho haciendo que haya más fuerzas actuando sobre el dispositivo o incluso fallas en la medición

del sensor electrónico. Tanto el acelerómetro como el magnetómetro de estos dispositivos tienen una pequeña corrección de errores, pero no es suficiente y debemos implementar nuestra propia solución.

Se debe tomar en cuenta que las lecturas de los sensores son muy rápidas, de alrededor de 30Hz en cada sensor. Así que nuestra solución debe ser eficiente ya que 30 veces por segundo deberemos filtrar el ruido de cada sensor y además hacer todo el resto del procesamiento que requiere el programa, como el pedir datos de internet, dibujar los PI en la pantalla, calcular la posición del dispositivo, etc. Incluso colocando el dispositivo sobre una mesa plana, los datos del acelerómetro contienen ruido, como podemos ver en la gráfica 4.10 en la que mostramos los valores del acelerómetro en el eje x, donde tenemos una variación en los datos de hasta 0.27. Este tipo de ruido es mucho mayor cuando no estamos en condiciones ideales, así como cuando el dispositivo está en movimiento. Incluso tenemos ruido en el magnetómetro, -si vemos la gráfica 4.11 vemos que la dirección varía hasta por 4°-.

También pueden existir problemas de calibración con el acelerómetro ya sea por algún problema de tolerancia en la manufactura del sensor o incluso porque el usuario puede haber golpeado su teléfono algunas veces. De nuevo, viendo la gráfica 4.10, vemos que los valores están cerca de 0.92 para el eje x del acelerómetro cuando deberían ser cercanos a 0 por estar en una superficie plana.

Dependiendo del tipo de aplicación que se tenga, se requerirá un filtro distinto. Por ejemplo, para un juego querremos que el dispositivo reaccione ante cambios pequeños en los valores del acelerómetro. En otro tipo de aplicaciones pueden requerir que se sientan más "estables", donde los cambios pequeños son ignorados a menos que tiendan con el tiempo de un valor `A` a un valor `B`.

Para nuestra aplicación de realidad aumentada queremos una combinación de estos filtros. Debemos estabilizar los datos para que cosas como el pulso del usuario no hagan que se muevan demasiado los PI que se muestran en la pantalla. También debemos eliminar el ruido natural del sensor para que los PI no parezcan estar vibrando, pues afectaría gravemente la experiencia del usuario. Por otro lado, queremos que si el usuario gira hacia alguna dirección, la aplicación no tarde en reaccionar de tal forma que el usuario tenga que esperar a que se actualicen los valores de los sensores y se desplacen los PI hacia donde ahora está apuntando el dispositivo.

Veamos la gráfica 4.12. En 4.12(a) tenemos un dispositivo que va girando poco a poco en una dirección y el valor del acelerómetro en el eje x va aumentando. En 4.12(b) el usuario giró rápidamente el dispositivo y luego lo regresó. Utilizaremos estas gráficas de ejemplo para ver cómo funcionan

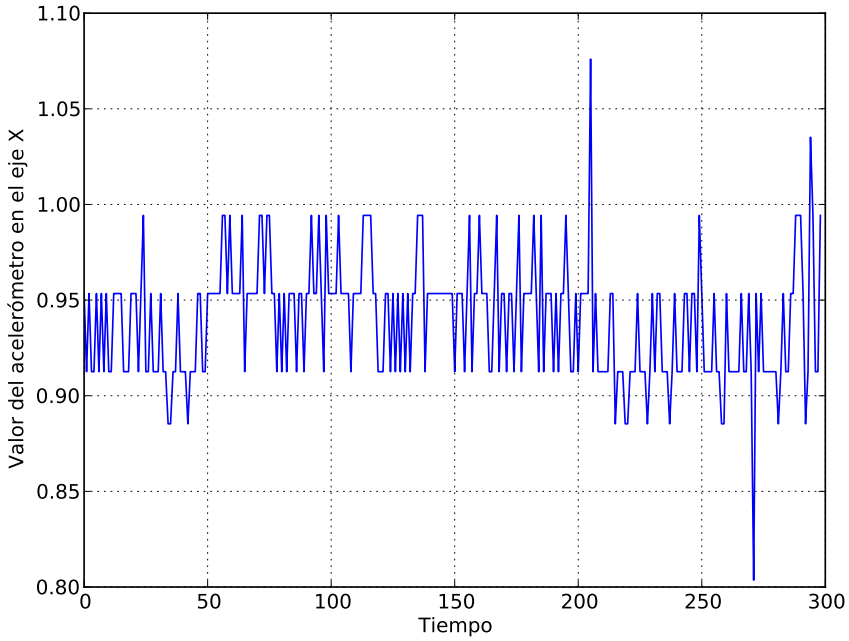


Figura 4.10: Valores del acelerómetro en el eje x mientras el dispositivo se encontraba en una mesa plana sin moverse

diferentes filtros para nuestros datos.

Para filtrar este tipo de ruido probaremos con tres diferentes filtros:

1. Promedio de datos
2. Filtro pasa bajos
3. Filtro "óptimo" (filtro pasa bajos con un umbral de cambio)

Otro de los filtros que vienen a la mente para usar en este tipo de situaciones es el filtro de Kalman[3], sin embargo hay algunos problemas para su implementación en esta aplicación en particular:

[3]El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960. Sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal, es decir, utilizar medidas que son observadas a través del tiempo y que contienen ruido y producir valores que tienen a ser más cercanos a los valores reales que los valores medidos.

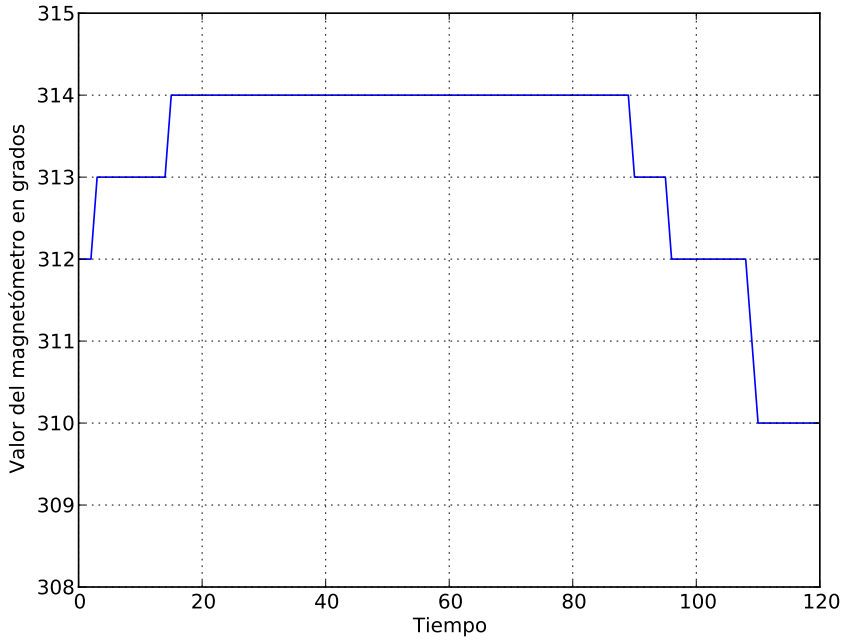
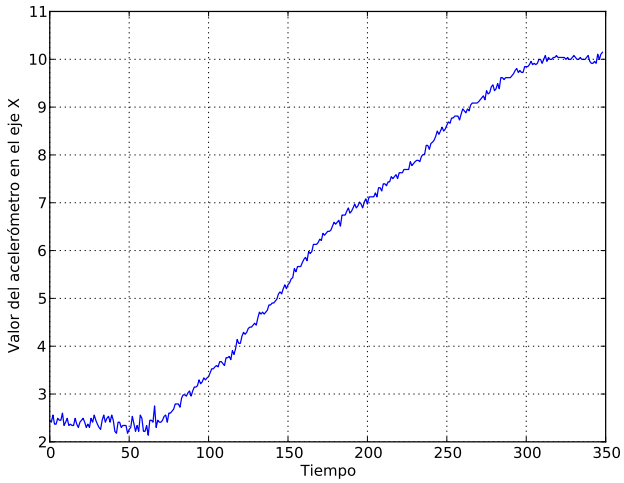


Figura 4.11: Valores de la dirección en grados mientras el dispositivo se encontraba en una mesa plana sin moverse

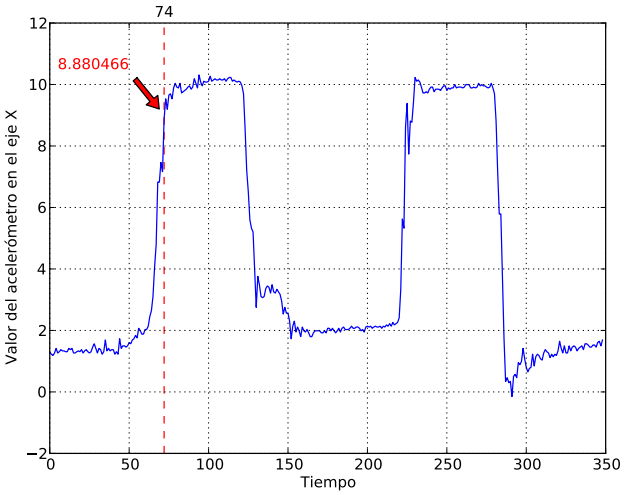
- El filtro de Kalman es eficiente, pero, al correr la aplicación en una máquina virtual que tiene hardware limitado, el desempeño no sería el óptimo.

- El filtro de Kalman supone que el ruido tiene una distribución normal (como una curva de campana). Sin embargo, el ruido en los sensores del dispositivo no tiene esta distribución. El movimiento del dispositivo, por ejemplo, cambia el tipo de ruido que vemos en el sensor.

- La eficiencia del filtro de Kalman se basa en que se programa de acuerdo a lo que uno espera obtener de los sensores, es decir, si esperamos que el dispositivo se mueva de manera circular en un determinado sentido entonces programamos el filtro de Kalman para esperar este tipo de movimiento. Pero para nuestra aplicación queremos medir movimientos arbitrarios del dispositivo.



(a) Valor del eje x del acelerómentro mientras se gira el dispositivo lentamente hacia arriba



(b) Valor del eje x del acelerómentro mientras se gira el dispositivo hacia arriba rápidamente, luego se gira hacia abajo, se espera un poco y se vuelve a girar hacia arriba rápidamente

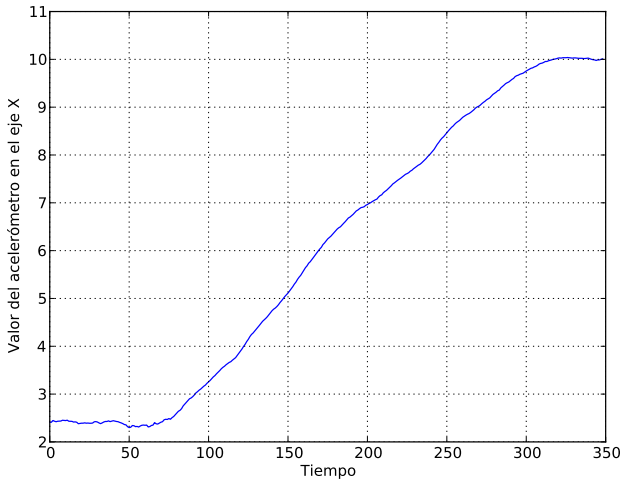
Figura 4.12: Datos sin filtrar del eje x del acelerómetro para dos tipos de movimientos distintos

Tiempo	Datos sin filtrar	Filtro de Promedios
67	3.1054392	2.22147873
68	4.0588636	2.42033579
69	4.7943625	2.70091496
70	6.8237944	3.19533360
71	6.8237944	3.68975224
72	7.4639506	4.23320412
73	7.164303	4.74669124
74	8.880466	5.42089838
75	9.534244	6.12915653
76	9.193735	6.78429527
77	9.656827	7.43943405
78	9.697687	8.00331639
79	9.534244	8.47730454
80	9.888372	8.7837623
81	10.038197	9.10520256

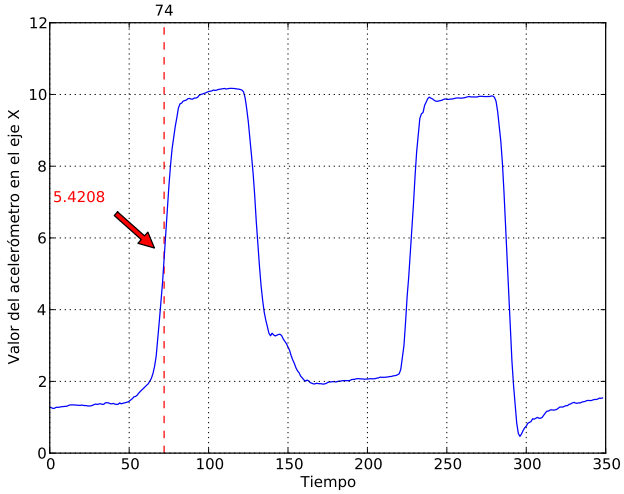
Tabla 4.2: Valores sin filtrar vs Filtro de promedios

4.3.1. Filtro de promedios

El primer filtro que analizamos es el de promedios. En la figura 4.13 graficamos los datos del promedio de las últimas diez muestras; vemos que los datos son mucho más suaves; sin embargo, en la figura 4.13(b) vemos que el cambio en la lectura se detecta mucho después que en los datos sin procesar, debido a este suavizado de los datos. Podemos ver que en el tiempo 74 la lectura da 5.4208, mientras que en la figura 4.12 tenemos que es 8.880466 (este retraso se puede ver con más detalle en la tabla 4.2). Haciendo varias mediciones, los valores del filtro de promedios tardan alrededor de 10 unidades de tiempo en igualar los valores sin filtrar. Esto representa un problema pues puede traducirse en un retraso en la actualización de la interfaz gráfica de la aplicación que pueda notar el usuario. Si tenemos 30 actualizaciones por segundo, representa un retraso de $\frac{1}{3}$segundo, que a pesar de no ser mucho, sí es notable en la interfaz gráfica. Probando este filtro en la aplicación se nota un evidente retraso en el movimiento de los PI al hacer movimientos rápidos del dispositivo. Se podría disminuir el número de muestras usadas en el promedio, pero entonces los datos no se suavizan lo suficiente.



(a) Valor del eje x del acelerómentro mientras se gira el dispositivo lentamente hacia arriba procesado con un filtro de promedios



(b) Valor del eje x del acelerómentro mientras se gira el dispositivo hacia arriba rápidamente, luego se gira hacia abajo, se espera un poco y se vuelve a girar hacia arriba rápidamente procesado con un filtro de promedios

Figura 4.13: Datos del eje x del acelerómetro promediados con las últimas 10 muestras para dos tipos de movimientos distintos

4.3.2. Filtro pasa bajos

Un filtro pasa bajos corresponde a un filtro caracterizado por permitir el paso de las frecuencias más bajas y atenuar las frecuencias más altas que una frecuencia de corte. Este tipo de filtros se utiliza para muchas cosas como suavizar datos, crear barreras acústicas, desenfocar imágenes y muchas más.

Lo que haremos con este filtro es darle a cada nueva lectura del sensor un peso y a la lectura anterior otro. La nueva lectura tiene un peso mucho menor que la anterior. Podemos describir el filtro con la siguiente formula:

$$S_n = S_{n-1} + \alpha(L_n - S_{n-1}),$$

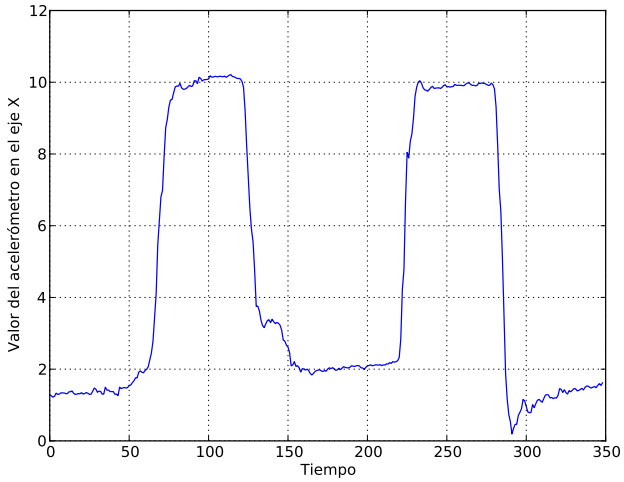
donde S es la salida *(valor filtrado)*, L es el valor de entrada del sensor *(valor sin filtrar)*, y α es un coeficiente con valor en $[0, 1]$. Si el coeficiente es 1, la salida es exactamente la misma que la entrada. Si el coeficiente es 0, la salida es siempre el número inicial. El caso que nos interesa es cuando α está entre estos valores. Entre más bajo sea el coeficiente, mayor será el suavizado.

Vemos en la figura 4.14 las gráficas del acelerómetro procesados por el filtro pasa bajos con $\alpha = 0.5$ y $\alpha = 0.15$. En la figura 4.14(a) vemos que a pesar de que la gráfica se suavizó un poco, aún presenta claras muestras de ruido. En la figura 4.14(b) vemos que el suavizado es muy bueno, pero presenta el mismo problema que el filtro de promedios, tiene un retraso en los cambios bruscos. De hecho el retraso es mayor que en el filtro de promedios. Haciendo varias mediciones, los valores en el filtro pasa bajos tardan alrededor de 15 unidades de tiempo en igualar a los valores sin filtrar, lo cual presenta el mismo problema de retraso en la interfaz gráfica.

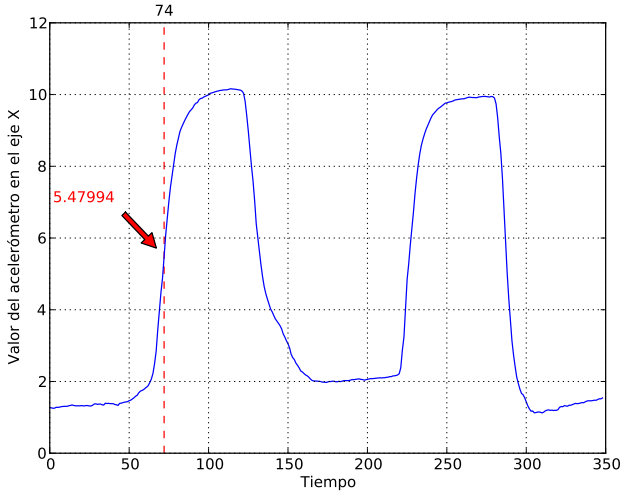
4.3.3. Filtro óptimo

De las observaciones del filtro de promedios y el filtro pasa bajos vemos que lo que necesitamos lograr es suavizar los datos que estén en cierto rango, pero poder tomar los cambios grandes inmediatamente. Lo llamaremos valor "óptimo". Para esto se verificará la diferencia entre la medición actual y la anterior y compararla con un umbral de cambio. Si la diferencia entre el valor promedio y el valor sin filtrar es mayor a nuestro umbral de cambio entonces utilizamos el valor sin filtrar; si no es así, utilizaremos el valor del filtro pasa bajos, como podemos ver en el algoritmo 2.

Con este algoritmo obtenemos una gráfica suficientemente suave que reacciona rápidamente a cambios grandes. La gráfica de la figura 4.15(a) es exactamente igual que la gráfica que obtendríamos utilizando únicamente



(a) $\alpha = 0.5$



(b) $\alpha = 0.15$

Figura 4.14: Datos del eje x del acelerómentro mientras se gira el dispositivo hacia arriba rápidamente, luego se gira hacia abajo, se espera un poco y se vuelve a girar hacia arriba rápidamente, procesado con un filtro pasa bajos

Algoritmo 2 Filtro óptimo

Entrada: nuevaMedicion, valorAnterior
  $\alpha \leftarrow 0.15$
  umbral $\leftarrow 0.5$
  if |nuevaMedicion − valorAnterior| ≤ umbral **then**
    nuevoValor ← (valorAnterior + α(nuevaMedicion − valorAnterior))
    {utilizamos el valor del filtro pasa bajos}
  else
    nuevoValor ← nuevaMedicion {valor sin filtrar}
  end if
  return nuevoValor

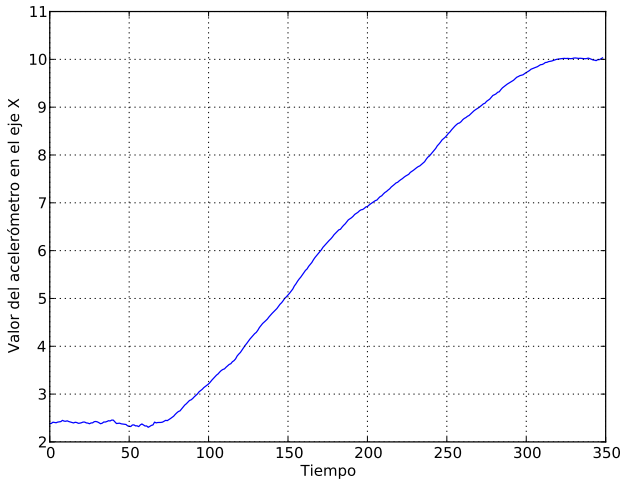
el filtro pasa bajos, pues al no haber ningún cambio brusco siempre se utiliza el valor de este filtro. Por otro lado, en la figura 4.15(b) podemos ver que la gráfica es suave y que reacciona de manera adecuada a cambios rápidos. En esos cambios la gráfica no es tan suave como quisiéramos, pero al ser un movimiento rápido del dispositivo y desplazarse rápidamente los PI en la pantalla, el usuario no nota esta "falta de suavidad" en ese momento. Vemos que cuando $t = 74$ el valor de la gráfica es 8.880466 al igual que en la gráfica 4.12(a).

Un punto importante en este algoritmo es la elección de un `umbral` correcto. Para el caso del valor del acelerómetro en el eje X utilizamos 0.5, pues como ya vimos, el acelerómetro arroja datos con una variación de hasta 0.27 estando estático; al moverse este ruido aumenta un poco. Además, los valores de x están en $[0, 10]$; así 0.5 es $\frac{1}{20}$ del giro posible, lo cual no representa un cambio grande. El valor del `umbral` cambiará dependiendo de qué estemos filtrando.

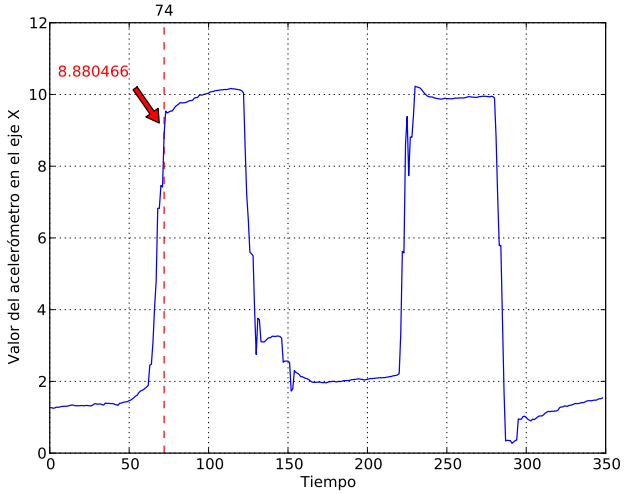
Ya que tenemos el filtro a usar debemos saber qué es lo que vamos a filtrar. Podríamos filtrar cada componente del acelerómetro y del magnetómetro y después calcular con estos datos la orientación y la inclinación, pero para ahorrar recursos y por simplicidad, filtraremos directamente los valores ya calculados:

1. **Dirección**, valores en $[0, 359]$
2. **Inclinación**, valores en $[-90, 90]$

En la dirección vimos que los datos varían hasta por cuatro unidades estando en reposo, así que utilizaremos 8 como `umbral`.



(a) Valor del eje x del acelerómentro mientras se gira el dispositivo lentamente hacia arriba procesado con el filtro óptimo



(b) Valor del eje x del acelerómentro mientras se gira el dispositivo hacia arriba rápidamente, luego se gira hacia abajo, se espera un poco y se vuelve a girar hacia arriba rápidamente procesado con el filtro óptimo

Figura 4.15: Datos del eje x del acelerómentro para dos movimientos distintos utilizando el filtro óptimo. Tenemos $\alpha = 0.15$ y umbral $= 0.5$

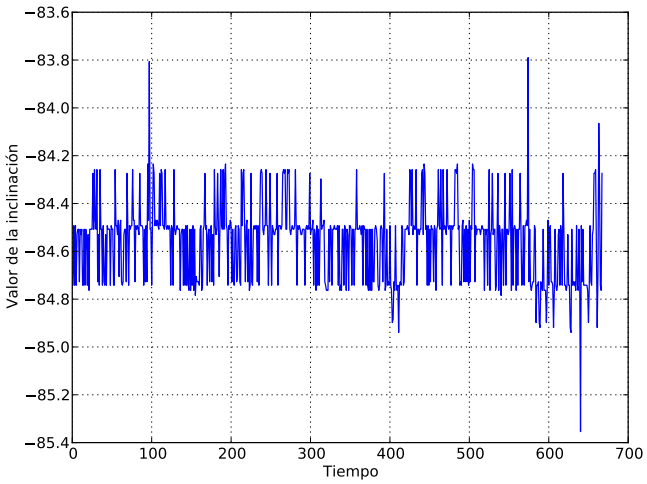


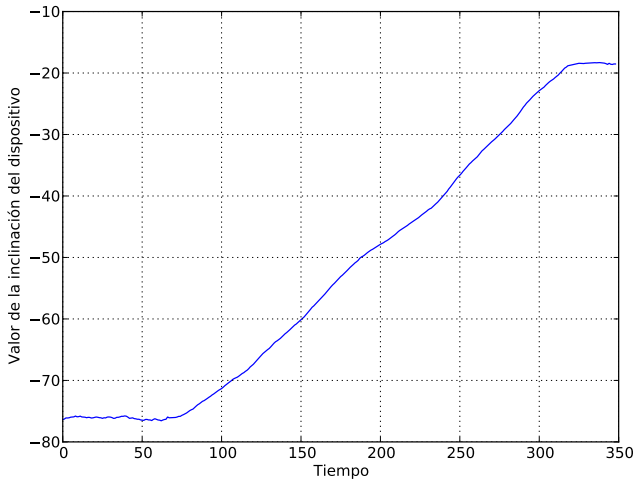
Figura 4.16: Valores de la inclinación mientras el dispositivo se encontraba en una mesa plana sin moverse

En la inclinación los datos varían hasta por 1.56417 estando en total reposo, como podemos ver en la gráfica 4.16, así que tomaremos 3 como `umbral`.

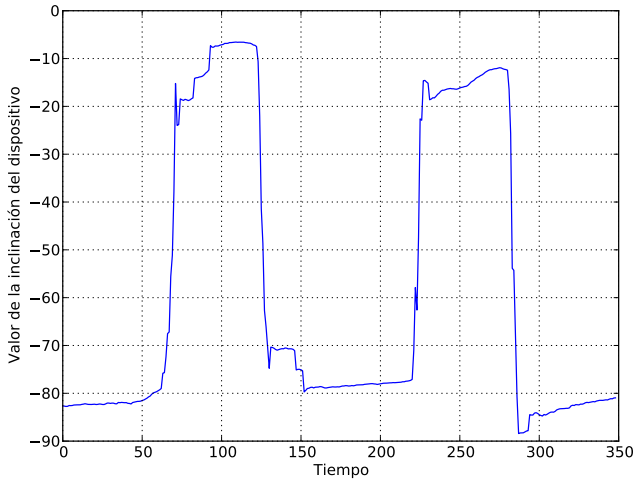
Utilizando estos umbrales, las gráficas tanto de inclinación como de dirección se suavizan considerablemente cuando hay un movimiento suave y cuando existen movimientos bruscos reaccionan rápidamente a los cambios. En la figura 4.17 vemos la gráfica de la inclinación procesada por el filtro óptimo tanto para un movimiento lento como para un giro brusco.

4.3.4. Filtrado de la dirección del dispositivo

Estamos filtrando dos valores, la inclinación, que está en $[-90, 90]$, y la dirección, que está en $[0, 359]$. La inclinación, para llegar de -90 a 90, debe pasar por todos los valores intermedios, pero la dirección no; si estamos en 359° y giramos un poco a la derecha, entonces pasaremos a 0°. Esto representa un problema para nuestro filtro, pues pensará que hubo un cambio muy grande y entonces no suavizará los datos, cuando en realidad el cambio fue solo de 1°. Para resolver esto, guardaremos el valor de la dirección en dos variables distintas, que usaremos dependiendo de la dirección en que apunta el dispositivo. La primer variable, `direccion`, la usaremos cuan-



(a) Valor de la inclinación del dispositivo mientras se gira el dispositivo lentamente hacia arriba procesado con el filtro óptimo



(b) Valor de la inclinación del dispositivo mientras se gira el dispositivo hacia arriba rápidamente, luego se gira hacia abajo, se espera un poco y se vuelve a girar hacia arriba rápidamente procesado con el filtro óptimo

Figura 4.17: Datos del eje x del acelerómentro para dos movimientos distintos utilizando el filtro óptimo. Tenemos $\alpha = 0.15$ y umbral $= 3$

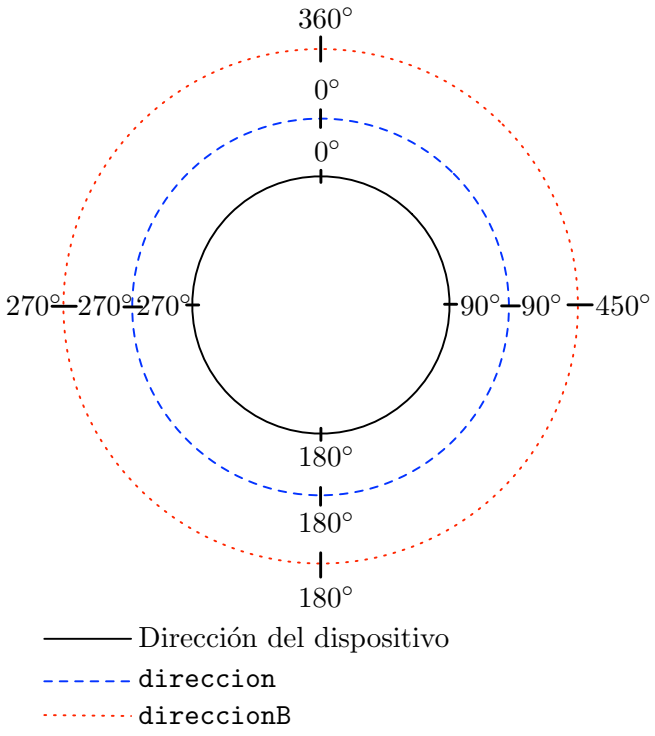


Figura 4.18: Valores de la dirección del dispositivo y las dos variables usadas para poder filtrar sin problemas

do el dispositivo esté en $[90°, 270°]$ y la segunda, `direccionB`, la usaremos cuando el dispositivo esté en $(270°, 90°)$. En `direccionB` no guardaremos los valores tal cual de la dirección, sino que les sumaremos 360 cuando la dirección esté en $[0°, 180°]$; de este modo, en el cruce de 359° a 0°, que es nuestro problema, el cambio será realmente de 359° a 360° y de este modo no habrá problemas con nuestro filtro. Lo único que tendremos que hacer es restar 360 antes de devolver nuestro valor de dirección. En la figura 4.18 podemos ver como funcionarían estas variables.

Capítulo 5

GPS

5.1. Funcionamiento

El GPS es un sistema de navegación satelital que fue desarrollado por el Departamento de Defensa de Estados Unidos a principios de 1970. Inicialmente se desarrolló sólo con propósitos militares, pero después se hizo disponible para el público en general.

El GPS provee información continua de posición y tiempo en cualquier parte del mundo sin importar las condiciones climatológicas. Al ser usado por un número ilimitado de usuarios, así como por razones de seguridad, es un sistema pasivo de un solo sentido, es decir, un dispositivo puede recibir las señales que manda un satélite de GPS, pero no puede enviarle información ni comunicarse con él de ninguna otra manera.

Originalmente, el sistema de GPS constaba de veinticuatro satélites, puestos en marcha en 1993. Estos satélites orbitan la tierra sobre seis ejes, con cuatro satélites en cada eje, como podemos ver en la figura 5.1. Gracias a esta distribución siempre pueden verse de cuatro a diez satélites desde cualquier parte del mundo [27]. Para 2008 ya había treinta y un satélites operando sobre estas mismas órbitas, por lo que ya no hay una distribución uniforme, pero los satélites adicionales aumentan la precisión de las mediciones proporcionando medidas redundantes.

5.1.1. Segmentos

El sistema de GPS consiste de tres segmentos, como se ve en la figura 5.2: el segmento del espacio, el segmento de control y el segmento de usuario. El segmento del espacio consiste de los treinta y un satélites antes mencionados.

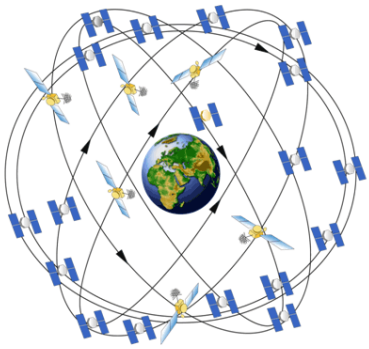


Figura 5.1: Distribución de los satélites GPS en seis órbitas sobre la tierra

Cada satélite transmite una señal, la cual incluye un mensaje de navegación con las coordenadas de el satélite como una función de tiempo. Las señales transmitidas están controladas por relojes atómicos muy precisos.

El segmento de control del GPS consiste de de una red mundial de estaciones de rastreo con una estación de control central localizada en Colorado, Estados Unidos. La tarea principal del segmento de control es rastrear los satélites GPS para poder determinar y predecir su ubicación y verificar que todos los sistemas del satélite estén funcionando debidamente. Las estaciones de control sí pueden comunicarse directamente con los satélites enviándoles información.

El segmento de usuario incluye todos los usuarios civiles y militares. Con un receptor GPS conectado a una antena GPS, un usuario puede recibir las señales GPS de los satélites que pueden ser usadas para determinar su posición en cualquier lugar del mundo. El GPS actualmente está disponible a todos los usuarios sin ningún costo directo.

5.1.2. Cálculo de posición

Para poder calcular su posición, un dispositivo receptor de GPS debe recibir la señal de tres satélites y determinar a qué distancia están de él, -figura 5.3(a)-. Esto se hace gracias a que los satélites incluyen en su transmisión el tiempo en el que se mandó la señal; al viajar estas señales a la velocidad de la luz c, se multiplica c por el tiempo que tardó en llegar la señal al receptor. El receptor sincroniza su reloj con el de los satélites para poder hacer este cálculo.

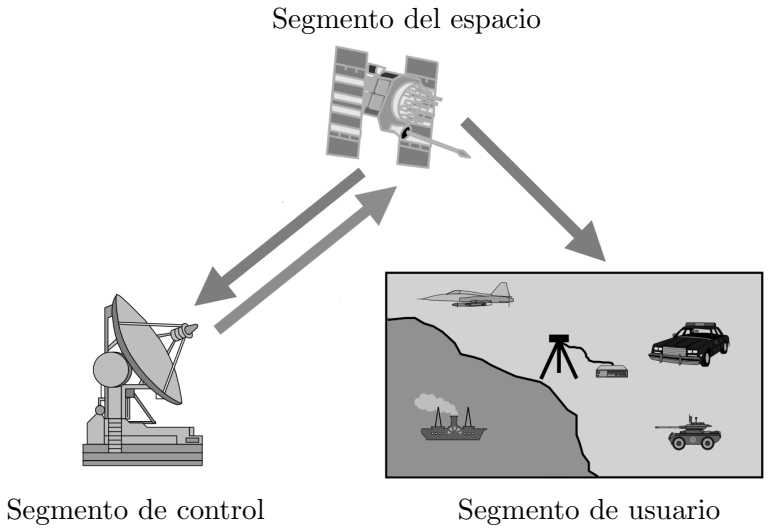


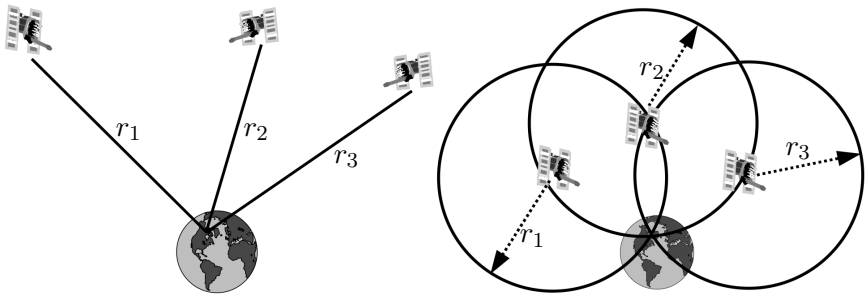
Figura 5.2: Segmentos que conforman el sistema GPS

Sabiendo entonces la distancia y posición de dos satélites, el receptor se encuentra en algún punto de la intersección de las dos esferas con centro en cada uno de los satélites, con radios r_1 y r_2 respectivamente, donde r_i es la distancia del receptor al satélite. Esta intersección es un círculo. Si conocemos la posición de un tercer satélite y su radio r_3, entonces el receptor se encuentra en alguno de los dos puntos de intersección de las tres esferas generadas por cada satélite, -figura 5.3(b)-. Podemos descartar uno de estos puntos de intersección como posición del receptor dado que está en el espacio. De esta manera, podemos conocer la ubicación en tres coordenadas, latitud, longitud y altura de nuestro receptor. En la práctica se utiliza un cuarto satélite para poder calcular el error de reloj que se genera en este tipo de señales [12].

Para tener una precisión aún mayor se usa un método que utiliza dos receptores GPS que rastrean simultáneamente los mismos satélites GPS. Con este método el nivel de precisión va desde unos cuantos centímetros a pocos metros.

5.2. Uso

La información que proporciona el GPS es fundamental para el funcionamiento de la aplicación. En la sección 3.4.5 ya vimos un poco de cómo se



(a) Distancia y posición de tres satélites GPS respecto a un receptor

(b) Intersección de las esferas con centro en cada satélite y radio r_n

Figura 5.3: Cálculo de la posición de un receptor GPS en base a la distancia y posición de tres satélites

usa el GPS dentro de Android. Pero no indicamos cómo pedimos que nos dé la ubicación. Para esto, debemos pedir un proveedor de ubicación al sistema. La mayoría de los teléfonos móviles con Android tienen varias maneras de encontrar la ubicación del dispositivo. Pueden utilizar las antenas de la red de telefonía para triangular su posición; también pueden detectar redes inalámbricas cercanas para acceder a una base de datos con la ubicación de éstas; por último, pueden utilizar el dispositivo GPS que tienen instalado. Es necesario indicar qué proveedor queremos utilizar, ya que sólo el GPS nos da la ubicación con la precisión que necesitamos.

Lamentablemente, por convenciones de Android, no se debe pedir directamente el GPS como proveedor, sino que hay que especificar las características que deseamos de este proveedor. Para esto utilizamos criterios y solicitamos que la precisión sea fina (código 9, línea 22) y que el proveedor de ubicación soporte darnos información de acimut (código 9, línea 23), ya que, como vimos en la figura 3.2, utilizaremos esta información para poder calcular el ángulo entre el dispositivo y los PI.

Por último, solicitamos que las actualizaciones de ubicación se hagan cada treinta segundos o cada que el dispositivo se haya movido dos metros, lo que suceda primero (código 9, línea 29).

En caso de que esté apagado el dispositivo GPS lanzamos una excepción y mostramos un diálogo para que el usuario pueda encenderlo -figura 3.7-.

```
/**
 * Iniciamos el servicio de GPS y su escucha, solicitamos
 * actualización de la ubicación cada 30 segundos o cuando
 * el dispositivo se mueva más de 2 metros. En general
 * importa actualizar el valor sólo si se ha cambiado de
 * posición.
 *
 * Solicitamos que el proveedor de ubicación tenga una
 * precisión fina, esto significa utilizar el GPS pero no
 * solicitamos ese proveedor directamente para seguir las
 * convenciones de Android.
 *
 * @throws LocationProviderNullException Se arroja en caso de
 *         que el GPS esté deshabilitado, <code>Main</code>
 *         hace el manejo de la excepción
 */
private void initGPS() throws LocationProviderNullException {
  String context = Context.LOCATION_SERVICE;
  locationManager = (LocationManager) Main.context
    .getSystemService(context);
  Criteria criteria = new Criteria();
  criteria.setAccuracy(Criteria.ACCURACY_FINE);
  criteria.setBearingRequired(true);
  String provider = locationManager.getBestProvider(criteria,
    true);
  if (provider == null) {
    throw new LocationProviderNullException();
  }
  locationManager.requestLocationUpdates(provider, 30000, 2,
    locationListener);
  updatePOILocation(locationManager.getLastKnownLocation(
    provider));
}
```

Código 9: Inicialización del GPS

5.3. Distancia geográfica

Gracias al GPS conocemos la latitud y longitud del dispositivo, y las fuentes de información para los PI nos dan las coordenadas de cada uno de ellos. Queremos saber a qué distancia se encuentran del dispositivo y cuál es el acimut.

Si estuviéramos en un plano cartesiano esto sería muy sencillo, ya que bastaría con utilizar la fórmula tradicional de distancia:

$$\text{distancia} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Pero éste no es el caso, pues la tierra es un cuerpo de forma casi esférica con un achatamiento en los polos, resultando en un esferoide, con las siguientes características:

- Circunferencia ecuatorial: $40,075,014$ m
- Circunferencia polar: $40,007,832$ m
- Radio ecuatorial: $6,378,137$ m
- Radio polar: $6,356,752$ m
- Achatamiento: 21.384 km
- Excentricidad: 0.00329

El achatamiento de la tierra es muy pequeño para su tamaño, por lo que para los siguientes cálculos, supondremos que la tierra es esférica. Así, para calcular la distancia entre dos puntos sobre la tierra usaremos la distancia del circulo mayor u ortodrómica. Esta es la distancia más corta entre cualesquiera dos puntos en la superficie de una esfera, medida a partir de un camino en la superficie de la esfera. En la geometría euclideana la distancia entre dos puntos es la longitud de la línea recta que los une. En la esfera, no hay líneas rectas, y éstas se reemplazan por geodésicas, las cuáles en la esfera son círculos mayores[1] -figura 5.4-.

Entre dos puntos diferentes sobre la esfera que no sean directamente opuestos uno de otro, existe un círculo mayor único. Los puntos separan el círculo mayor en dos arcos; la longitud del arco más corto es la distancia del círculo mayor entre los puntos. Entre dos puntos que sean directamente

[1] Círculo en la esfera cuyo centro es el mismo que el centro de la esfera.

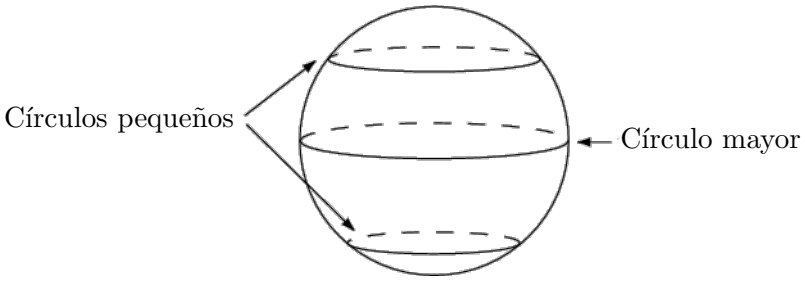


Figura 5.4: Círculo mayor

opuestos, hay una cantidad infinita de círculos mayores, pero todos sus arcos tienen la misma distancia, la cuál es la mitad de la circunferencia πr.

Definamos $\theta_1, \lambda_1; \theta_2, \lambda_2$ como la latitud y longitud del punto uno y punto dos respectivamente y $\Delta\theta, \Delta\lambda$ sus diferencias. Entonces $\Delta\widehat{\sigma}$ es el ángulo central[2] entre ambos puntos, que se calcula con la ley de cosenos para geometría esférica:

$$\Delta\widehat{\sigma} = \operatorname{arc\,cos}\left(\operatorname{sen}\phi_1 \operatorname{sen}\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda\right) \tag{5.1}$$

Así, la distancia entre los dos puntos, es decir, la longitud de arco para una esfera de radio r y $\Delta\widehat{\sigma}$ en radianes es:

$$\text{distancia} = r\Delta\widehat{\sigma} \tag{5.2}$$

La formula 5.1 puede tener errores de redondeo muy grandes si la distancia es pequeña, si los puntos están a un kilómetro de distancia, el coseno de ángulo central es 0.99999999. Para solucionar este problema se utiliza una ecuación llamada fórmula del semiverseno[3] que es más estable numéricamente para distancias pequeñas:

$$\Delta\widehat{\sigma} = 2\operatorname{arc\,sen}\left(\sqrt{\operatorname{sen}^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \operatorname{sen}^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{5.3}$$

A pesar de que la fórmula 5.3 es precisa para la mayoría de las distancias, también sufre de errores de redondeo para el caso poco común de puntos

[2]En un esfera es el ángulo cuyo vértice está en el centro de la esfera y sus lados pasan por un par de puntos sobre la esfera.

[3]Fórmula utilizada en la navegación para calcular distancias entre dos puntos de un gran círculo. Recibe su nombre por ser escrita comúnmente en términos de la función semiverseno dada por: $\text{semiverseno}(\theta) = \frac{\text{verseno}(\theta)}{2} = \operatorname{sen}^2(\frac{\theta}{2}) = \frac{1-\cos(\theta)}{2}$.

opuestos sobre la esfera. Para solucionar este problema se utiliza una fórmula más complicada que es precisa para todas las distancias en la esfera, la fórmula de Vicenty[4] [11]:

$$\Delta\widehat{\sigma} = \arctan\left(\frac{\sqrt{(\cos\phi_2 \operatorname{sen}\Delta\lambda)^2 + (\cos\phi_1 \operatorname{sen}\phi_2 - \operatorname{sen}\phi_1 \cos\phi_2 \cos\Delta\lambda)^2}}{\operatorname{sen}\phi_1 \operatorname{sen}\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda}\right) \tag{5.4}$$

Android nos proporciona directamente una forma de calcular la distancia entre dos puntos con el método `distanceTo(Location dest)` de la clase `Location`, el cual recibe una ubicación y nos devuelve la distancia en metros a ella. Este cálculo se realiza basado en el estándar WGS84 [7], el cual es un sistema geodésico para determinar un marco de coordenadas en la tierra y definirla como un esferoide. Revisando el código fuente del método podemos ver que la fórmula es similar a 5.4, y de hecho está basada en un artículo publicado por Vicenty en 1975 [11].

[4]Fórmula para calcular distancias entre dos puntos en la superficie de un esferoide, desarrollada por Thaddeus Vicenty en 1975.

Capítulo 6

Sobreposición de imágenes sobre la cámara

Una vez que tenemos todos los datos de los sensores, es decir, la dirección en la que apunta el dispositivo con respecto al norte magnético, su inclinación y su posición geográfica, ya tenemos todo lo necesario para calcular la posición de los PI respecto al dispositivo. Además, también ya tenemos una manera de obtener PI con sus respectivos datos. Así, queremos dibujar cada PI en la pantalla sobre la imagen de la cámara de tal manera que el PI dibujado quede exactamente sobre el lugar que está indicando y que se ve en la cámara, como se muestra en la figura 3.1.

6.1. Ángulo de visión de la cámara

Lo primero que necesitamos calcular para poder posicionar los PI es el ángulo de visión de la cámara, ya que necesitamos saber exactamente qué parte del mundo real es la que se verá en la pantalla, es decir, la parte de la escena que es captada en el sensor de la cámara, como podemos ver en la figura 6.1. Éste puede medirse de tres maneras: horizontal, vertical o diagonalmente. Nos interesan los ángulos horizontal y vertical ya que esos se traducen directamente a lo que vemos en la pantalla. Para obtener estos ángulos, Android proporciona a partir de la versión 2.2 los métodos `getHorizontalViewAngle` y `getVerticalViewAngle`, que nos dan esos valores directamente. Sin embargo, hay un problema, ya que este método sólo está disponible en las últimas versiones y aun así hay teléfonos donde no funciona correctamente, como el Droid Eris que arroja una excepción al ejecutar el método, a pesar de tener la versión 2.3 de Android. Es por eso que

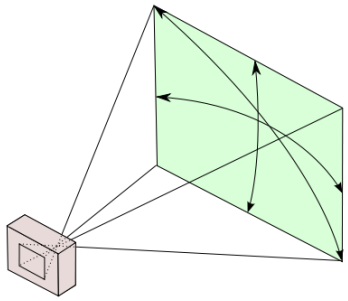


Figura 6.1: Ángulo de visión de la cámara

probamos estos métodos y en caso de algún error utilizamos ángulos por omisión obtenidos de la página del fabricante, HTC. Además, al revisar las especificaciones de varios equipos, comprobamos que tienen valores bastante similares. Así, usamos para el ángulo horizontal 49.5° y para el ángulo vertical 34.2°.

6.2. Cálculo de la posición de los objetos en la pantalla

La posición horizontal de cada PI está determinada por su acimut y la dirección de la cámara. Definamos la noción de "brazos de cámara": El "brazo izquierdo" es el punto más a la izquierda que puede ver la cámara y el "brazo derecho" es el punto más a la derecha que ve la cámara, como vemos en la figura 6.2. Cada uno de estos brazos tiene un valor que se calcula cada que la cámara cambia de dirección, -ver código 10-.

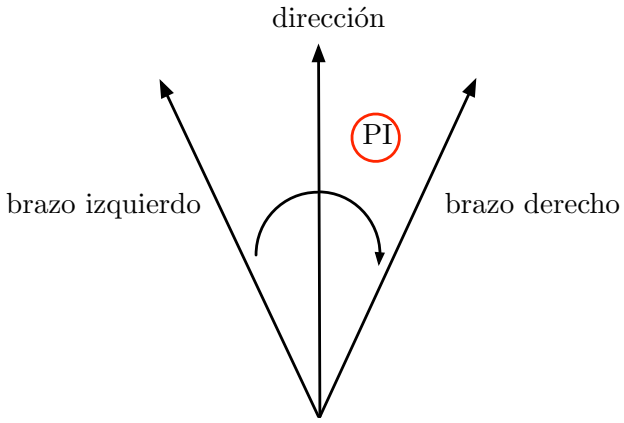


Figura 6.2: Ángulo de visión de la cámara

```
1  private float brazoIzquierdo() {
2    float bi = direccion - (ANGULO_HORIZONTAL_DE_CAMARA/2);
3    if (bi < 0)
4      bi += 360;
5    return bi;
6  }
7
8  private float brazoDerecho() {
9    float bd = direccion + (ANGULO_HORIZONTAL_DE_CAMARA/2);
10   if (bd > 360)
11     bd -= 360;
12   return bd;
13 }
```

Código 10: Cálculo del brazo izquierdo y derecho de la cámara

Una vez teniendo estos valores podemos calcular la posición horizontal de cada PI. También necesitaremos las dimensiones de la pantalla en pixeles, lo que puede obtenerse fácilmente como vemos en el código 11. Lo que haremos será calcular la diferencia entre la dirección y el brazo izquierdo (código 12, línea 5), que nos da la posición del PI con respecto al brazo

izquierdo en grados; después calculamos la proporción de este valor con respecto al ángulo horizontal de vista de la cámara (código 12, línea 13). Por último multiplicamos este porcentaje por el ancho de la pantalla para obtener su coordenada en x. Todo esto se hace directamente siempre y cuando el brazo derecho sea mayor al brazo izquierdo. De no ser así entonces la dirección 0° está entre el brazo izquierdo y derecho, por lo que tenemos que hacer algunos ajustes. Si la dirección es mayor o igual que el brazo izquierdo podemos usar el procedimiento antes mencionado; si no, debemos sumar 360 al resultado (código 12, línea 10).

```
1 private void initLayout() {
2     Display pantalla = ((WindowManager) Main.context
3             .getSystemService(Context.WINDOW_SERVICE))
4             .getDefaultDisplay();
5     pantallaAltura = pantalla.getHeight();
6     pantallaAncho = pantalla.getWidth();
7     piPopup = new PIPopup(screenWidth, screenHeight);
8 }
```

Código 11: Obtención de las dimensiones de la pantalla en pixeles

Para calcular la posición en y utilizamos un procedimiento similar, solo que en este caso únicamente necesitaremos un brazo, que llamaremos brazo superior. Esto es porque en la inclinación los valores están en $[-90°, 90°]$ y nunca cruzamos de 90° a −90°. Además, necesitaremos la inclinación del PI, que se calcula directamente en la clase `PI`, primero verificando que tanto el PI como el dispositivo tengan disponible la información de altitud. Si no la tenemos entonces ponemos la inclinación en 0° que es en el horizonte(código 13, línea 17). Si tenemos esta información, debemos calcular la diferencia entre la altitud del PI y la del dispositivo (código 13, línea 6), después dividimos esta diferencia entre la distancia a la que se encuentra el dispositivo para que la inclinación sea proporcional a la distancia. Por último, si la altitud del PI es menor que la del dispositivo ponemos la inclinación como negativa (código 13, línea 13).

Con esta información podemos calcular la posición en y del PI, en el código 14 podemos ver cómo realizamos este cálculo que es muy similar al código 12.

El resultado de posición que obtuvimos para el PI será el centro del

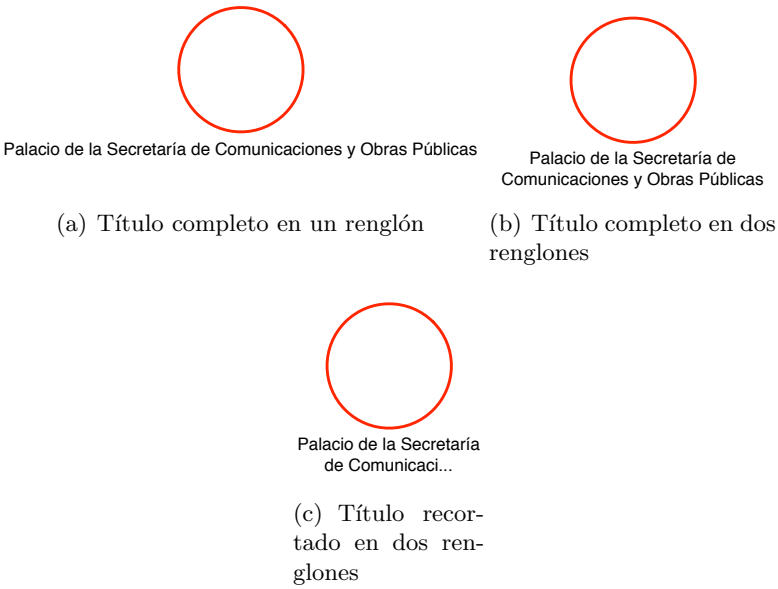
```
1  private float posicionX(float direccion, float brazoIzquierdo,
2                          float brazoDerecho) {
3    float x;
4    if (brazoIzquierdo < brazoDerecho) {
5      x = direccion - brazoIzquierdo;
6    } else {
7      if (direccion >= brazoDerecho) {
8        x = direccion - brazoIzquierdo;
9      } else {
10       x = 360 - brazoIzquierdo + direccion;
11     }
12   }
13   float proporcion = x / ANGULO_HORIZONTAL_DE_CAMARA;
14   return proporcion * pantallaAncho;
15 }
```

Código 12: Cálculo de la posición en x de un PI

círculo que dibujaremos en la pantalla sobre cada PI. Debajo de cada círculo pondremos el nombre del PI como vimos en la figura 3.1, aunque cortaremos este texto ya que varios de los PI tienen un nombre muy largo. Por ejemplo, con la consulta a GeoNames hecha en la sección 3.2 uno de los resultados es "Palacio de la Secretaría de Comunicaciones y Obras Públicas", que al tener sesenta caracteres de largo complica el que se muestren varios PI en pantalla al mismo tiempo pues colisionarían los nombres. Una opción es poner el texto en varios renglones, que también utilizaremos, pero solo usaremos dos renglones como máximo para que no se extienda indefinidamente hacia abajo el nombre del PI. Además, en general con la primera parte del nombre podremos saber de que se está hablando y siempre podemos tocar el PI para obtener más información. Entonces, siempre recortaremos el nombre en caso de que sea de más de cuarenta caracteres, pues a partir de esta longitud el texto se extiende más allá del ancho del PI y se presenta el problema de colisiones como vemos en la figura 6.3.

```
1  private void setInclinacion() {
2      if (PI.ubicacionDispositivo.hasAltitude() &&
3          ubicacionPI.hasAltitude()) {
4          double diferenciaAltitud;
5          boolean negativo = false;
6          diferenciaAltitud = Math.abs(ubicacionPI.getAltitude()
7                  - PI.ubicacionDispositivo.getAltitude());
8          double coef = diferenciaAltitud / (double) distancia;
9          inclinacion = (float) Math.toDegrees(Math.atan(coef));
10
11         if (ubicacionPI.getAltitude() <
12             PI.ubicacionDispositivo.getAltitude()) {
13             inclinacion *= -1;
14     } else {
15         //En caso de no tener un valor de altitud lo
16         //colocamos en el horizonte
17         inclinacion = 0;
18     }
19 }
```

Código 13: Cálculo de la la inclinación de un PI



(a) Título completo en un renglón

(b) Título completo en dos renglones

(c) Título recortado en dos renglones

Figura 6.3: Diferentes maneras de mostrar el título de un PI debajo de él

```
1  private float brazoSuperior() {
2    return inclinacion + (ANGULO_VERTICAL_DE_CAMARA/2);
3  }
4
5  private float posicionY(float inclinacion,
6                          float brazoSuperior) {
7    float y;
8    if (brazoSuperior >= 0) {
9      y = brazoSuperior - inclinacion;
10   } else {
11     if (brazoSuperior < inclinacion) {
12       y = brazoSuperior - inclinacion;
13     } else {
14       y = -brazoSuperior - incinclinacion;
15     }
16   }
17   float proporcion = y / ANGULO_VERTICAL_DE_CAMARA;
18   return proporcion * pantallaAlto;
19 }
```

Código 14: Cálculo de la posición en y de un PI

6.2.1. Tamaño de los PI en la pantalla

Dado que Android es un sistema abierto, cualquier compañía puede utilizarlo para fabricar dispositivos con él. Esto presenta uno de los grandes problemas de Android pues hay una gran cantidad de dispositivos distintos, con pantallas de diferentes tamaños y resoluciones, y los desarrolladores tienen que lidiar con esto. Es por esto que no podemos utilizar un tamaño fijo para dibujar en la pantalla, ya que si elegimos por ejemplo un radio fijo para cada PI en pixeles, éste se verá muy distinto en un dispositivo con resolución de 480×320 con pantalla de 3.2in que en un dispositivo con resolución de 960×640 y el mismo tamaño de pantalla; de hecho, se verá de la mitad de tamaño en el segundo dispositivo. Queremos que lo que dibujemos en pantalla se vea bien relativo al tamaño de la pantalla, más allá de la resolución, para que el texto sea legible y para que cada PI sea de un tamaño suficiente para poder tocarlo y obtener más información sobre él.

Queremos que cada PI tenga siempre el mismo tamaño en pulgadas una

vez dibujado en la pantalla, independientemente de su tamaño o resolución. Para esto calcularemos el tamaño en pixeles que necesita tener el radio de cada PI para tener el tamaño deseado en pulgadas. Android nos da dos datos con los que podemos realizar este cálculo; el primero es el ancho y alto de la pantalla en pixeles, y el segundo es los puntos por pulgada en el ancho y alto de la pantalla. Con esta información es fácil calcular la longitud en pixeles que debe tener el radio del círculo que dibujaremos, -ver algoritmo 3-. Para el raro caso en que la densidad en alguna de las dimensiones sea distinta, calculamos para x e y utilizando el radio mayor que tengamos. Como vemos en el algoritmo 3 estamos usando $\frac{3}{16}$in como radio deseado para cada PI, puesto que este tamaño es suficiente para ver bien el PI y poder tocarlo; si fuese más pequeño sería difícil poder seleccionarlo. También debemos calcular el tamaño del texto que dibujaremos para que se vea de un tamaño adecuado, para lo que hacemos algo muy similar al algoritmo 3, pero utilizando el ancho del texto a mostrar.

Algoritmo 3 Cálculo del radio del PI en pixeles

Entrada: pantallaAncho, pantallaAlto, densidadAncho, densidadAlto

$\text{radioPI} \leftarrow \frac{3}{16}$

$\text{longitudX} \leftarrow \frac{\text{pantallaAncho}}{\text{densidadAncho}}$

$\text{longitudY} \leftarrow \frac{\text{pantallaAlto}}{\text{densidadAlto}}$

$\text{xIn} \leftarrow \frac{\text{longitudX}}{\text{radioPI}}$

$\text{yIn} \leftarrow \frac{\text{longitudY}}{\text{radioPI}}$

$\text{radioX} \leftarrow \frac{\text{pantallaAncho}}{\text{xIn}}$

$\text{radioY} \leftarrow \frac{\text{pantallaAlto}}{\text{yIn}}$

$\text{radioPI} \leftarrow \text{máx}(\text{radioX}, \text{radioY})$

6.3. Detección de colisiones

El tamaño de la pantalla en la que mostraremos toda la información de la aplicación en general es pequeño, de 3 a 4 pulgadas en diagonal y de 2.66 a 3 pulgadas de largo, con lo que si queremos que cada círculo que representa a un PI tenga un radio de $\frac{3}{16}$ de pulgada entonces sólo podemos mostrar siete u ocho PI al mismo tiempo. Además, al seleccionar el rango para el

Figura 6.4: PI colisionando

cuál queremos que se muestren PI es altamente probable que haya más de un PI en una misma dirección, o que estén suficientemente cerca como para chocar al dibujarse. Por ejemplo, si aumentamos el rango de vista de los PI para la figura 3.1, tendremos detrás de la Catedral el Centro Cultural de España, el Museo de la Caricatura y la Plaza de Santo Domingo entre otros, por lo que los PI colisionarán unos con otros y se verá como en la figura 6.4. Cuando esto sucede no podemos ver claramente ninguno de los PI ni seleccionar alguno de ellos para obtener más información.

Lo que haremos para solucionar este problema es dibujar los PI por orden de más cercano a más lejano y para cada uno de ellos verificaremos si colisiona con alguno de los que ya se dibujaron anteriormente. Si colisiona entonces lo moveremos hacia arriba para que no colisione pero pueda dibujarse. Claro que no podemos hacer esto con todos lo puntos ya que si por ejemplo, hay diez puntos que colisionan, entonces deberíamos subir cada uno de ellos y de igual manera en la pantalla no veríamos más que los primeros tres o cuatro. Así, pondremos una variable que defina el número máximo de colisiones que puede tener un PI; después de ese número, si un PI más colisiona ya no se dibujará. Podemos ver cómo hacer esto en el código 15.

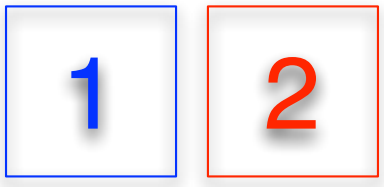
```
public static void checarColisionesDePI(PI pi2, int n) {
  for (int i = 0; i < n; ++i) {
    PI pi1 = listaPI.get(i);
    int colision = checarColisionDePI(pi2, pi1);
    switch (colision) {
      case -1:
        //No debe dibujarse y ya no hace falta checar nada más
        poi.setEsVisiblePorColisiones(false);
        return;
      case 0:
        //No colisiona, no hay que hacer nada
        poi.setEsVisiblePorColisiones(true);
        break;
      case 1:
        //Colisiona, hay que moverlo y seguir checando
        poi.moverPIArriba();
        break;
      default:
        break;
    }
  }
}
```

Código 15: Cálculo de colisiones entre dos PI

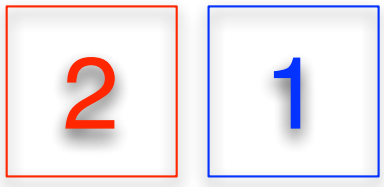
Para detectar las colisiones haremos una verificación en x y una en y. Para esto utilizaremos fronteras para cada PI a la izquierda, derecha, arriba y abajo. Estas fronteras se calculan cada que cambia la posición del PI ya que se obtienen a partir de su posición en x y y sumándole y restándole a cada coordenada una constante que definimos a partir del tamaño del PI, sumándole un par de pixeles por tolerancia.

Analizaremos el caso de la detección de colisiones para x, siendo completamente análogo el caso de las colisiones verticales. Tenemos cinco posiciones en las que se pueden encontrar dos PI:

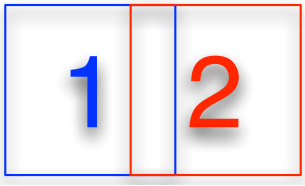
1. El primer PI está a la izquierda del segundo PI, no colisionan. Figura 6.5(a)



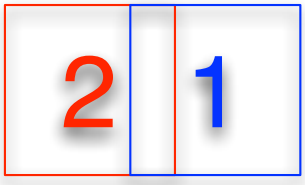
(a) Título completo en un renglón



(b) Título completo en dos renglones



(c) Título recortado en dos renglones



(d) Título recortado en dos renglones

Figura 6.5: Diferentes posiciones en las que se pueden encontrar dos PI

2. El segundo PI está a la izquierda del primer PI, no colisionan. Figura 6.5(b)

3. El primer PI está a la izquierda del segundo PI, colisionan. Figura 6.5(c)

4. El segundo PI está a la izquierda del primer PI, colisionan. Figura 6.5(d)

5. El último caso en realidad son tres situaciones distintas pero las trataremos como el mismo caso por su similitud. Éste se da cuando alguna de las fronteras es igual a la otra, es decir, ambos PI están en la misma posición y tienen las mismas fronteras, o la frontera izquierda de uno colisiona con la frontera derecha del otro o viceversa.

Una vez teniendo estos casos, lo primero que verificaremos es el caso 5 pues es el más sencillo (código 16, línea 5). Después empezamos con los demás casos, primero checamos cuando los PI están en el caso 1 ó 3 (código 16, línea 10), y el caso en que colisionan es el 3 (código 16, línea 11). Luego checamos cuando los PI están en el caso 2 ó 4 (código 16, línea 15) y el caso en que colisionan es 4 (código 16, línea 16). Después verificamos si colisionan verticalmente de manera análoga, aunque no se muestra el código. En caso de que colisionen vertical y horizontalmente entonces se enciman

Figura 6.6: PI que colisionaban tras aplicar el algoritmo de detección de colisiones

(código 16, línea 26), si se alcanza el número máximo de colisiones para ese PI ya no dibujaremos el que acaba de colisionar (código 16, línea 27); si no se ha alcanzado, incrementamos el número de colisiones y moveremos este PI hacia arriba. Si no colisionan regresamos 0 para dibujar el PI en su posición actual.

Utilizando este algoritmo en la figura 6.4, con un máximo número de colisiones de 2, obtenemos la figura 6.6 donde se muestran tres PI en vez de los cuatro anteriores, pero ya no hay colisiones y el Museo de la Caricatura aparece desplazado hacia arriba para indicar que está más lejos.

Cada vez que recibimos una nueva lectura de los sensores debemos calcular las posición de cada PI y calcular si hay colisiones; esto se hace dentro del escucha de los sensores (código 7, líneas 21 y 24). Una vez realizados estos cálculos, se redibuja todo en la pantalla con una llamada a `postInvalidate` (código 7, línea 30)

6.3.1. Reacción al toque de la pantalla

Ya que dibujamos todo lo que queremos en la pantalla, falta poder reaccionar a las acciones del usuario, en específico cuando toca uno de los PI en la pantalla. Para esto, cada evento de toque de pantalla que ocurra en la capa `ARLayer` (sección 3.3) lo pasamos a cada uno de los PI. Cada PI se encargará de ver si las coordenadas de este evento de toque de pantalla están dentro del área donde está dibujado; de ser así, abre una nueva ventana que cargará la página web con información adicional sobre este PI, como vimos en la figura 3.3 (código 18).

```
1  private static int checarColisionDePI(PI p2, PI p1) {
2    boolean colisionHorizontal, colisionVertical = false;
3
4    if (p1.izquierda == p2.izquierda || p1.derecha == p2.derecha
5    || p1.izquierda == p2.derecha
6    || p1.derecha == p2.izquierda){
7      //Chocan horizontalmente
8      colisionHorizontal = true;
9    } else {
10     if (p2.izquierda > p1.izquierda) {
11       if (p2.izquierda < p1.derecha) {
12         //Chocan horizontalmente
13         colisionHorizontal = true;
14       }
15     } else {
16       if (p1.izquierda < p2.derecha) {
17         //Chocan horizontalmente
18         colisionHorizontal = true;
19       }
20     }
21   }
22   //Verificamos si colisiona verticalmente
23
24   if (colisionHorizontal && colisionVertical) {
25     if (p1.contadorColisiones >= COLISIONES_MAXIMAS) {
26       return -1;
27     } else {
28       p1.contadorColisiones++;
29       return 1;
30     }
31   } else {
32     return 0;
33   }
34 }
```

Código 16: Método que decide qué hacer con base en los resultados de la verificación de colisiones

```
1  private void actualizarLayoutPI(Location ubicacion) {
2    if (ubicacion != null) {
3      actualizarUbicacionPI(ubicacion);
4    }
5
6    for (PI pi : listaPI) {
7      pi.contadorColisiones = 0;
8      pi.setEsVisiblePorColisiones(true);
9    }
10   int i = 0;
11   for (PI pi : listaPI) {
12     //Si el POI no es visible no hace falta calcular nada
13     //pues no se dibujará
14     if (pi.esVisibleEnRango()) {
15       // Calculamos la posición en x y en y de este POI y
16       //redondeamos al entero más cercano
17       int x = Math.round(posicionX(pi.getAcimut()));
18       int y = Math.round(posicionY(pi.getInclinacion()));
19       pi.layoutPI(x, y, x, y);
20       checarColisionDePI(pi, i);
21       ++i;
22     }
23   }
24 }
```

Código 17: Actualización de la posición de cada PI

```
/**
 * Mandamos cada evento de toque de pantalla a cada PI
 */
public boolean onTouchEvent(MotionEvent event) {
  boolean ret = false;
  for (PI pi : listaPI) {
    ret = pi.dispatchTouchEvent(event);
  }
  return ret;
}

/**
 * Calculamos si este evento de toque de pantalla
 * le corresponde a este PI checando si las coordenadas
 * del evento están dentro del área de este PI
 */
public boolean onTouchEvent(MotionEvent event) {
  if ((new Region(getLeft(), getTop(), getRight(),
      getBottom())).contains((int) event.getX(),
      (int) event.getY())) {
    ARLayer.PIPopup.show(this);
    return true;
  }
  return false;
}
```

Código 18: Manejo de eventos de toque de pantalla

Capítulo 7

Conclusión

La realidad aumentada se ha vuelto cada vez más popular gracias a la potencia y cantidad de sensores de los dispositivos móviles. Una gran cantidad de personas tienen un teléfono inteligente con cámara, GPS, acelerómetro, magnetómetro y otros sensores. Esto permitirá que cada vez se hagan más aplicaciones como la realizada en este trabajo. En particular Android sigue creciendo enormemente como plataforma, convirtiéndose en una gran oportunidad para los desarrolladores de llevar sus aplicaciones a más usuarios.

Las mayores dificultades para realizar esta aplicación fueron el obtener datos suaves de los sensores, desplegar los PI en la pantalla de manera correcta y resolver las colisiones entre ellos.

Para poder resolver el primer problema primero observamos que los sensores, a pesar de ser bastante precisos, presentan mucho ruido. Además éste varía entre dispositivos y puede ser afectado por muchos factores externos, como estar cerca de alguna fuente de interferencia magnética, por ejemplo una bocina; también una caída o golpe del dispositivo puede afectarlo. Por eso debemos contemplar un amplio margen para el ruido sin que el proceso de filtrado sea tan pesado que ocasione retrasos significativos en la interfaz gráfica.

Revisamos primero dos filtros, promedios y paso bajos. Ambos suavizan los datos lo suficiente para la aplicación, pero ninguno responde a cambios rápidos, teniendo un gran retraso en varias situaciones. El suavizado de datos de ambos filtros es muy similar, como podemos ver en las figuras 4.13 y 4.14, por lo que se decidió utilizar el filtro paso bajos pues es mucho más eficiente tanto en memoria (sólo tenemos que guardar el dato anterior) como en procesamiento (en el filtro de promedios debemos acceder a cada elemento del arreglo que guarda los valores, sumarlos y dividirlos). Si bien esto no

es costoso, al tener que hacerlo treinta veces por segundo para tres valores distintos se vuelve indispensable ahorrar cada recurso posible. Recordemos también que trabajamos en hardware muy limitado.

Para poder reaccionar rápidamente a los cambios grandes en los valores de los sensores revisamos un tercer filtro, el filtro óptimo. Agregamos una condición con la cuál, si el nuevo valor dado por los sensores es mayor a nuestro umbral, entonces tomamos ese nuevo valor sin filtrar, con lo que eliminamos todo retraso en la interfaz gráfica.

El segundo problema, mostrar los PI en la pantalla de manera correcta, lo solucionamos gracias a tener el ángulo de visión de la cámara -figura 6.1- ya sea a partir de los métodos indicados en la sección 6.1 o de las especificaciones dadas por el fabricante. Con estos datos utilizamos las nociones de brazos (sección 6.2), para poder determinar la posición de cada PI en porcentaje respecto a la pantalla y después, teniendo las medidas de la pantalla en pixeles, poder escalar a estas dimensiones. También obtuvimos los puntos por pulgada de la pantalla para que los PI se vean del mismo tamaño en todos los dispositivos.

El empalme de los distintos PI en la pantalla presentó también un gran problema pues hacía inusable la aplicación. Para resolverlo delimitamos fronteras para cada PI y con base en la figura 6.5 hicimos los distintos casos para detectar las colisiones. Ordenamos los PI del más cercano al más lejano, y los intentamos dibujar así; antes de dibujar cada PI comprobamos si colisionaba con alguno de los ya dibujados en cuyo caso lo movemos para dibujarlo arriba. También agregamos un límite de colisiones para cada PI, con lo que los PI que colisionen no se expandirán de manera indefinida hacia arriba; además, cuando al detectar una colisión y ver que se había alcanzado el límite de colisiones para este PI, dejamos de procesar el resto de colisiones y no gastamos recursos.

Esta aplicación puede tener muchos usos; aquí nos enfocamos particularmente en mostrar lugares históricos y turísticos, lo cual tiene una gran utilidad. El ejemplo del zócalo es una clara muestra de esto pues fácilmente podemos saber qué son todos los edificios que nos rodean: la Catedral Metropolitana, el Palacio Nacional, etc. Además es fácilmente extendible a otro tipo de fuentes de información como negocios, gasolinerías, cajeros automáticos, etc. También podría extenderse a usos un poco más complicados, por ejemplo: podríamos programar visitas guiadas a ciudades o lugares turísticos donde la aplicación nos muestre a dónde ir en la pantalla y una vez que lleguemos nos indique los lugares de interés; podrían fabricarse dispositivos con Android que tengan otra fuente para obtener su posición además de GPS, para que funcione en lugares cerrados; de este modo podríamos

extender la aplicación para que nos dé información de lo que hay dentro de un museo o lugares similares.

Hay más aplicaciones o usos de la realidad aumentada; por ejemplo, la aplicación Google Sky Map [4] muestra las constelaciones y estrellas al apuntar un teléfono hacia el cielo. En medicina también se ha utilizado (figura 1.4): en mayo de 2011 se utilizó una aplicación de realidad aumentada para realizar una operación de rodilla [24]. Otros posibles usos van desde la publicidad hasta resolver sudokus [3].

Sin duda, la realidad aumentada seguirá ampliando sus horizontes volviéndose cada vez más popular y será utilizada para realizar actividades que tal vez ahora ni siquiera imaginemos.

En este trabajo se ha buscado integrar una aplicación de la realidad aumentada a un dispositivo móvil con el sistema operativo Android, con la que se pretende que viajar y encontrar información pertinente para el turista se realice de manera sumamente práctica, sencilla y eficaz, pudiendo además ser extendida para diferentes usos.

Glosario

6dof	Siglas del inglés 6 Degrees of Freedom.
acelerómetro multieje	Dispositivo que mide la aceleración. Los acelerómetros multieje detectan la magnitud y la dirección de la aceleración como una cantidad vectorial y pueden utilizarse para determinar orientación.
acimut	En náutica, el acimut es el ángulo o longitud de arco entre el punto cardinal norte en sentido horario de 0° a 360° y otro punto.
API	Siglas del inglés Application Programming Interface. Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
AWT	Siglas del inglés Abstract Window Toolkit. Es el primer conjunto de herramientas de interfaz gráfica de Java.
buffer	Región de memoria utilizada temporalmente para almacenar datos mientras se mueven de un lugar a otro.

bytecode	El bytecode es un código intermedio más abstracto que el código de máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código de máquina.
codec	Siglas del inglés COmpressor-DEcompressor o, más comúnmente, COder-DEcoder. Es un dispositivo o programa capaz de codificar y/o decodificar un flujo de datos digitales o una señal.
círculo mayor	Círculo en la esfera cuyo centro es el mismo que el centro de la esfera.
Droid Eris	Teléfono marca HTC con Android 2.1. Procesador Qualcomm® MSM7600^{TM} de 528MHz. Memoria RAM 288MB. http://www.htc.com/us/products/droid-eris-verizon.
filtro de Kalman	El filtro de Kalman es un algoritmo desarrollado por Rudolf E. Kalman en 1960. Sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal, es decir, utilizar medidas que son observadas a través del tiempo y que contienen ruido y producir valores que tienen a ser más cercanos a los valores reales que los valores medidos.
fórmula de Vicenty	Fórmula para calcular distancias entre dos puntos en la superficie de un esferoide, desarrollada por Thaddeus Vicenty en 1975.
fórmula del semiverseno	Fórmula utilizada en la navegación para calcular distancias entre dos puntos de un gran círculo. Recibe su nombre por ser escrita comúnmente en términos de la función semiverseno dada por: $\text{semiverseno}(\theta) = \frac{\text{verseno}(\theta)}{2} = \text{sen}^2(\frac{\theta}{2}) = \frac{1-\cos(\theta)}{2}$.

GeoNames	Base de datos geográfica gratuita y accesible a través de Internet bajo una licencia Creative Commons. http://www.geonames.org/
Git	Sistema de control de versiones distribuido con énfasis en la velocidad. Fue originalmente diseñado y desarrollado por Linus Torvalds. http://git-scm.com/
GPS	Siglas del inglés Global Positioning System. Sistema que permite conocer la posición de un objeto móvil gracias a la recepción de señales emitidas por una red de satélites.
HMD	Siglas del inglés Head Mounted Display. Dispositivo de visualización similar a un casco que permite reproducir imágenes creadas por computadora sobre una pantalla o similar muy cercano a los ojos o directamente sobre la retina.
HTML	Siglas del inglés HyperText Markup Language. Es el lenguaje de marcado predominante para la elaboración de páginas web.
JIT	Siglas del inglés Just In Time. La compilación en tiempo de ejecución es una técnica para mejorar el rendimiento de sistemas de programación que compilan a bytecode; consiste en traducir el bytecode a código de máquina nativo en tiempo de ejecución.
JSON	Siglas del inglés JavaScript Object Notation u objeto de notación de javascript. Es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.http://www.json.org/.

JVM	Siglas del inglés Java Virtual Machine, o máquina virtual de Java. Es un máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (Java bytecode), el cual es generado por el compilador del lenguaje Java.
magnetómetro	Aparato que mide la intensidad, y algunas veces también la dirección, de un campo magnético.
MEMS	Siglas del inglés Micro-Electro-Mechanical Systems.
MIDlet	Un MIDlet es una aplicación que utiliza el perfil MIDP (Mobile Information Device Profile) y la especificación CLDC (Connected Limited Device Configuration), ambos para Java ME.
NDK	Siglas del inglés Native Development Kit. Es un kit de desarrollo para poder programar directamente sobre la plataforma de cómputo que se está usando y no usar la máquina virtual.
Nexus One	Teléfono marca HTC con Android 2.3. Procesador Qualcomm QSD 8250 Snapdragon de 1 GHz. Memoria RAM 512MB. http://www.google.com/phone/detail/nexus-one.
OpenGL ES	Siglas del inglés Open Graphics Library for Embedded System. Es un subconjunto de la API de gráficos en 3D OpenGL diseñado para sistemas embebidos como teléfonos celulares, PDA y consolas de videojuegos. http://www.khronos.org/opengles/
PI	Punto de Interés que se mostrará en el dispositivo.

SDK	Siglas del inglés Software Development Toolkit, o kit de desarrollo de software. Es generalmente un conjunto de herramientas de desarrollo que le permiten a un programador crear aplicaciones para un sistema concreto.
SGL	Siglas del inglés Scalable Graphics Library. Es el subsistema gráfico utilizado por Android.
SQL	Siglas del inglés Structured Query Language. Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas
Swing	Swing es el principal conjunto de herramientas de interfaz gráfica de Java.
WebKit	Motor de disposición diseñado para permitir a navegadores web procesar páginas web. Es utilizado por navegadores como Safari y Chrome. http://webkit.org/
Wikipedia	Enciclopedia libre y políglota de la Fundación Wikimedia. http://www.wikipedia.org/
XML	Siglas del inglés eXtensible Markup Language o lenguaje de marcas extensible. Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C) http://www.w3.org/XML/.
ángulo central	En un esfera es el ángulo cuyo vértice está en el centro de la esfera y sus lados pasan por un par de puntos sobre la esfera.

Bibliografía

[1] Versión 2.3 de Android `http://developer.android.com/sdk/android-2.3.html`.

[2] Laurence Goasduff and Christy Pettey. *Gartner Says Worldwide Mobile Device Sales Grew 13.8 Percent in Second Quarter of 2010, But Competition Drove Prices Down.* `http://www.gartner.com/it/page.jsp?id=1421013`.

[3] Sudoku Grab `http://itunes.apple.com/us/app/sudoku-grab/id305614901?mt=8`.

[4] Skymap `http://www.google.com/mobile/skymap/`.

[5] Compañía SkyHook `http://www.skyhookwireless.com/`.

[6] SQLite `http://www.sqlite.org`.

[7] Clase `Location` de Android `http://developer.android.com/reference/android/location/ Location.html`).

[8] Clase `SurfaceView` de Android `http://developer.android.com/reference/android/view/SurfaceView.html`.

[9] Sensor `TYPE_ORIENTATION` de Android `http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ORIENTATION`.

[10] Edwin Hall. *On a new action of the magnet on electric currents. Am. J. Math.*, 2:287–292, 1879.

[11] T. Vincenty. *Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Survey Review*, 23(176):88–93, 1975.

[12] Richard B. Langley. *The Mathematics of GPS. GPS World*, 2(7):45–50, 1991.

[13] Ronald Azuma. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.

[14] Steven Feiner, Blair MacIntyre, Tobias Höllerer, and Anthony Webster. *A Touring Machine: Prototyping 3D Augmented Reality Systems for Exploring the Urban Environment. Personal and Ubiquitous Computing*, 1(3), 1997.

[15] Miguel Ribo, Axel Pinz, and Anton L. Fuhrmann. A new optical tracking system for virtual and augmented reality applications. In *In Proceedings of the IEEE Instrumentation and Measurement Technical Conference*, pages 1932–1936, 2001.

[16] Guy Kulwanoski and Jeff Schnellinger. *The Principles of Piezoelectric Accelerometers.* `http://www.sensorsmag.com/sensors/acceleration-vibration/the-principles-piezoelectric-accelerometers-1022`, 1 2004.

[17] Kiyoshi Kiyokawa. *An Introduction to Head Mounted Displays for Augmented Reality. Emerging Technologies of Augmented Reality: Interfaces and Design.*, 2007.

[18] Nassir Navab, Joerg Traub, Tobias Sielhorst, Marco Feuerstein, and Christoph Bichlmeier. *Action- and workflow-driven augmented reality for computer-aided medical procedures. IEEE Comput Graph Appl*, 27(5):10–4, 2007.

[19] Andy Rubin. *Where's my Gphone?* `http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html`, 11 2007.

[20] Dan Bornstein. *Dalvik VM Internals.* Google IO, 2008.

[21] Brian X. Chen. *If You're Not Seeing Data, You're Not Seeing.* `http://www.wired.com/gadgetlab/2009/08/augmented-reality/`, agosto 2009.

[22] Charles Arthur. *Eric Schmidt's dog whistle to mobile developers: abandon Windows Phone*. `http://www.guardian.co.uk/technology/2010/jun/25/android-schmidt-mobile-platform`, 2010.

[23] Dan Bornstein. *Dalvik JIT* `http://android-developers.blogspot.com/2010/05/dalvik-jit.html`, 05 2010.

[24] Annkur. *iPod Touch Used For Joint Replacement Surgery, Arrives In India*. `http://onlygizmos.com/ipod-touch-used-for-joint\-replacement-surgery-arrives-in-india/2011/04/`, abril 2011.

[25] Frank Ableson and Robi Sen. *Android in Action*. Manning Publications, 2 edition, 1 2011.

[26] Oliver Bimber and Ramesh Raskar. *Spatial Augmented Reality: Merging Real and Virtual Worlds*. A. K. Peters, Ltd., 2005.

[27] Ahmed El-Rabbany. *Introduction to GPS: The Global Positioning System, Second Edition*. Artech House Publishers, August 2006.

[28] Reto Meier. *Professional Android 2 Application Development*. Wrox Press Ltd., 2010.

[29] James Steele and Nelson To. *The Android Developer's Cookbook: Building Applications with the Android SDK*. Addison-Wesley, 2010.