Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Centro de Estudios de Electrónica y Tecnologías de la Información

TRABAJO DE DIPLOMA

Diseño de podómetro en dispositivo móvil: el *i*-Walker

Autor: Wilker Quiala Cutiño

Tutor: Dr.C. Alberto Taboada Crispi

Santa Clara

2013

"Año 55 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Centro de Estudios de Electrónica y Tecnologías de la Información

TRABAJO DE DIPLOMA

Diseño de podómetro en dispositivo móvil: el *i*-Walker

Autor: Wilker Quiala Cutiño

e-mail: quiala@uclv.edu.cu

Tutor: Dr.C. Alberto Taboada Crispi

PT, IT, CEETI, Fac. Ing. Eléctrica, ataboada@uclv.edu.cu

Consultante: Dr.C. Julián L. Cárdenas Barrera

PT, CEETI, Fac. Ing. Eléctrica, julian@uclv.edu.cu

Santa Clara

2013

"Año 55 de la Revolución"

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería Biomédica, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Tutor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

Pensamiento.

"El genio es un uno por ciento de inspiración y un noventa y nueve por ciento de transpiración"

Thomas Alva Edison

Dedicatoria.

A todos los que ayudaron para que esta tesis fuera posible y a los que prestaron su móvil para que pudiera trabajar.

Agradecimentos.

A mi novia por apoyarme

A mi tutor por soportarme

A mis amigos

Al profe Julián

A mi mamá y a mi papá por confiar en mí

Tarea técnica.

- Estudio de la documentación científica relacionada con el diseño de podómetros y sus posibles aplicaciones en Ingeniería Biomédica.
- Profundización en el estudio del lenguaje de programación Java.
- Análisis de las rutinas necesarias utilizadas en Eclipse para la realización de la aplicación.
- Realización de pruebas con el fin de evaluar el correcto desempeño de la aplicación.

Firma del Autor	Firma del Tutor

Resumen.

Las aplicaciones para dispositivos móviles forman parte del uso diario de muchas personas. Las que tienen aplicación en el campo de la ingeniería biomédica, como por ejemplo los podómetros, están ganando terreno, aunque, hasta donde sabemos, en nuestro país no se ha desarrollado una aplicación con tal propósito. En este trabajo se propuso implementar un podómetro en un dispositivo móvil, con el cual se pudiesen realizar estudios en el campo de la Ingeniería Biomédica. Con el desarrollo y posterior comprobación del correcto funcionamiento de la aplicación bajo diferentes condiciones, se logró implementar un podómetro con gran exactitud y precisión, ideal para realizar estudios relacionados con la marcha de las personas.

Tabla de contenido.

Introducción.

Un pedómetro es un aparato empleado para contar el número de pasos, y que indirectamente puede servir para medición de distancia, velocidad y cadencia del caminar de una persona [14].

A partir de la revisión la documentación relacionada con el tema, se ha podido constatar la existencia de gran cantidad de podómetros, los cuales se han utilizado en aplicaciones médicas, seguimiento del rendimiento en el deporte, en la industria automovilística y aérea como sistema de navegación inercial [6].

Gracias al avance actual de la tecnología no solo se han creado aparatos con el uso específico de podómetros, sino también se han desarrollado como aplicaciones para ser utilizadas en dispositivos móviles aprovechando los sensores que traen integrados los mismos.

A pesar de que la cantidad de pasos, desde el punto de vista de la actividad física, es un parámetro fundamental a la hora de analizar patrones de comportamiento, estilos de vida, monitoreo frecuente de personas que tienen afectada su capacidad motora, así como el rendimiento en la esfera deportiva, en nuestro país no contamos con podómetros de factura nacional, según la bibiliografía revisada. Por lo anteriormente expuesto proponemos la realización de una aplicación con la función de podómetro, de manera tal que podamos aprovechar los sensores propios de los teléfonos inteligentes sin tener que importar componentes para realizar específicamente un podómetro, lo que constituiría un mayor gasto económico. Dicha aplicación será implementada en el sistema operativo Android, ya que el mismo es de código libre y los softwares para el desarrollo de aplicaciones pueden ser descargados gratuitamente de Internet.

El objetivo general de este trabajo es implementar un podómetro en un dispositivo móvil.

Para el logro del objetivo general se plantean los siguientes objetivos específicos: analizar la documentación científica relacionada con el diseño de podómetros y sus aplicaciones, profundizar en el lenguaje de programación de Java, analizar las rutinas utilizadas en

Eclipse necesarias para implementar la aplicación y realizar pruebas bajo diferentes condiciones como vía para evaluar el correcto funcionamiento de la aplicación.

Organización del informe

El informe de la investigación consta con tres capítulos.

Capítulo 1: Introducción a los dispositivos móviles y los podómetros. Presenta una caracterización de los diferentes dispositivos móviles, en cuanto a sistemas operativos que usan y sensores que presentan. Luego se explica el funcionamiento de un podómetro así como sus aplicaciones.

Capítulo 2: Implementación del podómetro para el conteo de pasos. Constituye una explicación de la implementación de aplicaciones para el sistema operativo Android, y el entorno de desarrollo de estas. Además se muestran los códigos fundamentales usados en la programación del podómetro.

Capítulo 3: Discución de los resultados de la implementación de y evaluación del podómetro *i*-Walker. Se muestran los resultados de la evaluación del podómetro y la explicación de los mismos.

Capítulo 1: Introducción a los dispositivos móviles y a los podómetros.

Los dispositivos móviles son aparatos de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales [33].

1.1. Dispositivos portátiles y móviles, sensores incorporados y facilidades para desarrollo de aplicaciones.

Dado el variado número de niveles de funcionalidad asociado con dispositivos móviles, se adoptaron los siguientes estándares para la definición de dispositivos móviles: *dispositivo móvil de datos limitados*, los cuales tienen una pantalla pequeña , principalmente basada en pantalla de tipo texto con servicios de datos generalmente limitados a SMS y acceso WAP (Ej.: teléfonos móviles); *dispositivo móvil de datos básicos*, estos tienen una pantalla de mediano tamaño (entre 120 x 120 y 240 x 240 píxeles), menú o navegación basada en íconos por medio de una "rueda" o cursor, y que ofrecen acceso a e-mails, lista de direcciones, SMS, y un navegador web básico (Ej.: teléfonos inteligentes); *dispositivo móvil de datos mejorados*, dichos dispositivos que tienen pantallas de medianas a grandes (por encima de los 240 x 120 píxeles), navegación de tipo *stylus*, o simplemente pantalla táctil y que ofrecen las mismas características que el "dispositivo móvil de datos básicos" más aplicaciones nativas como aplicaciones de Microsoft Office Mobile (Word, Excel, PowerPoint) y aplicaciones corporativas usuales, en versión móvil, como Sap, portales intranet, etc [28]. Este tipo de dispositivos incluyen los Sistemas Operativos como Android, *i*OS, Symbian OS, BlackBerry OS, entre otros (Ej.: *i*-Phone, Samsung Galaxy, *i*-Pad), siendo estos dispositivos los más usados actualmente [53].

Android es un sistema operativo, que al ser de código libre y tener tantos desarrolladores perfeccionándolo constantemente sin necesidad de estar afiliados con alguna compañía, ha marcado la supremacía sobre los demás [1]. Además, gracias a que los *dispositivos móviles de datos mejorados* presentan una interfaz con facilidades para diferentes tipos de usuarios, y que también traen incorporados (en su mayoría) una amplia variedad de sensores, ha sido posible el desarrollo de un gran número de aplicaciones para los mismos, las cuales han sido implementadas en diferentes esferas (entretenimiento, salud, deportes, negocios, etc.),

dependiendo de su utilización. Un ejemplo de lo anterior, lo constituye el podómetro, usado para medir cantidad y cadencia de pasos. Este basa su funcionamiento en el acelerómetro, sensor de gran uso en estos dispositivos [14].

1.1.1. Netbooks.

Una netbook, es una categoría de computadora portátil de bajo costo y generalmente reducidas dimensiones, lo cual aporta una mayor movilidad y autonomía. Es utilizada principalmente para navegar por Internet y realizar funciones básicas como procesamiento de texto y de hojas de cálculo. Ejemplos de estas netbooks son: HP 2133 Mini-Note PC, Intel Classmate PC, Lanix Neuron LT10, Lenovo S10, LG X110, Samsung NC10 y Toshiba Satellite NB105.

Los sistemas operativos usados son Windows XP Home y Windows 7 Starter de Microsoft y otros sistemas operativos basados en GNU/Linux, entre los cuales varios de ellos han sido especialmente programados para su uso en estos ordenadores, como el Ubuntu Netbook Remix. Sin embargo, las *netbooks* no son lo suficientemente potentes para tareas como la edición de vídeos o para juegos de gráficos pesados, pero con ciertos programas emuladores de síntesis de imágenes y procesos de aligeramiento pueden ejecutarlos en un rango de calidad bajo-medio, y generalmente no tienen unidades ópticas (aunque se les puede conectar una unidad externa) [47].

Estas traen incorporados varios sensores, como de temperatura así como de movimiento, los cuales tienen como objetivo el monitoreo y protección del funcionamiento del equipo y no presentan facilidades para el desarrollo de aplicaciones que puedan llevarse a cabo por el usuario.

1.1.2. Tabletas.

Una tableta (del inglés: *tablet* o *tablet computer*) es un tipo de computadora portátil, de mayor tamaño que un smartphone o una PDA, integrado en una pantalla táctil (sencilla o multitáctil) con la que se interactúa primariamente con los dedos o una pluma *stylus* (pasiva o activa), sin necesidad de teclado físico ni ratón. Estos últimos se ven reemplazados por un teclado virtual y, en determinados modelos, por una mini-trackball integrada en uno de los bordes de la pantalla.

La tableta funciona como una computadora, solo que más orientado a la multimedia, lectura de contenidos y a la navegación web que a usos profesionales. Para que pueda leerse una memoria o disco duro externo USB, debe contar con USB OTG. Dependiendo del sistema operativo que implementen y su configuración, al conectarse por USB a un ordenador, se pueden presentar como dispositivos de almacenamiento, mostrando sólo la posible tarjeta de memoria conectada, la memoria flash interna e incluso la flash ROM. Por ejemplo, en Android el usuario debe de activar el modo de dispositivo de almacenamiento, apareciendo mientras como una ranura sin tarjeta [38]. Ejemplos de las tabletas son: AOC Breeze, Apple *i*Pad, Samsung Galaxy Tab, Motorola Xoom, HP TouchPad, bq readers Verne Plus, HTC Flyer, Huawei S7 y ZTE V9 [47].

Los sistemas operativos usados en las tabletas [19] son Android, *i*OS, webOS, Chrome OS, BlackBerry Tablet OS, Windows CE, Windows Phone y Windows 8, siendo Android quien lidera el mercado [11].

Las tabletas cuentan con varios sensores, como son las cámaras, sensores inerciales, acelerómetros, sensores fotoeléctricos, micrófonos, entre otros; estos no solo están destinados a un mejor desempeño del dispositivo, sino también a una interacción directa con el usuario, facilitándose así el desarrollo de aplicaciones que tienen como base fundamental de su funcionamiento dichos sensores. Aunque es necesario señalar que las mismas no son nada prácticas durante la realización de ciertas actividades, como correr o practicar algún ejercicio físico, y no son tan fáciles de llevar como otros dispositivos, entiéndase teléfonos inteligentes de última generación. Esto ha condicionado que las aplicaciones concebidas para estas actividades, no estén disponibles o no sean lo suficientemente específicas en las mismas [12].

1.1.3. Teléfonos inteligentes.

Teléfono inteligente (*smartphone* en inglés) es un término comercial para denominar a un teléfono móvil que ofrece la posibilidad de instalación de programas para incrementar el procesamiento de datos y la conectividad [20] [21]. Estas aplicaciones pueden ser desarrolladas por el fabricante del dispositivo, por el operador o por un tercero. El término "inteligente" hace referencia a cualquier interfaz, como un teclado QWERTY en miniatura,

una pantalla táctil, lo más habitual, o simplemente el sistema operativo móvil que posee, diferenciando su uso mediante una exclusiva disposición de los menús, teclas y atajos [39].

Casi todos los teléfonos inteligentes también permiten al usuario instalar programas adicionales, normalmente inclusive desde terceros, lo que dota a estos teléfonos de muchísimas aplicaciones en diferentes terrenos. Algunos ejemplos de teléfonos inteligentes son: serie MOTOBLUR de Motorola, serie S60 y Symbian^3 de Nokia, serie Optimus de LG, serie BlackBerry de Research In Motion, serie Galaxy, Wave y Omnia de Samsung, serie XPERIA de Sony Ericsson, serie Sense de HTC, serie Galaxy Nexus de Google/Samsung, serie iPhone de Apple, serie Nokia Lumia 900 de Nokia, entre otros [7].

Los sistemas operativos presentes en estos dispositivos son diversos, entre los cuales podemos encontrar: Android, *i*OS, Symbian OS, BlackBerry OS, Windows Phone, Linux embebido, Web OS, Bada, MeeGo, Java ME y Windows CE, imponiéndose Android como el más usado actualmente [18].

Entre otras características comunes, están la función de multitarea, el acceso a Internet vía WiFi o 3G, GPS, una cámara digital integrada, administración de contactos, algunos programas de navegación así como ocasionalmente la habilidad de leer documentos de negocios en variedad de formatos como PDF y Microsoft Office, y además incorporan gran variedad de sensores como acelerómetros, micrófonos, sensores de humedad, sensores de proximidad, sensores fotoeléctricos, sensores inerciales, y otros. La disponibilidad de estos sensores, y el hecho de que están destinados en su mayoría a la utilización directa por parte del usuario, han permitido que los teléfonos inteligentes sean los más usados, en cuanto al desarrollo de aplicaciones se trata, siendo la Ingeniería Biomédica uno de los campos beneficiados [40].

1.2. Podómetros y sus aplicaciones en Ingeniería Biomédica.

La cantidad de pasos visto desde la perspectiva de la actividad física de las personas es un parámetro fundamental a tener en cuenta a la hora analizar patrones de comportamiento, estilos de vida, monitoreo frecuente del estado de salud de las personas que se han visto afectadas o no por algún tipo de enfermedad, así como el chequeo del rendimiento en la

esfera deportiva; un dispositivo que podría significar de gran utilidad para la obtención de dicho parámetro es el podómetro [2].

Un podómetro es un aparato empleado para medir el número de pasos, y que indirectamente puede servir para medición de distancia, velocidad y cadencia del caminar de una persona. Tiene un sensor interno que es capaz de detectar el balanceo producido por cada zancada y registrarlo. Mediante la personalización de la distancia media de la zancada deduce aproximadamente distancias, velocidades, cadencias y las calorías usadas al caminar [37].

Con toda la tecnología actual disponible, hay una enorme variedad de podómetros para elegir. Todavía hay podómetros estándar que se enganchan en el cinturón de la persona y cuentan la distancia recorrida a partir del movimiento de la cadera. También hay podómetros que usan sistema de posicionamiento global (GPS, por sus siglas en inglés) para dar una lectura exacta de la distancia recorrida [3].

Actualmente los teléfonos inteligentes nos proporcionan la posibilidad de implementar podómetros como aplicaciones en ellos, gracias a que los mismos traen incorporado un acelerómetro que es el sensor que adquiere la información necesaria para el funcionamiento del podómetro [42] [43].

En Ingeniería Biomédica, el podómetro tiene una gran utilización como dispositivo de obtención de datos para el desarrollo de diversos estudios, como son los relacionados al análisis de la marcha, siendo útil para evaluar, planificar y tratar a personas con condiciones que afectan su capacidad para caminar [4]. La marcha patológica puede reflejar las compensaciones por patologías de base, o ser responsable de la causa de los síntomas en sí mismo. Parálisis cerebrales y pacientes con ictus son los más frecuentes en los laboratorios de la marcha. El estudio de la marcha permite diagnosticar y plantear estrategias de intervención a realizar, para permitir futuros desarrollos en ingeniería de rehabilitación [29].

Aparte de las aplicaciones clínicas, es también de uso general en la biomecánica deportiva para ayudar a atletas a optimizar y mejorar el rendimiento atlético y para identificar la postura relacionada, o problemas relacionados con el movimiento en personas con lesiones.

Puede utilizarse como un identificador biométrico para identificar a las personas individuales. Los parámetros se agrupan en espacio-temporal (longitud del paso, ancho de paso, la velocidad al caminar, el tiempo de ciclo) y cinemáticos (articulación de rotación de la cadera, la rodilla y el tobillo, la medida de ángulos de las articulaciones de la cadera / rodilla / tobillo y muslo / tronco / ángulos de pie). Existe una alta correlación entre la longitud del paso y la altura de una persona [41].

También puede utilizarse como la autenticación de dispositivos electrónicos portátiles. Y también, para las investigaciones de resbalones y caídas, superficie de resistencia al deslizamiento.

1.2.1. Bases sobre implementación de podómetros.

A Thomas Jefferson generalmente le es dado el crédito de inventar el podómetro moderno, aunque nunca solicitó una patente (principalmente se trata como una especulación histórica). A veces se utiliza el término de "Tomish meter", cuando un podómetro está enganchado al cinturón de una persona. Dentro de dicho dispositivo fue añadido peso, unas pelotas de metal que al balancearse sobre un péndulo eran utilizadas para medir la distancia que alguien se desplazaba. El movimiento era generado por el balanceo de las caderas, al caminar la persona; este dispositivo, aunque no muy exacto, daba un cálculo aproximado de cuán lejos había caminado la persona [50].

Los podómetros aumentaron cuando la tecnología mejoró, y durante la década de 1930 se hicieron más populares. Estos podómetros todavía operaban bajo el mismo principio básico del movimiento del péndulo para medir distancia. En América era muy popular entre los corredores. Sin embargo, las mediciones no eran muy exactas.

Cuando los dispositivos cambiaron a la operación electrónica, durante la revolución de la computadora, el podómetro, originalmente un dispositivo analógico, cambió al sistema de operación por computadora para el registro de información. Esto quería decir que los podómetros, con menos peso, podían mantener una mayor exactitud en la distancia medida. Sin embargo, estos podómetros perdieron importancia en comparación con los modernos, los podómetros digitales [46].

En la actualidad podemos encontrar dispositivos diseñados específicamente como podómetros, como por ejemplo:

- *Pedometer Distance-Measuring Device*, diseñado por John C. Sherrill. Este podómetro electrónico está constituido por una calculadora de mano y un interruptor cerrado para el balance del peso en respuesta a la persona que va a utilizarlo.
- *Electronic Pedometer with step sensor in removeble insole*, diseñado por Kasuhiko Yukawa. En este caso el podómetro esta insertado en una suela de zapato.
- *Pulse meter with pedometer function*, diseñado por Chiaki Nakamura. En este dispositivo se combina un contador de pulso electrónico y un podómetro, los que pueden ser utilizados a partir de un solo sensor, usado para determinar los pasos al caminar y la razón de pulso.

A partir del auge que han tenido los teléfonos inteligentes, y gracias a las posibilidades técnicas que nos brindan es cada vez más frecuente encontrar podómetros implementados en dichos teléfonos [19], estos son algunos de los que podemos encontrar:

- *Desicion Manager:* muestra la cantidad de pasos, velocidad mínima y máxima, tiempo de duración del recorrido, además nos va diciendo los parámetros en un intervalo que es ajustado por el usuario [17].
- *Our pedometer:* incluye cambio de parámetros como la sensibilidad, razón de actualización, comienzo automático, tiempo en que deseamos comenzar, a partir de esto nos muestra cantidad de pasos recorridos y velocidad estimada [44].
- *Pedometer:* permite tener creadas varias secciones, así como el cambio de los parámetros para la realización de la medición, y nos muestra cantidad de pasos, velocidad estimada y calorías quemadas [52].

1.2.1.1. Diagrama en bloques de un podómetro.

Un podómetro está compuesto básicamente por varios bloques funcionales (Figura 1), a los cuales se les pueden incorporar varios elementos tanto funcionales como de procesamiento de acuerdo al nivel de exactitud que se desea lograr, así como en la aplicación que se quiera implementar.

Figura 1: Diagrama en bloques.

Lo obtenido a la salida del sensor es introducido en una unidad de “discriminación”, de ahí a una de operación lógica, donde son procesados los datos y nos da la distancia total que se desplazó. La unidad de operación lógica controla una unidad de visualización que muestra la cantidad de pasos o la distancia recorrida, de acuerdo a la instrucción ingresada por el usuario. La unidad de “reset” es acoplada a la unidad de operación lógica para dar la instrucción de borrar la señal en la unidad de visualización. La unidad de “discriminación” está diseñada para ignorar los posibles ruidos o las pulsaciones irregulares en los contactos del sensor, una fuente de radio, o cualquier otra causa desconocida, permitiendo pasar solamente los verdaderos indicadores del paso a la unidad de operación lógica [49].

1.2.2. Fuentes de errores potenciales en la estimación de los podómetros.

El proceso de caminar está modulado o modificado por muchos factores, y los cambios que imprimen en el patrón de marcha habitual pueden ser transitorios o permanentes [45]. De no ser tomados en cuenta, estos factores constituirían una fuente de error potencial en la medición obtenida por el podómetro [24]. Los factores pueden ser de diversos tipos:

- Extrínsecos: El terreno, el calzado, la vestimenta, el transporte de carga.
- Intrínsecos: El sexo, peso, altura, edad [5] [25] [34].
- Físicos: El peso, la talla, la constitución física [13].
- Psicológicos: La personalidad, las emociones [32].
- Fisiológicos: Las características antropométricas [26].
- Patológicos: Traumatismos, patologías neurológicas, músculo esquelético, trastornos psiquiátricos [30] [31].

Los parámetros que se tiene en cuenta para el análisis de la marcha [15] [16] son:

- Longitud del paso.
- Longitud de la zancada [9].
- Cadencia.
- Velocidad.
- Base dinámica.
- Línea de progresión [27].
- Angulo del pie [23].

En nuestro caso, tomaremos en cuenta la longitud del paso como referencia para el cálculo de la cantidad de pasos.

1.2.3. Disponibilidad de aplicaciones de podómetros.

Actualmente existe una gran variedad de podómetros implementados en diferentes modelos de teléfonos inteligentes y con base en diferentes sistemas operativos, pero esto no facilita su adquisición, puesto que casi la totalidad de estas aplicaciones son situadas en sitios web como AppleStore y AndroidMarket, en los cuales para realizar la descarga de las mismas es necesario pagar, esto se debe a que en su gran mayoría son creadas por compañías dedicadas a la industria del software o terceros los cuales tienen como empleo la realización de aplicaciones [52].

1.3. Conclusiones parciales.

Como se ha podido apreciar, existe gran diversidad de dispositivos portátiles y móviles (*netbooks*, tabletas, teléfonos inteligentes), los cuales son de gran ayuda para nuestro desempeño en varias tareas, destacándose los teléfonos inteligentes, gracias a su fácil manejo y por la amplia gama de sensores que traen incorporados. A su vez estos corren sobre varios sistemas operativos (Android, *i*OS, Symbian OS, BlackBerry OS, etc.), que son los que permiten el uso y el propio rendimiento del dispositivo, siendo Android el más usado, y uno de los que más posibilidades brinda, al ser este un software libre y de código abierto.

El desarrollo de aplicaciones en los teléfonos inteligentes ha estado marcado propiamente por el desarrollo de los mismos, dado esto generalmente por el gran número de sensores que presentan y la posibilidad de ser usados por el usuario a su “antojo” (condicionado propiamente por el sistema operativo), siendo los acelerómetros uno de los más utilizados; ejemplo de estas aplicaciones lo constituyen los podómetros.

También se ha podido constatar que los podómetros tienen gran uso en el ámbito de la Ingeniería Biomédica, poniéndose esto de manifiesto tanto en aplicaciones clínicas como en la biomecánica deportiva; además que actualmente no solo se usan dispositivos diseñados específicamente como podómetros para la realización de estudios, sino que son utilizados teléfonos inteligentes mediante aplicaciones que cumplen dicha función.

Capítulo 2: Implementación del podómetro para el conteo de pasos.

En este capítulo se hará refencia a la implementación de aplicaciones para Android, así como el entorno para su desarrollo, los códigos principales de la programación de aplicación, además de proponer una metodología para la evaluación del podómetro.

2.1. Implementación de aplicaciones para el sistema operativo Android.

El desarrollo de aplicaciones para Android no requiere aprender lenguajes complejos de programación. Todo lo que se necesita es un conocimiento aceptable de Java y estar en posesión del kit de desarrollo de software o «SDK» provisto por Google el cual se puede descargar gratuitamente [36].

Las aplicaciones se desarrollan habitualmente en el lenguaje Java con Android Software Development Kit (Android SDK), pero están disponibles otras herramientas de desarrollo, incluyendo un Kit de Desarrollo Nativo para aplicaciones o extensiones en C o C++, Google App Inventor, un entorno visual para programadores novatos, aunque muchos recomiendan el software Eclipse porque brinda un entorno cómodo para el programador y una gran cantidad de herramientas [51].

Luego de programar la aplicación, los archivos son compilados con la extensión .apk, es decir, un paquete para el sistema operativo Android. Este formato es una variante del formato JAR de Java y se usa para distribuir e instalar componentes empaquetados para la plataforma Android para teléfonos inteligentes y tabletas.

Un archivo .apk normalmente contiene lo siguiente:

- AndroidManifest.xml
- classes.dex
- resources.arsc
- res (carpeta)
- META-INF (carpeta)

El formato APK es básicamente un archivo comprimido ZIP con diferente extensión por lo cual pueden ser abiertos e inspeccionados usando un software archivador de ficheros como 7-Zip, WinZip, WinRar o Ark.

2.2.1. Entorno de desarrollo del Eclipse.

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus [48].

La base para Eclipse es la Plataforma de cliente enriquecido (del Inglés Rich Client Platform RCP). Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal - inicio de Eclipse, ejecución de plugins.
- OSGi - una plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT) - Un widget toolkit portable.
- JFace - manejo de archivos, manejo de texto, editores de texto.
- El Workbench de Eclipse - vistas, editores, perspectivas, asistentes.

Los widgets de Eclipse están implementados por una herramienta de widget para Java llamada SWT, a diferencia de la mayoría de las aplicaciones Java, que usan las opciones estándar Abstract Window Toolkit (AWT) o Swing. La interfaz de usuario de Eclipse también tiene una capa GUI intermedia llamada JFace, la cual simplifica la construcción de aplicaciones basadas en SWT.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas,

las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente, al permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros lenguajes de programación.

En cuanto a las aplicaciones clientes, Eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plugin de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. Mediante diversos plugins, estas herramientas están también disponibles para otros lenguajes como C/C++ (Eclipse CDT) y en la medida de lo posible para lenguajes de script no tipados como PHP o Javascript. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio plano para archivos, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente [54].

2.2.2. Manejo de la señal del acelerómetro.

Cuando tratamos de determinar cuan lejos ha caminado una persona, hay información diponible para nosotros. Cuando la persona camina, hay movimiento en el eje Z (vertical) del cuerpo con cada paso. Una manera simple de medir la distancia caminada es usar el

movimiento en el eje Z para determinar la cantidad de pasos dados y luego multiplicar estos pasos por el promedio de la longitud de la zancada [8].

Un algoritmo común usa para el conteo de pasos algunas maneras de detección de picos. Generalmente, las muestras son tomadas:

- *n* es el número de pasos caminados.
- *k* es una constante para la conversión (pies o metros recorridos).

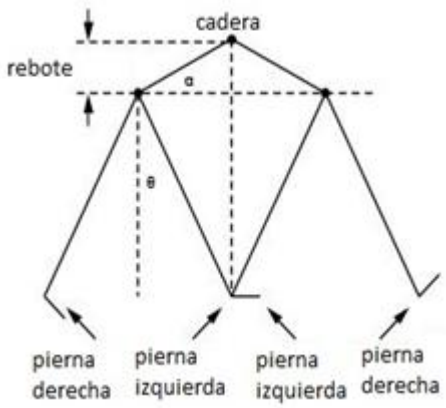

Figura 2: Movimiento vertical de la cadera mientras caminamos.

El mantener el acelerómetro cerca del cuerpo es importante para mantener la exactitud. Un algoritmo adaptable que "aprenda" de las características de la zancada del usuario puede mejorar significativamente la exactitud [22].

La señal que nos brinda el acelerómetro puede ser usada a partir de una programación que se realiza en el entorno de desarrollo de la aplicación, luego cuando la aplicación es corrida en nuestro dispositivo móvil, ella adquiere la señal del sensor haciendo uso de los comandos ingresados durante la programación. La forma en que sucede esto en el dispositivo varía entre uno y otro, dependiendo de la arquitectura de los mismos [10].

2.2.3. Diseño de la interfaz de usuario.

La aplicación debe contar con la visualización de los elementos propios de un podómetro, es decir, cantidad de pasos, distancia recorrida y velocidad.

Por tanto, la aplicación contará con una ventana principal donde aparecerán los elementos anteriores, de forma tal que puedan ser visualizados por el usuario (Figura 3), y con una ventana secundaria que mostrará el menú de configuración para la corrida de la aplicación, ajustable para cada usuario (Figura 4). En estos menús se podrá variar la sensibilidad del sensor, la manera en que será corrida la aplicación en el dispositivo, así como la longitud de la zancada del sujeto, la que será usada para el cálculo posterior de la distancia recorrida y la velocidad.

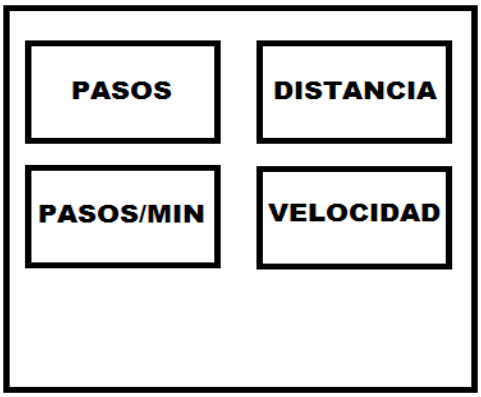

Figura 3: Ventana principal de la aplicación.

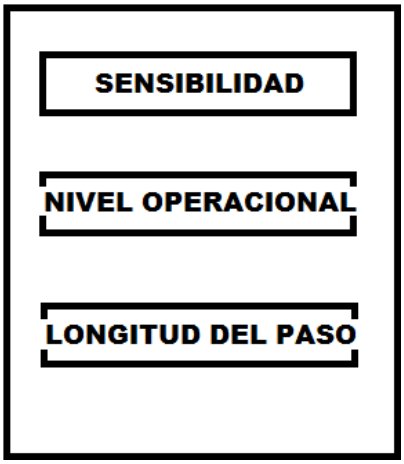

Figura 4: Ventana secundaria de la aplicación.

2.3. Código del programa y habilitación de la aplicación por el usuario.

2.3.1. Procedimientos para crear la aplicación.

Primeramente se debe crear un proyecto de Android, esto se realiza seleccionando *AndroidProyect* en *File/New/AndroidProyect* desde el Eclipse. Esto abrirá una ventana en

la cual se procede a especificar el nombre del proyecto y a seleccionar la versión (*target*) de Android deseada, en nuestro caso usaremos Android 2.3. Luego se le da nombre a la aplicación, se define el paquete Java por defecto para las clases y el nombre de la clase principal. Esta acción creará toda la estructura de carpeta necesaria para compilar un proyecto Android (Figura 4).

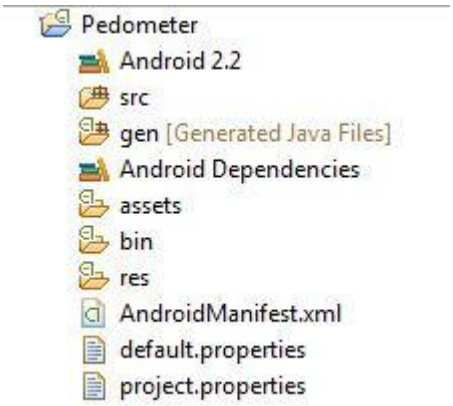

Figura 5: Carpetas generadas para un proyecto de Android.

Dentro de la carpeta src desarrollaremos el código fuente de nuestra aplicación, es decir, código de la interfaz gráfica, clases auxiliares, etc.

La carpeta res contendrá los ficheros de recursos necesarios para el proyecto, como son imágenes y cadenas de texto.

El fichero AndroidManifest.xml contendrá la definición en XML de los aspectos principales de la aplicación, como su identificación, sus componentes y permisos necesarios para su ejecución.

2.3.2. Comandos principales.

A continuación se muestran los comandos principales usados en la programación de la aplicación. Es necesario aclarar que para la realización de estos códigos se partió incialmente de determinados segmentos que fueron adquiridos en Google Codelos, diseñados principalmente por Levente Bagi.

El código del contador de pasos requiere diversos bloques.

```
package wk.tesis.podometro;
import java.util.ArrayList;
import android.hardware.Sensor;
```

```
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;
```

Se incluye la declaración de la clase y de las variables asociadas, como se ilustra en el segmento siguiente.

```
public class StepDetector implements SensorEventListener
{
 private final static String TAG = "StepDetector";
 private float mLimit = 10;
 private float mLastValues[] = new float[3*2];
 private float mScale[] = new float[2];
 private float mYOffset;

 private float mLastDirections[] = new float[3*2];
 private float mLastExtremes[][] = { new float[3*2], new float[3*2] };
 private float mLastDiff[] = new float[3*2];
 private int mLastMatch = -1;

 private ArrayList<StepListener> mStepListeners = new ArrayList<StepListener>();
```

Luego se indica la gravedad, entiéndase acelereación de la gravedad, como referencia de las mediciones de aceleración relativas a ella.

```
 public StepDetector() {
 int h = 480;
 mYOffset = h * 0.5f;
 mScale[0] = - (h * 0.5f * (1.0f / (SensorManager.STANDARD_GRAVITY * 2)));
 mScale[1] = - (h * 0.5f * (1.0f / (SensorManager.MAGNETIC_FIELD_EARTH_MAX)));
 }
```

Se asigna un valor para la sensibilidad del sensor, que puede ser modificado.

```
 public void setSensitivity(float sensitivity) {
 mLimit = sensitivity; // 1.97 2.96 4.44 6.66 10.00 15.00 22.50 33.75 50.62
 }

 public void addStepListener(StepListener sl) {
 mStepListeners.add(sl);
 }
```

El análisis del movimiento detectado por el sensor, es decir, si es un cambio de orientación o si se está caminando, se realiza con las instrucciones siguientes. Si se está caminando, se comienza a detectar el paso.

```
 //public void onSensorChanged(int sensor, float[] values) {
 public void onSensorChanged(SensorEvent event) {
 Sensor sensor = event.sensor;
 synchronized (this) {
 if (sensor.getType() == Sensor.TYPE_ORIENTATION) {
 }
 else {
 int j = (sensor.getType() == Sensor.TYPE_ACCELEROMETER) ? 1 : 0;
 if (j == 1) {
```

```
 float vSum = 0;
 for (int i=0 ; i<3 ; i++) {
 final float v = mYOffset + event.values[i] * mScale[j];
 vSum += v;
 }
 int k = 0;
 float v = vSum / 3;

 float direction = (v > mLastValues[k] ? 1 : (v < mLastValues[k] ? -1 : 0));
 if (direction == - mLastDirections[k]) {
```

Si hay cambio de dirección, se sigue el siguiente bloque.

```
 int extType = (direction > 0 ? 0 : 1); // minumum or maximum?
 mLastExtremes[extType][k] = mLastValues[k];
 float diff = Math.abs(mLastExtremes[extType][k] - mLastExtremes[1 -
extType][k]);

 if (diff > mLimit) {

 boolean isAlmostAsLargeAsPrevious = diff > (mLastDiff[k]*2/3);
 boolean isPreviousLargeEnough = mLastDiff[k] > (diff/3);
 boolean isNotContra = (mLastMatch != 1 - extType);

 if (isAlmostAsLargeAsPrevious && isPreviousLargeEnough && isNotContra) {
 Log.i(TAG, "step");
 for (StepListener stepListener : mStepListeners) {
 stepListener.onStep();
 }
 mLastMatch = extType;
 }
 else {
 mLastMatch = -1;
 }
 }
 mLastDiff[k] = diff;
 }
 mLastDirections[k] = direction;
 mLastValues[k] = v;
 }
 }
 }
 }
 public void onAccuracyChanged(Sensor sensor, int accuracy) {
 // TODO Auto-generated method stub
 }
}
```

2.3.3. Metodología para correr la aplicación.

Primeramente para poder ejecutar la aplicación, es necesario copiarla al dispositivo y luego instalarla.

Una vez instalada la aplicación procedemos a ejecutar la misma. Cuando ya tenemos la aplicación abierta debemos presionar el botón de **menú** y luego el de **configuración**. De esta manera accedemos al menú de configuración del podómetro. Para configurar el podómetro, es necesario ajustar la sensibilidad del sensor, la cual presenta una escala del 1 al 9 (de menor a mayor sensibilidad), aunque es recomendable usar una sensibilidad de 6, presionando **sensibilidad**. También podemos elegir de qué manera deseamos correr la aplicación, presionando en **nivel operacional**, de manera que podemos seleccionar si la aplicación correrá en un primer plano o si se mantendrá ejecutándose en el fondo mientras realizamos otras tareas con el dispositivo. Además, se debe ingresar la longitud de la zancada para el cálculo de la velocidad y de la distancia recorrida, esto lo realizamos presionando **longitud del paso**. Luego de realizar la configuración regresamos a la ventana principal y reiniciamos la aplicación, presionando **menú** y luego **reiniciar**.

Si se quiere detener la aplicación, sin tener que salir de la misma, presionamos **menú** y luego en **pausa**, y para volver a ejecutarla presionamos **menú** y luego en **inicio**. Para abandonar completamente la aplicación presionamos **menú** y luego **quitar**.

2.4. Metodología propuesta para la evaluación del podómetro.

Para poder utilizar la aplicación es necesario primeramente realizar una comprobación del funcionamiento adecuado de la misma, teniendo en cuenta que esta deberá ser usada con cualquier persona, en dependencia del estudio a realizar.

2.4.1. Diseño del experimento.

Primeramente, seleccionamos una muestra de personas, las cuales, luego de ubicarles correctamente el dispositivo, se pondrán a caminar una distancia de 50 m. Externamente, otra persona contará los pasos realizados durante la caminata, los cuales serán usados para corroborar los resultados obtenidos en la medición con el podómetro.

Los datos obtenidos tanto por la persona como por el podómetro, servirán para determinar la desviación estándar, el error, absoluto y relativo, cometido con nuestro dispositivo. También permitirá estimar una medida de cuan fiable será la medición que realiza el podómetro.

- Cálculo de la desviación estándar:

$$Desviación\ Estándar = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(Xi - \overline{X})^2}, \quad (1)$$

donde:

Xi = dato.

$\overline{X}$ = promedio.

i = número de datos para sacar la desviación estándar.

N = cantidad de mediciones.

- Cálculo del error absoluto:

$$Error\ absoluto = Yn - Xn, \quad (2)$$

donde:

Yn = valor esperado.

Xn = valor medido.

- Cálculo del error relativo:

$$Error\ relativo = \left|\frac{Yn - Xn}{Yn}\right| x\ 100\%, \quad (3)$$

donde:

Yn = valor esperado.

Xn = valor medido.

Posteriormente seleccionamos dos personas, a las cuales se les realizarán varias mediciones, con los mismos requisitos de la prueba anterior, pero para un número fijo de pasos, esto nos permitirá determinar la precisión y exactitud de nuestro dispositivo.

- Cálculo de la precisión:

$$Precisión = 1 - \left| \frac{Xn - \overline{Xn}}{\overline{Xn}} \right| x\ 100\% \quad , \tag{4}$$

donde:

Xn = un valor de la medición.

$\overline{Xn}$ = promedio de las mediciones.

- Cálculo de la exactitud:

$$Exactitud = 1 - \left| \frac{Yn - Xn}{Yn} \right| x\ 100\% \quad , \tag{5}$$

donde:

Yn = valor esperado.

Xn = valor medido.

2.4.1.1. Número de sujetos, sus características y entornos para la prueba.

Para la realización del experimento es necesario tomar una muestra de sujetos que cubran el mayor rango posible de personas, es decir, que incluya una variación de los parámetros intrínsecos (sexo, edad), físicos (talla, peso, constitución física) y fisiológicos (características antropométricas) de las mismas.

Para ello la muestra debe ser de al menos 60 sujetos, teniendo en cuenta que debe existir diferencias entre las mismas (expuestas anteriormente), lo que nos permitirá realizar mediciones en los posibles tipos de sujeto que usarán la aplicación y así lograr una mejor fiabilidad de la evaluación.

En cuanto a la elección del entorno, la prueba debe ser realizada en un terreno llano, ya que la aplicación está diseñada fundamentalmente con el objetivo de analizar patrones de marcha.

2.5. Conclusiones parciales.

Hemos podido apreciar de qué manera se realizó la implementación del *i*-Walker, partiendo de cómo se desarrollan las aplicaciones para el sistema operativo Android, analizando los medios usados y recomendando el software Eclipse para realizar aplicaciones.

También se explicó el entorno de desarrollo del Eclipse, junto con las herramientas que dentro del software son usadas en la programación de la aplicación. Por otra parte se mostró la interfaz de usuario que debe presentar la aplicación, se mostraron los principales códigos usados en la programación y la metodología para ejecutar la misma por parte del usuario. Finalmente se propuso una metodología para la evaluación del podómetro y se diseñó el experimento para llevar a cabo la misma.

Capítulo 3: Discusión de los resultados de la implementación y evaluación del podómetro *i*-Walker.

En este capítulo se mostrará la aplicación finalizada, el funcionamiento de la misma, función de los botones y los parámetros de la configuración, así como los resultados obtebidos con la misma durante el proceso de evaluación.

3.1. Interfaz de usuario: funciones de los botones.

A continuación se muestra la interfaz de usuario del *i*-Walker ya implementada, junto con la función de cada uno de sus botones.

- Ventana principal:

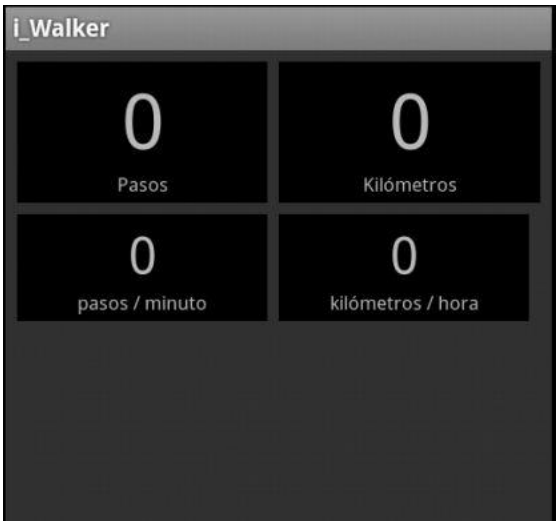

Figura 6: Ventana principal.

- Panel de control:

Figura 7: Panel de control.

- Menú de configuración:

Figura 8: Menú de configuración.

En la ventana principal se muestran los resultados correspondientes al conteo de pasos, la distancia recorrida, la cantidad de pasos por minutos y la velocidad desarrollada.

Cuando presionamos el botón de menú (propio del dispositivo) se despliega el panel de control, con los siguientes botones:

- Pausa o Inicio: permite detener o iniciar la puesta en marcha de la aplicación.
- Configuración: accede al menú de configuración del *i*-Walker.
- Reiniciar: reinicia el valor de los resultados, sin afectar la configuración realizada usuario.
- Quitar: cierra completamente la aplicación.

Al acceder al menú de configuración del *i*-Walker observamos una ventana en la cual se muestran los parámetros ajustables correspondientes al podómetro, los cuales necesariamente deben ser ajustados independientemente para cada usuario:

- Sensibilidad: nos permite ajustar la sensibilidad del sensor, variable desde 1 a 9.
- Nivel operacional: nos permite seleccionar el funcionamiento del sensor, ya que el mismo puede variar al apagarse la pantalla.
- Longitud del paso: nos permite introducir la longitud de nuestra zancada, para la realización del cálculo de la distancia y la velocidad.

3.2. Resultados de la evaluación del podómetro *i*-Walker.

Luego de realizar todas las mediciones con los 60 sujetos de prueba, se realizó un análisis de las muestras para verificar si los resultados obtenidos en la medición con el podómetro tenían relación con los valores reales correspondientes a cada sujeto. Para esto se realizó en MATLAB la prueba de los rangos signados de Wilcoxon, con un valor de α=0,05, haciendo uso de la función `signrank`. Se obtuvo un valor de p = 0,09 (resultado que devuelve la prueba), por tanto al ser $p > \alpha$, se demuestra que las dos muestras (reales y medidas) son iguales poblacionalmente.

3.2.1 Medición de pasos (pasos vs. sujetos, para patrón y medido con el *i*-Walker).

Las muestras reales y las medidas por el podómetro fueron puestas en una gráfica (Figura 9) de forma tal que se pueda realizar una mejor comprensión de los resultados obtenidos. En la misma se puede apreciar como una muestra se superpone casi en su totalidad con la otra, esto se debe a que la medición solo varió en el rango de ±2 pasos en algunos sujetos.

También se realizó el cálculo del error absoluto (Figura 10) obteniéndose para los mismos una desviación promedio de 1,08 pasos y una desviación estándar de 0,78 pasos. Analizando la gráfica correspondiente, podemos observar que el máximo error cometido fue de 3 pasos, apreciable solamente en dos sujetos, en 15 sujetos el error fue de 2 pasos, en 29 sujetos el error fue de 1 paso y en 14 sujetos el conteo fue exacto; todo esto con una tendencia al error negativo, es decir, al conteo de pasos de más.

Cuando analizamos la gráfica realizada con los valores del error relativo (Figura 11) se puede apreciar que este es siempre menor que el 2% y en la mayoría no sobrepasa el 1,5%. En esta gráfica se hace más evidente la tendencia al error negativo, lo que se debe al movimiento brusco al detenerse realizado por la gran mayoría de los sujetos, el cual es detectado por el sensor como otro paso.

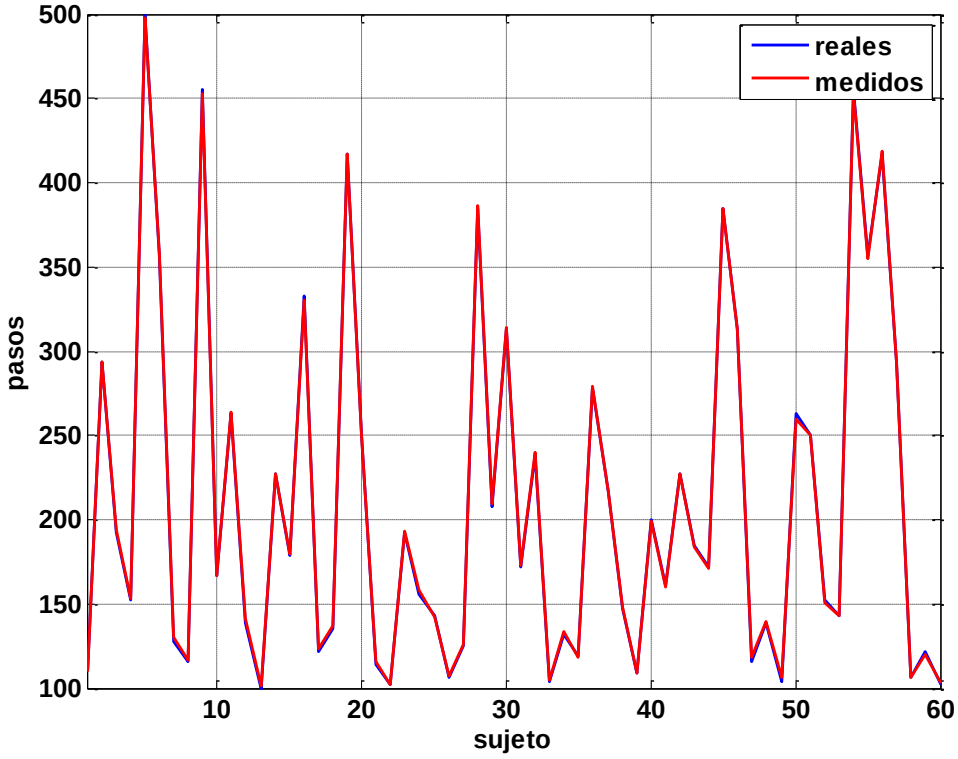

Figura 9: Muestras reales y medidas.

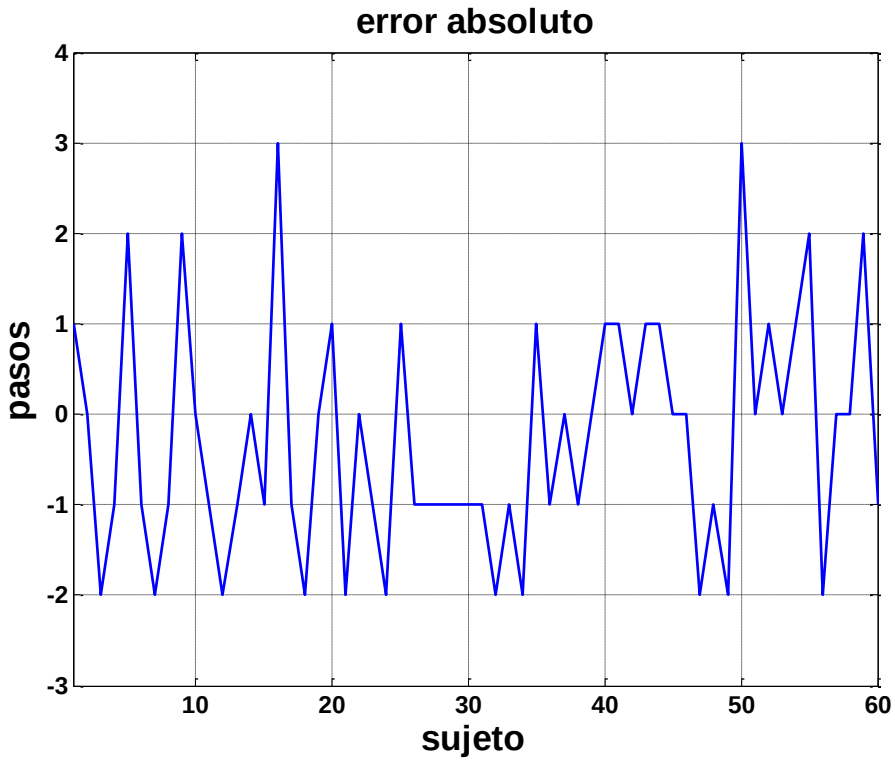

Figura 10: Error absoluto.

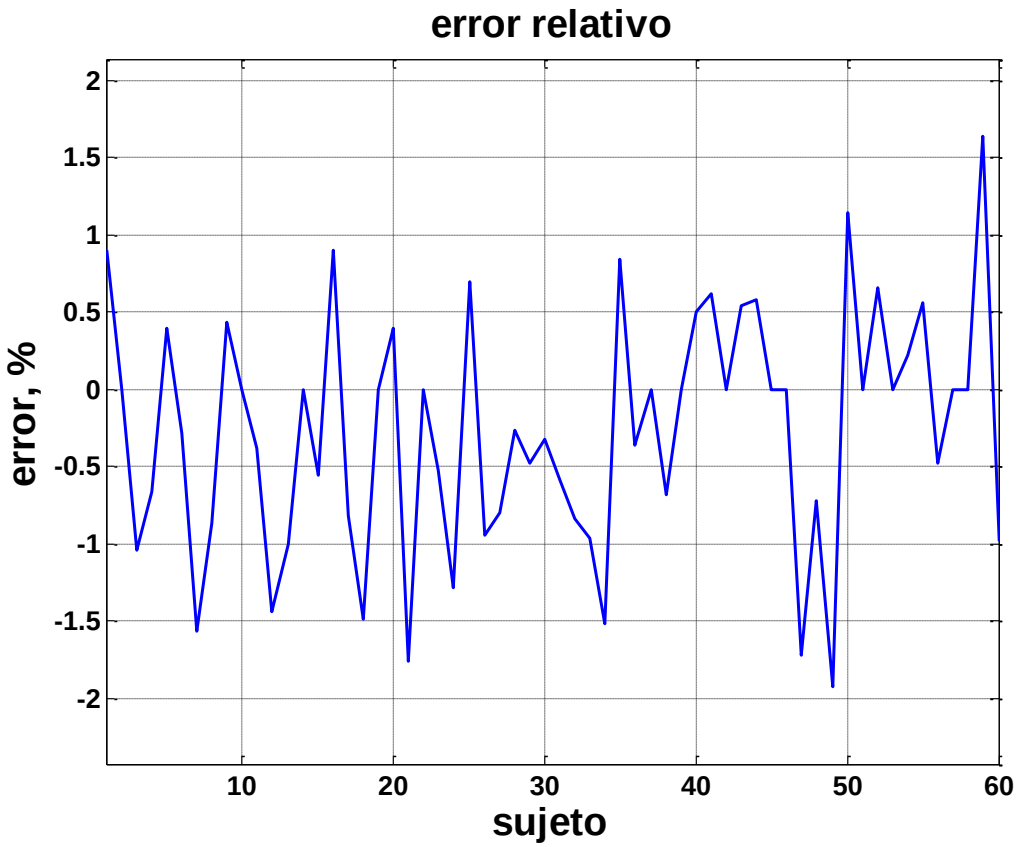

Figura 11: Error relativo

También se realizó la evaluación de la precisión y la exactitud del podómetro; para ello se eligieron dos sujetos con características diferentes, a los cuales se le realizó 50 mediciones para 200 pasos, en cada uno de los casos, con los datos obtenidos, se llevó a cabo el cálculo de la precisión y la exactitud.

Para el primer sujeto se obtuvo una precisión de 99,60%, y en la Figura 12 se muestran los valores de exactitud correspondientes a cada medición. Se obtuvo un promedio de exactitud de 99,52%.

Para el segundo sujeto se alcanzó una precisión de 99,52%. Se puede apreciar en la Figura 13 los valores de exactitud obtenidos en cada medición, para los cuales se tiene un promedio 99,48% de exactitud.

Con estos valores de precisión y exactitud se puede constatar el excelente desempeño del *i*-Walker.

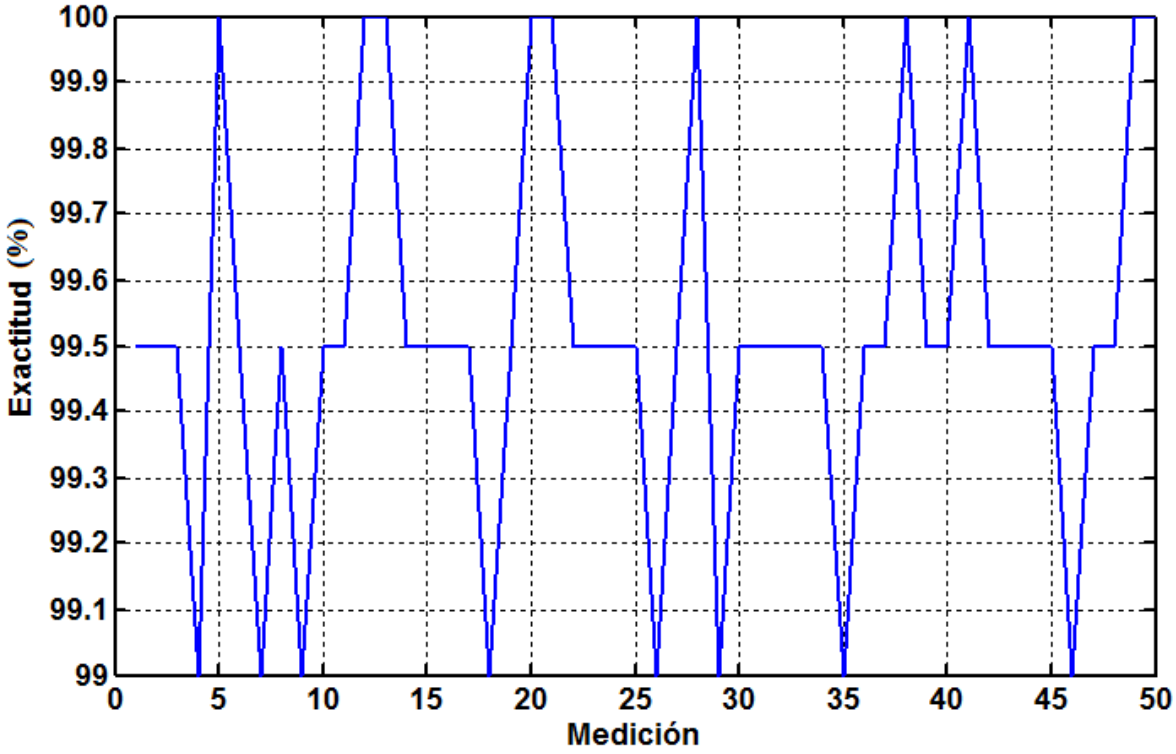

Figura 12: Exactitud para sujeto 10.

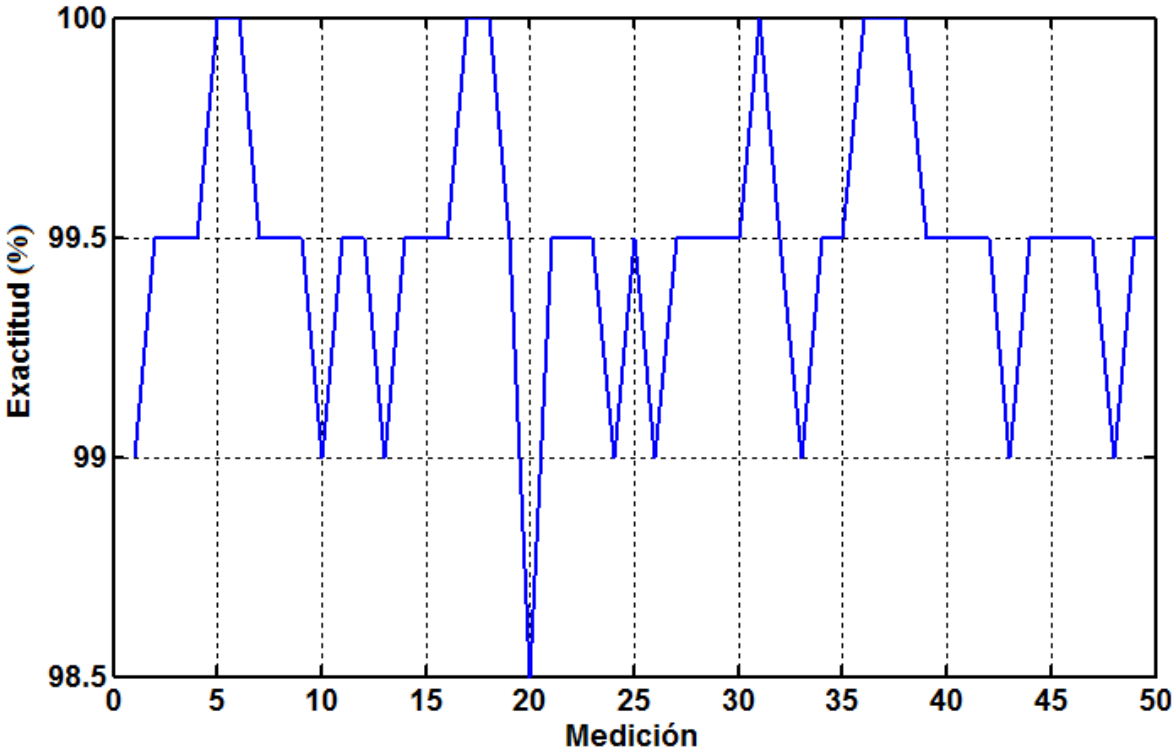

Figura 13: Exactitud para sujeto 40.

3.2.2. Medición de la distancia y la velocidad.

La obtención de la distancia y la velocidad desarrollada, dependen directamente de la longitud del paso introducido por el usuario. Entonces, a partir de esta información y con la cantidad de pasos contados, se realiza el cálculo de la distancia y la velocidad. Este cálculo forma parte de la programación del podómetro (ver anexos), es decir, que el sensor de nuestro dispositivo solo cuenta los pasos, detectando los picos en el movimiento al caminar, y no la distancia o la velocidad a la que nos movemos.

3.4. Conclusiones parciales.

A lo largo del capítulo se han analizado los resultados obtenidos en la evaluación del podómetro *i*-Walker, explicando en primer lugar las funciones de los elementos presentes en la aplicación.

Primeramente se realizó la prueba de rangos signados de Wilcoxon, para determinar la igualdad poblacional de las muestras (con los datos obtenidos). Luego se procedió al análisis de las gráficas correspondientes al error relativo y absoluto, donde se muestra que el error cometido es muy bajo y reflejándose la tendencia sobre el error negativo debido al "paso falso" detectado por el sensor cuando el sujeto termina la caminata.

También se llevó a cabo la evaluación de la precisión y exactitud del dispositivo, determinándose para ambos casos un 99%, quedando reflejado cuan fiable es la medición que realiza el *i*-Walker.

Conclusiones.

- El análisis de toda la documentación correspondiente, permitió conocer con detalle el funcionamiento de los podómetros.
- A partir del estudio del lenguaje de programación Java, se logró realizar las rutinas necesarias en Eclipse para el desarrollo de la aplicación.
- La implementación de la aplicación *i*-Walker 1.0 se logró realizar correctamente.
- La realización de pruebas bajo diferentes condiciones, permitió evaluar y corregir la exactitud del podómetro.

Recomendaciones.

- Incrementar a la aplicación el cálculo de nuevos parámetros, como el gasto calórico.
- Poner notificación por voz en la aplicación.

Referencias.

[1] Agrafiotis, V., J.-P. Hubaux, et al. (2012). "Private Information Exposure in Online Social Networks with iOS, Android and Maemo Mobile Devices."

[2] Åkerberg, A., M. Lindén, et al. (2012). "How Accurate are Pedometer Cell Phone Applications?" Procedia Technology **5**: 787-792.

[3] Ali, S. and B. George (2012). A portable pedometer based on inductive proximity. Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on, IEEE.

[4] Aoyagi, Y. and R. J. Shephard (2011). "A model to estimate the potential for a physical activity-induced reduction in healthcare costs for the elderly, based on pedometer/accelerometer data from the Nakanojo Study." Sports Medicine **41**(9): 695-708.

[5] Aoyagi, Y. and R. J. Shephard (2013). "Sex differences in relationships between habitual physical activity and health in the elderly: Practical implications for epidemiologists based on pedometer/accelerometer data from the Nakanojo Study." Archives of gerontology and geriatrics **56**(2): 327-338.

[6] Appelberg, A., I. Dohrn, et al. (2012). "High agreement between self-reported pedometer steps and accelerometer derived steps in an elderly population." Journal of Science and Medicine in Sport **15**: S294.

[7] Asthana, A. and R. Asthana (2012). "IOS 5, Android 4.0 and Windows 8–A Review." IEEE Code of Ethics.

[8] Barreira, T., C. Tudor-Locke, et al. (2013). "Comparison of GT3X Accelerometer and YAMAX Pedometer Steps/Day in a Free-Living Sample of Overweight and Obese Adults." Journal of physical activity & health **10**(2): 263-270.

[9] Barreira, T. V., R. M. Brouillette, et al. (2012). "Comparison of Older Adults' Steps/Day Using NL-1000 Pedometer and Two GT3X+ Accelerometer Filters." Journal of aging and physical activity.

[10] Barreira, T. V., C. Tudor-Locke, et al. (2011). "Comparison Of Yamax Pedometer And Gt3x Accelerometer Steps In A Free-living Sample: 2535: Board#

143 June 3 9: 00 AM-10: 30 AM." Medicine & Science in Sports & Exercise **43**(5): 696.

[11] Barrera, D. and P. Van Oorschot (2011). "Secure software installation on smartphones." Security & Privacy, IEEE **9**(3): 42-48.

[12] Bass, M. and M. Varela Diaz (2012). "A Computer Adaptive Testing (CAT) approach to Patient Reported Outcomes (PROs) for mobile devices." Journal of Mobile Technology in Medicine **1**(4S): 20-20.

[13] Beets, M., C. F. Morgan, et al. (2011). "Convergent validity of pedometer and accelerometer estimates of moderate-to-vigorous physical activity of youth." Journal of physical activity & health **8**: S295.

[14] Behrens, T. K. and M. K. Dinger (2011). "Comparisons of accelerometer and pedometer determined steps in free living samples." Journal of physical activity & health **8**(3): 390.

[15] Boyce, G., G. Padmasekara, et al. (2012). "Accuracy of Mobile Phone Pedometer Technology." Journal of Mobile Technology in Medicine **1**(2): 23-27.

[16] Boyce, G., G. Padmasekara, et al. (2012). "Accuracy of Mobile Phone Pedometer Technology." Journal of Mobile Technology in Medicine **1**(2): 27-30.

[17] Brown, S. (2012). "The Top 40: Best Mobile Apps for Handheld Librarians." The Reference Librarian **53**(4): 456-465.

[18] Butler, M. (2011). "Android: Changing the mobile landscape." Pervasive Computing, IEEE **10**(1): 4-7.

[19] Carlson, D. and A. Schrader (2011). A wide-area context-awareness approach for Android. Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services, ACM.

[20] Carlson, D. and A. Schrader (2012). Dynamix: An open plug-and-play context framework for android. Internet of Things (IOT), 2012 3rd International Conference on the, IEEE.

[21] Champion, A. C. and D. Xuan (2012). "Mobile Handsets: A Panoramic Overview."

[22] Cho, D.-K., M. Mun, et al. (2010). AutoGait: A mobile platform that accurately estimates the distance walked. Pervasive computing and communications (PerCom), 2010 IEEE international conference on, IEEE.

[23] Chueng, P. P. Y., S. Chen, et al. (2012). "Using Mobile Phone Messages in Pedometer-Based Intervention for Working Adults in Hong Kong." Asian Journal of Exercise and Sports Science **9**(2).

[24] Colley, R. C., D. Garriguet, et al. (2011). "Physical activity of Canadian children and youth: accelerometer results from the 2007 to 2009 Canadian Health Measures Survey." Health Rep **22**(1): 15-23.

[25] David, P., J. Buckworth, et al. (2012). "A walking intervention for postmenopausal women using mobile phones and interactive voice response." Journal of Telemedicine and Telecare **18**(1): 20-25.

[26] De Greef, K. P., B. I. Deforche, et al. (2011). "The effects of a pedometer-based behavioral modification program with telephone support on physical activity and sedentary behavior in type 2 diabetes patients." Patient education and counseling **84**(2): 275-279.

[27] Duncan, S., K. White, et al. (2011). "Convergent validity of a piezoelectric pedometer and an omnidirectional accelerometer for measuring children's physical activity." Pediatric exercise science **23**(3): 399.

[28] Felt, A. P., M. Finifter, et al. (2011). A survey of mobile malware in the wild. Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM.

[29] Foster, D., C. Linehan, et al. (2010). "Motivating physical activity at work: using persuasive social media extensions for simple mobile devices."

[30] Fukuoka, Y., E. Kamitani, et al. (2011). "New insights into compliance with a mobile phone diary and pedometer use in sedentary women." Journal of physical activity & health **8**(3): 398.

[31] Fukuoka, Y., E. Kamitani, et al. (2011). "New insights into compliance with a mobile phone diary and pedometer use in sedentary women." Journal of physical activity & health **8**(3): 398.

[32] Fukuoka, Y., J. Komatsu, et al. (2011). "The mPED randomized controlled clinical trial: applying mobile persuasive technologies to increase physical activity in sedentary women protocol." BMC public health **11**(1): 933.

[33] Fukuoka, Y., T. Lindgren, et al. (2012). "Qualitative Exploration of the Acceptability of a Mobile Phone and Pedometer-Based Physical Activity Program in a Diverse Sample of Sedentary Women." Public Health Nursing **29**(3): 232-240.

[34] Fukuoka, Y., E. Vittinghoff, et al. (2010). "Innovation to motivation—pilot study of a mobile phone intervention to increase physical activity among sedentary women." Preventive medicine **51**(3): 287-289.

[35] Fuzi, M. F. B. M. (2013). "Pedometer."

[36] Gandhewar, N. and R. Sheikh (2010). "Google Android: An emerging software platform for mobile devices." International Journal on Computer Science and Engineering **1**(1): 12-17.

[37] Garcia, E., H. Ding, et al. (2010). Can a mobile phone be used as a pedometer in an outpatient cardiac rehabilitation program? Complex Medical Engineering (CME), 2010 IEEE/ICME International Conference on, IEEE.

[38] Glynn, L. G., P. S. Hayes, et al. (2013). "SMART MOVE-a smartphone-based intervention to promote physical activity in primary care: study protocol for a randomized controlled trial." Trials **14**(1): 157.

[39] Goadrich, M. H. and M. P. Rogers (2011). Smart smartphone development: iOS versus Android. Proceedings of the 42nd ACM technical symposium on Computer science education, ACM.

[40] Guo, Q. (2012). Android Health-Care App: Multi-function Step Counter, Mid Sweden University.

[41] Harrington, D., C. Tudor-Locke, et al. (2011). "Translation of moderate-to-vigorous physical activity recommendations into pedometer-based stepping targets in the Lower Mississippi Delta." Medicine and Science in Sports and Exercise **43**(5 Supplement): S235.

[42] Harris, T., S. Kerry, et al. (2013). "Randomised controlled trial of a complex intervention by primary care nurses to increase walking in patients aged 60–74 years: protocol of the PACE-Lift (Pedometer Accelerometer Consultation Evaluation-Lift) trial." BMC public health **13**(1): 5.

[43] Harris, T. J., C. G. Owen, et al. (2009). "A comparison of questionnaire, accelerometer, and pedometer: measures in older people." Med Sci Sports Exerc **41**(7): 1392-1402.

[44] Hongman, W., Z. Xiaocheng, et al. (2011). Acceleration and Orientation Multisensor Pedometer Application Design and Implementation on the Android Platform. Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on, IEEE.

[45] Hori, Y., Y. Tokuda, et al. (2013). Communication pedometer: a discussion of gamified communication focused on frequency of smiles. Proceedings of the 4th Augmented Human International Conference, ACM.

[46] Huang, Y., H. Zheng, et al. (2010). Activity monitoring using an intelligent mobile phone: a validation study. Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments, ACM.

[47] Lavian, T. and Z. Or-Bach (2011). PORTABLE UNIVERSAL COMMUNICATION DEVICE, Google Patents.

[48] Liu, P., K.-W. Yuen, et al. (2012). mENUNCIATE: Development of a computer-aided pronunciation training system on a cross-platform framework for mobile, speech-enabled application development. Chinese Spoken Language Processing (ISCSLP), 2012 8th International Symposium on, IEEE.

[49] M. Brandl. (2001) "High performance accelerometer based on CMOS technologies with low cost add-ons " p. 3.

[50] Park, T. H. and H.-C. Kim "From Mechanical Pedometer to Digital Pedometer: A Usability Study on a Walking Promotion System."

[51] Shyi-Shiou, W. and W. Hsin-Yi (2011). The Design of an Intelligent Pedometer Using Android. Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference on, IEEE.

[52] Simon, V. The Best iPhone, Android, and BlackBerry Apps.

[53] Tilson, D., C. Sorensen, et al. (2012). Change and control paradoxes in mobile infrastructure innovation: the Android and iOS mobile operating systems cases. System Science (HICSS), 2012 45th Hawaii International Conference on, IEEE.

Anexos.

Anexo 1: Código fuente de la aplicación.

Notificador de distancia:

```
package wk.tesis.podometro;

public class DistanceNotifier implements StepListener, SpeakingTimer.Listener {

 public interface Listener {
 public void valueChanged(float value);
 public void passValue();
 }
 private Listener mListener;

 float mDistance = 0;

 PedometerSettings mSettings;
 Utils mUtils;

 boolean mIsMetric;
 float mStepLength;

 public DistanceNotifier(Listener listener, PedometerSettings settings, Utils
utils) {
 mListener = listener;
 mUtils = utils;
 mSettings = settings;
 reloadSettings();
 }
 public void setDistance(float distance) {
 mDistance = distance;
 notifyListener();
 }

 public void reloadSettings() {
 mIsMetric = mSettings.isMetric();
 mStepLength = mSettings.getStepLength();
 notifyListener();
 }

 public void onStep() {

 if (mIsMetric) {
 mDistance += (float)(// kilometers
 mStepLength // centimeters
 / 100000.0); // centimeters/kilometer
 }
 else {
 mDistance += (float)(// miles
 mStepLength // inches
 / 63360.0); // inches/mile
```

```
 }

 notifyListener();
 }

 private void notifyListener() {
 mListener.valueChanged(mDistance);
 }

 public void passValue() {
 // Callback of StepListener - Not implemented
 }

 public void speak() {
 if (mSettings.shouldTellDistance()) {
 if (mDistance >= .001f) {
 mUtils.say(("" + (mDistance + 0.000001f)).substring(0, 4) + (mIsMetric ? "
kilometers" : " miles"));
 // TODO: format numbers (no "." at the end)
 }
 }
 }
 }
```

Notificador de paso:

```
package wk.tesis.podometro;

import java.util.ArrayList;

public class PaceNotifier implements StepListener, SpeakingTimer.Listener {

 public interface Listener {
 public void paceChanged(int value);
 public void passValue();
 }
 private ArrayList<Listener> mListeners = new ArrayList<Listener>();

 int mCounter = 0;

 private long mLastStepTime = 0;
 private long[] mLastStepDeltas = {-1, -1, -1, -1};
 private int mLastStepDeltasIndex = 0;
 private long mPace = 0;

 PedometerSettings mSettings;
 Utils mUtils;

 int mDesiredPace;

 boolean mShouldTellFasterslower;

 private long mSpokenAt = 0;

 public PaceNotifier(PedometerSettings settings, Utils utils) {
 mUtils = utils;
 mSettings = settings;
```

```
mDesiredPace = mSettings.getDesiredPace();
reloadSettings();
}
public void setPace(int pace) {
mPace = pace;
int avg = (int)(60*1000.0 / mPace);
for (int i = 0; i < mLastStepDeltas.length; i++) {
mLastStepDeltas[i] = avg;
}
notifyListener();
}
public void reloadSettings() {
mShouldTellFasterslower =
mSettings.shouldTellFasterslower()
&& mSettings.getMaintainOption() == PedometerSettings.M_PACE;
notifyListener();
}

public void addListener(Listener l) {
mListeners.add(l);
}

public void setDesiredPace(int desiredPace) {
mDesiredPace = desiredPace;
}

public void onStep() {
long thisStepTime = System.currentTimeMillis();
mCounter ++;

 if (mLastStepTime > 0) {
long delta = thisStepTime - mLastStepTime;

mLastStepDeltas[mLastStepDeltasIndex] = delta;
mLastStepDeltasIndex = (mLastStepDeltasIndex + 1) % mLastStepDeltas.length;

long sum = 0;
boolean isMeaningfull = true;
for (int i = 0; i < mLastStepDeltas.length; i++) {
if (mLastStepDeltas[i] < 0) {
isMeaningfull = false;
break;
}
sum += mLastStepDeltas[i];
}
if (isMeaningfull && sum > 0) {
long avg = sum / mLastStepDeltas.length;
mPace = 60*1000 / avg;

// TODO: remove duplication. This also exists in SpeedNotifier
if (mShouldTellFasterslower && !mUtils.isSpeakingEnabled()) {
if (thisStepTime - mSpokenAt > 3000 && !mUtils.isSpeakingNow()) {
float little = 0.10f;
float normal = 0.30f;
float much = 0.50f;
```

```
boolean spoken = true;
if (mPace < mDesiredPace * (1 - much)) {
mUtils.say("much faster!");
}
else
if (mPace > mDesiredPace * (1 + much)) {
mUtils.say("much slower!");
}
else
if (mPace < mDesiredPace * (1 - normal)) {
mUtils.say("faster!");
}
else
if (mPace > mDesiredPace * (1 + normal)) {
mUtils.say("slower!");
}
else
if (mPace < mDesiredPace * (1 - little)) {
mUtils.say("a little faster!");
}
else
if (mPace > mDesiredPace * (1 + little)) {
mUtils.say("a little slower!");
}
else {
spoken = false;
}
if (spoken) {
mSpokenAt = thisStepTime;
}
}
}
}
else {
mPace = -1;
}
}
mLastStepTime = thisStepTime;
notifyListener();
}

private void notifyListener() {
for (Listener listener : mListeners) {
listener.paceChanged((int)mPace);
}
}

public void passValue() {
// Not used
}

public void speak() {
if (mSettings.shouldTellPace()) {
if (mPace > 0) {
```

```
 mUtils.say(mPace + " steps per minute");
 }
 }
 }

}
```

Podómetro:

```
package wk.tesis.podometro;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class Pedometer extends Activity {
 private static final String TAG = "Pedometer";
 private SharedPreferences mSettings;
 private PedometerSettings mPedometerSettings;
 private Utils mUtils;

 private TextView mStepValueView;
 private TextView mPaceValueView;
 private TextView mDistanceValueView;
 private TextView mSpeedValueView;;
 TextView mDesiredPaceView;
 private int mStepValue;
 private int mPaceValue;
 private float mDistanceValue;
 private float mSpeedValue;
 private float mDesiredPaceOrSpeed;
 private int mMaintain;
 private float mMaintainInc;
 private boolean mQuitting = false; // Set when user selected Quit from menu,
can be used by onPause, onStop, onDestroy

 private boolean mIsRunning;

 @Override
 public void onCreate(Bundle savedInstanceState) {
 Log.i(TAG, "[ACTIVITY] onCreate");
```

```
 super.onCreate(savedInstanceState);

 mStepValue = 0;
 mPaceValue = 0;

 setContentView(R.layout.main);

 mUtils = Utils.getInstance();
 }

 @Override
 protected void onStart() {
 Log.i(TAG, "[ACTIVITY] onStart");
 super.onStart();
 }

 @Override
 protected void onResume() {
 Log.i(TAG, "[ACTIVITY] onResume");
 super.onResume();

 mSettings = PreferenceManager.getDefaultSharedPreferences(this);
 mPedometerSettings = new PedometerSettings(mSettings);

 mUtils.setSpeak(mSettings.getBoolean("speak", false));

 // Read from preferences if the service was running on the last onPause
 mIsRunning = mPedometerSettings.isServiceRunning();

 // Start the service if this is considered to be an application start (last
onPause was long ago)
 if (!mIsRunning && mPedometerSettings.isNewStart()) {
 startStepService();
 bindStepService();
 }
 else if (mIsRunning) {
 bindStepService();
 }

 mPedometerSettings.clearServiceRunning();

 mStepValueView = (TextView) findViewById(R.id.step_value);
 mPaceValueView = (TextView) findViewById(R.id.pace_value);
 mDistanceValueView = (TextView) findViewById(R.id.distance_value);
 mSpeedValueView = (TextView) findViewById(R.id.speed_value);
 mDesiredPaceView = (TextView) findViewById(R.id.desired_pace_value);


 mMaintain = mPedometerSettings.getMaintainOption();
 ((LinearLayout) this.findViewById(R.id.desired_pace_control)).setVisibility(
 mMaintain != PedometerSettings.M_NONE
 ? View.VISIBLE
 : View.GONE
 );
 if (mMaintain == PedometerSettings.M_PACE) {
```

```
mMaintainInc = 5f;
mDesiredPaceOrSpeed = (float)mPedometerSettings.getDesiredPace();
}
else
if (mMaintain == PedometerSettings.M_SPEED) {
mDesiredPaceOrSpeed = mPedometerSettings.getDesiredSpeed();
mMaintainInc = 0.1f;
}
Button button1 = (Button) findViewById(R.id.button_desired_pace_lower);
button1.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
mDesiredPaceOrSpeed -= mMaintainInc;
mDesiredPaceOrSpeed = Math.round(mDesiredPaceOrSpeed * 10) / 10f;
displayDesiredPaceOrSpeed();
setDesiredPaceOrSpeed(mDesiredPaceOrSpeed);
}
});
Button button2 = (Button) findViewById(R.id.button_desired_pace_raise);
button2.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
mDesiredPaceOrSpeed += mMaintainInc;
mDesiredPaceOrSpeed = Math.round(mDesiredPaceOrSpeed * 10) / 10f;
displayDesiredPaceOrSpeed();
setDesiredPaceOrSpeed(mDesiredPaceOrSpeed);
}
});
if (mMaintain != PedometerSettings.M_NONE) {
((TextView) findViewById(R.id.desired_pace_label)).setText(
mMaintain == PedometerSettings.M_PACE
? R.string.desired_pace
: R.string.desired_speed
);
}


displayDesiredPaceOrSpeed();
}

private void displayDesiredPaceOrSpeed() {
if (mMaintain == PedometerSettings.M_PACE) {
mDesiredPaceView.setText("" + (int)mDesiredPaceOrSpeed);
}
else {
mDesiredPaceView.setText("" + mDesiredPaceOrSpeed);
}
}

@Override
protected void onPause() {
Log.i(TAG, "[ACTIVITY] onPause");
if (mIsRunning) {
unbindStepService();
}
if (mQuitting) {
mPedometerSettings.saveServiceRunningWithNullTimestamp(mIsRunning);
```

```
}
else {
mPedometerSettings.saveServiceRunningWithTimestamp(mIsRunning);
}

super.onPause();
savePaceSetting();
}

@Override
protected void onStop() {
Log.i(TAG, "[ACTIVITY] onStop");
super.onStop();
}

protected void onDestroy() {
Log.i(TAG, "[ACTIVITY] onDestroy");
super.onDestroy();
}

protected void onRestart() {
Log.i(TAG, "[ACTIVITY] onRestart");
super.onDestroy();
}

private void setDesiredPaceOrSpeed(float desiredPaceOrSpeed) {
if (mService != null) {
if (mMaintain == PedometerSettings.M_PACE) {
mService.setDesiredPace((int)desiredPaceOrSpeed);
}
else
if (mMaintain == PedometerSettings.M_SPEED) {
mService.setDesiredSpeed(desiredPaceOrSpeed);
}
}
}

private void savePaceSetting() {
mPedometerSettings.savePaceOrSpeedSetting(mMaintain, mDesiredPaceOrSpeed);
}

private StepService mService;

private ServiceConnection mConnection = new ServiceConnection() {
public void onServiceConnected(ComponentName className, IBinder service) {
mService = ((StepService.StepBinder)service).getService();

mService.registerCallback(mCallback);
mService.reloadSettings();

}

public void onServiceDisconnected(ComponentName className) {
mService = null;
}
```

```
 };

 private void startStepService() {
 if (! mIsRunning) {
 Log.i(TAG, "[SERVICE] Start");
 mIsRunning = true;
 startService(new Intent(Pedometer.this,
 StepService.class));
 }
 }

 private void bindStepService() {
 Log.i(TAG, "[SERVICE] Bind");
 bindService(new Intent(Pedometer.this,
 StepService.class), mConnection, Context.BIND_AUTO_CREATE +
Context.BIND_DEBUG_UNBIND);
 }

 private void unbindStepService() {
 Log.i(TAG, "[SERVICE] Unbind");
 unbindService(mConnection);
 }

 private void stopStepService() {
 Log.i(TAG, "[SERVICE] Stop");
 if (mService != null) {
 Log.i(TAG, "[SERVICE] stopService");
 stopService(new Intent(Pedometer.this,
 StepService.class));
 }
 mIsRunning = false;
 }

 private void resetValues(boolean updateDisplay) {
 if (mService != null && mIsRunning) {
 mService.resetValues();
 }
 else {
 mStepValueView.setText("0");
 mPaceValueView.setText("0");
 mDistanceValueView.setText("0");
 mSpeedValueView.setText("0");
 SharedPreferences state = getSharedPreferences("state", 0);
 SharedPreferences.Editor stateEditor = state.edit();
 if (updateDisplay) {
 stateEditor.putInt("steps", 0);
 stateEditor.putInt("pace", 0);
 stateEditor.putFloat("distance", 0);
 stateEditor.putFloat("speed", 0);
 stateEditor.commit();
 }
 }
 }

 private static final int MENU_SETTINGS = 8;
```

```
private static final int MENU_QUIT = 9;

private static final int MENU_PAUSE = 1;
private static final int MENU_RESUME = 2;
private static final int MENU_RESET = 3;

public boolean onPrepareOptionsMenu(Menu menu) {
menu.clear();
if (mIsRunning) {
menu.add(0, MENU_PAUSE, 0, R.string.pause)
.setIcon(android.R.drawable.ic_media_pause)
.setShortcut('1', 'p');
}
else {
menu.add(0, MENU_RESUME, 0, R.string.resume)
.setIcon(android.R.drawable.ic_media_play)
.setShortcut('1', 'p');
}
menu.add(0, MENU_RESET, 0, R.string.reset)
.setIcon(android.R.drawable.ic_menu_close_clear_cancel)
.setShortcut('2', 'r');
menu.add(0, MENU_SETTINGS, 0, R.string.settings)
.setIcon(android.R.drawable.ic_menu_preferences)
.setShortcut('8', 's')
.setIntent(new Intent(this, Settings.class));
menu.add(0, MENU_QUIT, 0, R.string.quit)
.setIcon(android.R.drawable.ic_lock_power_off)
.setShortcut('9', 'q');
return true;
}


public boolean onOptionsItemSelected(MenuItem item) {
switch (item.getItemId()) {
case MENU_PAUSE:
unbindStepService();
stopStepService();
return true;
case MENU_RESUME:
startStepService();
bindStepService();
return true;
case MENU_RESET:
resetValues(true);
return true;
case MENU_QUIT:
resetValues(false);
unbindStepService();
stopStepService();
mQuitting = true;
finish();
return true;
}
return false;
}
```

```
 // TODO: unite all into 1 type of message
 private StepService.ICallback mCallback = new StepService.ICallback() {
 public void stepsChanged(int value) {
 mHandler.sendMessage(mHandler.obtainMessage(STEPS_MSG, value, 0));
 }
 public void paceChanged(int value) {
 mHandler.sendMessage(mHandler.obtainMessage(PACE_MSG, value, 0));
 }
 public void distanceChanged(float value) {
 mHandler.sendMessage(mHandler.obtainMessage(DISTANCE_MSG, (int)(value*1000),
0));
 }
 public void speedChanged(float value) {
 mHandler.sendMessage(mHandler.obtainMessage(SPEED_MSG, (int)(value*1000), 0));
 }
 };

 private static final int STEPS_MSG = 1;
 private static final int PACE_MSG = 2;
 private static final int DISTANCE_MSG = 3;
 private static final int SPEED_MSG = 4;

 private Handler mHandler = new Handler() {
 @Override public void handleMessage(Message msg) {
 switch (msg.what) {
 case STEPS_MSG:
 mStepValue = (int)msg.arg1;
 mStepValueView.setText("" + mStepValue);
 break;
 case PACE_MSG:
 mPaceValue = msg.arg1;
 if (mPaceValue <= 0) {
 mPaceValueView.setText("0");
 }
 else {
 mPaceValueView.setText("" + (int)mPaceValue);
 }
 break;
 case DISTANCE_MSG:
 mDistanceValue = ((int)msg.arg1)/1000f;
 if (mDistanceValue <= 0) {
 mDistanceValueView.setText("0");
 }
 else {
 mDistanceValueView.setText(
 ("" + (mDistanceValue + 0.000001f)).substring(0, 5)
 );
 }
 break;
 case SPEED_MSG:
 mSpeedValue = ((int)msg.arg1)/1000f;
 if (mSpeedValue <= 0) {
 mSpeedValueView.setText("0");
 }
```

```
 else {
 mSpeedValueView.setText(
 ("" + (mSpeedValue + 0.000001f)).substring(0, 4)
 );
 }
 break;
 }
 }

 };


}
```

Configuración del podómetro:

```
package wk.tesis.podometro;

import android.content.SharedPreferences;

public class PedometerSettings {

 SharedPreferences mSettings;

 public static int M_NONE = 1;
 public static int M_PACE = 2;
 public static int M_SPEED = 3;

 public PedometerSettings(SharedPreferences settings) {
 mSettings = settings;
 }

 public boolean isMetric() {
 return mSettings.getString("units", "metric").equals("metric");
 }

 public float getStepLength() {
 try {
 return Float.valueOf(mSettings.getString("step_length", "20").trim());
 }
 catch (NumberFormatException e) {
 // TODO: reset value, & notify user somehow
 return 0f;
 }
 }

 public float getBodyWeight() {
 try {
 return Float.valueOf(mSettings.getString("body_weight", "50").trim());
 }
 catch (NumberFormatException e) {
 // TODO: reset value, & notify user somehow
 return 0f;
 }
```

```
 }

 public boolean isRunning() {
 return mSettings.getString("exercise_type", "running").equals("running");
 }

 public int getMaintainOption() {
 String p = mSettings.getString("maintain", "none");
 return
 p.equals("none") ? M_NONE : (
 p.equals("pace") ? M_PACE : (
 p.equals("speed") ? M_SPEED : (
 0)));
 }

 public int getDesiredPace() {
 return mSettings.getInt("desired_pace", 180); // steps/minute
 }
 public float getDesiredSpeed() {
 return mSettings.getFloat("desired_speed", 4f); // km/h or mph
 }
 public void savePaceOrSpeedSetting(int maintain, float desiredPaceOrSpeed) {
 SharedPreferences.Editor editor = mSettings.edit();
 if (maintain == M_PACE) {
 editor.putInt("desired_pace", (int)desiredPaceOrSpeed);
 }
 else
 if (maintain == M_SPEED) {
 editor.putFloat("desired_speed", desiredPaceOrSpeed);
 }
 editor.commit();
 }
}
```

Configuración:

```
package wk.tesis.podometro;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class Settings extends PreferenceActivity {
 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);

 addPreferencesFromResource(R.xml.preferences);
 }
}
```

Temporizador:

```
package wk.tesis.podometro;
```

```
import java.util.ArrayList;

public class SpeakingTimer implements StepListener {

 PedometerSettings mSettings;
 Utils mUtils;
 boolean mShouldSpeak;
 float mInterval;
 long mLastSpeakTime;

 public SpeakingTimer(PedometerSettings settings, Utils utils) {
 mLastSpeakTime = System.currentTimeMillis();
 mSettings = settings;
 mUtils = utils;
 reloadSettings();
 }
 public void reloadSettings() {
 mShouldSpeak = mSettings.shouldSpeak();
 mInterval = mSettings.getSpeakingInterval();
 }

 public void onStep() {
 long now = System.currentTimeMillis();
 long delta = now - mLastSpeakTime;

 if (delta / 60000.0 >= mInterval) {
 mLastSpeakTime = now;
 notifyListeners();
 }
 }

 public void passValue() {
 // not used
 }

 public interface Listener {
 public void speak();
 }
 private ArrayList<Listener> mListeners = new ArrayList<Listener>();

 public void addListener(Listener l) {
 mListeners.add(l);
 }
 public void notifyListeners() {
 mUtils.ding();
 for (Listener listener : mListeners) {
 listener.speak();
 }
 }
}
```

Notificador de velocidad:

```
package wk.tesis.podometro;

public class SpeedNotifier implements PaceNotifier.Listener,
SpeakingTimer.Listener {
```

```
 public interface Listener {
 public void valueChanged(float value);
 public void passValue();
 }
 private Listener mListener;

 int mCounter = 0;
 float mSpeed = 0;

 boolean mIsMetric;
 float mStepLength;

 PedometerSettings mSettings;
 Utils mUtils;

 float mDesiredSpeed;

 boolean mShouldTellFasterslower;
 boolean mShouldTellSpeed;

 private long mSpokenAt = 0;

 public SpeedNotifier(Listener listener, PedometerSettings settings, Utils
utils) {
 mListener = listener;
 mUtils = utils;
 mSettings = settings;
 mDesiredSpeed = mSettings.getDesiredSpeed();
 reloadSettings();
 }
 public void setSpeed(float speed) {
 mSpeed = speed;
 notifyListener();
 }
 public void reloadSettings() {
 mIsMetric = mSettings.isMetric();
 mStepLength = mSettings.getStepLength();
 mShouldTellSpeed = mSettings.shouldTellSpeed();
 mShouldTellFasterslower =
 mSettings.shouldTellFasterslower()
 && mSettings.getMaintainOption() == PedometerSettings.M_SPEED;
 notifyListener();
 }
 public void setDesiredSpeed(float desiredSpeed) {
 mDesiredSpeed = desiredSpeed;
 }

 private void notifyListener() {
 mListener.valueChanged(mSpeed);
 }

 public void paceChanged(int value) {
 if (mIsMetric) {
 mSpeed = // kilometers / hour
```

```
 value * mStepLength // centimeters / minute
 / 100000f * 60f; // centimeters/kilometer
 }
 else {
 mSpeed = // miles / hour
 value * mStepLength // inches / minute
 / 63360f * 60f; // inches/mile
 }
 tellFasterSlower();
 notifyListener();
 }

 public void passValue() {
 // Not used
 }

 public void speak() {
 if (mSettings.shouldTellSpeed()) {
 if (mSpeed >= .01f) {
 mUtils.say("" + (mSpeed + 0.000001f)).substring(0, 4) + (mIsMetric ? "
kilometers per hour" : " miles per hour"));
 }
 }
 }
}
```

Vibración para el paso:

```
package wk.tesis.podometro;

import android.content.Context;
import android.os.Vibrator;

public class StepBuzzer implements StepListener {

 private Context mContext;
 private Vibrator mVibrator;

 public StepBuzzer(Context context) {
 mContext = context;
 mVibrator = (Vibrator)mContext.getSystemService(Context.VIBRATOR_SERVICE);
 }

 public void onStep() {
 buzz();
 }

 public void passValue() {

 }

 private void buzz() {
 mVibrator.vibrate(50);
 }
}
```

Detector de paso:

```
package wk.tesis.podometro;

import java.util.ArrayList;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;

public class StepDetector implements SensorEventListener
{
 private final static String TAG = "StepDetector";
 private float mLimit = 10;
 private float mLastValues[] = new float[3*2];
 private float mScale[] = new float[2];
 private float mYOffset;

 private float mLastDirections[] = new float[3*2];
 private float mLastExtremes[][] = { new float[3*2], new float[3*2] };
 private float mLastDiff[] = new float[3*2];
 private int mLastMatch = -1;

 private ArrayList<StepListener> mStepListeners = new ArrayList<StepListener>();

 public StepDetector() {
 int h = 480; // TODO: remove this constant
 mYOffset = h * 0.5f;
 mScale[0] = - (h * 0.5f * (1.0f / (SensorManager.STANDARD_GRAVITY * 2)));
 mScale[1] = - (h * 0.5f * (1.0f / (SensorManager.MAGNETIC_FIELD_EARTH_MAX)));
 }

 public void setSensitivity(float sensitivity) {
 mLimit = sensitivity; // 1.97 2.96 4.44 6.66 10.00 15.00 22.50 33.75 50.62
 }

 public void addStepListener(StepListener sl) {
 mStepListeners.add(sl);
 }

 //public void onSensorChanged(int sensor, float[] values) {
 public void onSensorChanged(SensorEvent event) {
 Sensor sensor = event.sensor;
 synchronized (this) {
 if (sensor.getType() == Sensor.TYPE_ORIENTATION) {
 }
 else {
 int j = (sensor.getType() == Sensor.TYPE_ACCELEROMETER) ? 1 : 0;
 if (j == 1) {
 float vSum = 0;
 for (int i=0 ; i<3 ; i++) {
 final float v = mYOffset + event.values[i] * mScale[j];
 vSum += v;
```

```
 }
 int k = 0;
 float v = vSum / 3;

 float direction = (v > mLastValues[k] ? 1 : (v < mLastValues[k] ? -1 : 0));
 if (direction == - mLastDirections[k]) {
 // Direction changed
 int extType = (direction > 0 ? 0 : 1); // minumum or maximum?
 mLastExtremes[extType][k] = mLastValues[k];
 float diff = Math.abs(mLastExtremes[extType][k] - mLastExtremes[1 -
extType][k]);

 if (diff > mLimit) {

 boolean isAlmostAsLargeAsPrevious = diff > (mLastDiff[k]*2/3);
 boolean isPreviousLargeEnough = mLastDiff[k] > (diff/3);
 boolean isNotContra = (mLastMatch != 1 - extType);

 if (isAlmostAsLargeAsPrevious && isPreviousLargeEnough && isNotContra) {
 Log.i(TAG, "step");
 for (StepListener stepListener : mStepListeners) {
 stepListener.onStep();
 }
 mLastMatch = extType;
 }
 else {
 mLastMatch = -1;
 }
 }
 mLastDiff[k] = diff;
 }
 mLastDirections[k] = direction;
 mLastValues[k] = v;
 }
 }
 }
 }

 public void onAccuracyChanged(Sensor sensor, int accuracy) {
 // TODO Auto-generated method stub
 }

}
```

Visualizador del paso:

```
package wk.tesis.podometro;

import java.util.ArrayList;

public class StepDisplayer implements StepListener, SpeakingTimer.Listener {

 private int mCount = 0;
 PedometerSettings mSettings;
```

```
 Utils mUtils;

 public StepDisplayer(PedometerSettings settings, Utils utils) {
 mUtils = utils;
 mSettings = settings;
 notifyListener();
 }
 public void setUtils(Utils utils) {
 mUtils = utils;
 }

 public void setSteps(int steps) {
 mCount = steps;
 notifyListener();
 }
 public void onStep() {
 mCount ++;
 notifyListener();
 }
 public void reloadSettings() {
 notifyListener();
 }
 public void passValue() {
 }

 public interface Listener {
 public void stepsChanged(int value);
 public void passValue();
 }
 private ArrayList<Listener> mListeners = new ArrayList<Listener>();

 public void addListener(Listener l) {
 mListeners.add(l);
 }
 public void notifyListener() {
 for (Listener listener : mListeners) {
 listener.stepsChanged((int)mCount);
 }
 }
}
```

Manejar notificación de paso:

```
package wk.tesis.podometro;

public interface StepListener {
 public void onStep();
 public void passValue();
}
```

Servicio del detector de paso:

```
package wk.tesis.podometro;

import android.app.Notification;
import android.app.NotificationManager;
```

```
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.os.Binder;
import android.os.IBinder;
import android.os.PowerManager;
import android.preference.PreferenceManager;
import android.util.Log;
import android.widget.Toast;


public class StepService extends Service {
 private static final String TAG = "wk.tesis.podometro.StepService";
 private SharedPreferences mSettings;
 private PedometerSettings mPedometerSettings;
 private SharedPreferences mState;
 private SharedPreferences.Editor mStateEditor;
 private Utils mUtils;
 private SensorManager mSensorManager;
 private Sensor mSensor;
 private StepDetector mStepDetector;
 // private StepBuzzer mStepBuzzer; // used for debugging
 private StepDisplayer mStepDisplayer;
 private PaceNotifier mPaceNotifier;
 private DistanceNotifier mDistanceNotifier;
 private SpeedNotifier mSpeedNotifier;
 private SpeakingTimer mSpeakingTimer;
 private PowerManager.WakeLock wakeLock;
 private NotificationManager mNM;

 private int mSteps;
 private int mPace;
 private float mDistance;
 private float mSpeed;

 public class StepBinder extends Binder {
 StepService getService() {
 return StepService.this;
 }
 }

 @Override
 public void onCreate() {
 Log.i(TAG, "[SERVICE] onCreate");
 super.onCreate();

 mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
 showNotification();
```

```
 // Load settings
 mSettings = PreferenceManager.getDefaultSharedPreferences(this);
 mPedometerSettings = new PedometerSettings(mSettings);
 mState = getSharedPreferences("state", 0);

 mUtils = Utils.getInstance();
 mUtils.setService(this);
 mUtils.initTTS();

 acquireWakeLock();

 // Start detecting
 mStepDetector = new StepDetector();
 mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
 registerDetector();

 IntentFilter filter = new IntentFilter(Intent.ACTION_SCREEN_OFF);
 registerReceiver(mReceiver, filter);

 mStepDisplayer = new StepDisplayer(mPedometerSettings, mUtils);
 mStepDisplayer.setSteps(mSteps = mState.getInt("steps", 0));
 mStepDisplayer.addListener(mStepListener);
 mStepDetector.addStepListener(mStepDisplayer);

 mPaceNotifier = new PaceNotifier(mPedometerSettings, mUtils);
 mPaceNotifier.setPace(mPace = mState.getInt("pace", 0));
 mPaceNotifier.addListener(mPaceListener);
 mStepDetector.addStepListener(mPaceNotifier);

 mDistanceNotifier = new DistanceNotifier(mDistanceListener, mPedometerSettings,
mUtils);
 mDistanceNotifier.setDistance(mDistance = mState.getFloat("distance", 0));
 mStepDetector.addStepListener(mDistanceNotifier);

 mSpeedNotifier = new SpeedNotifier(mSpeedListener, mPedometerSettings, mUtils);
 mSpeedNotifier.setSpeed(mSpeed = mState.getFloat("speed", 0));
 mPaceNotifier.addListener(mSpeedNotifier);

 mSpeakingTimer = new SpeakingTimer(mPedometerSettings, mUtils);
 mSpeakingTimer.addListener(mStepDisplayer);
 mSpeakingTimer.addListener(mPaceNotifier);
 mSpeakingTimer.addListener(mDistanceNotifier);
 mSpeakingTimer.addListener(mSpeedNotifier);
 mStepDetector.addStepListener(mSpeakingTimer);

 // Used when debugging:
 // mStepBuzzer = new StepBuzzer(this);
 // mStepDetector.addStepListener(mStepBuzzer);

 // Tell the user we started.
 Toast.makeText(this, getText(R.string.started), Toast.LENGTH_SHORT).show();
 }

 @Override
```

```
public void onStart(Intent intent, int startId) {
Log.i(TAG, "[SERVICE] onStart");
super.onStart(intent, startId);
}

@Override
public void onDestroy() {
Log.i(TAG, "[SERVICE] onDestroy");
mUtils.shutdownTTS();

// Unregister our receiver.
unregisterReceiver(mReceiver);
unregisterDetector();

mStateEditor = mState.edit();
mStateEditor.putInt("steps", mSteps);
mStateEditor.putInt("pace", mPace);
mStateEditor.putFloat("distance", mDistance);
mStateEditor.putFloat("speed", mSpeed);
mStateEditor.commit();

mNM.cancel(R.string.app_name);

wakeLock.release();

super.onDestroy();

// Stop detecting
mSensorManager.unregisterListener(mStepDetector);

// Tell the user we stopped.
Toast.makeText(this, getText(R.string.stopped), Toast.LENGTH_SHORT).show();
}

private void registerDetector() {
mSensor = mSensorManager.getDefaultSensor(
Sensor.TYPE_ACCELEROMETER /*|
Sensor.TYPE_MAGNETIC_FIELD |
Sensor.TYPE_ORIENTATION*/);
mSensorManager.registerListener(mStepDetector,
mSensor,
SensorManager.SENSOR_DELAY_FASTEST);
}

private void unregisterDetector() {
mSensorManager.unregisterListener(mStepDetector);
}

@Override
public IBinder onBind(Intent intent) {
Log.i(TAG, "[SERVICE] onBind");
return mBinder;
}

private final IBinder mBinder = new StepBinder();
```

```
public interface ICallback {
public void stepsChanged(int value);
public void paceChanged(int value);
public void distanceChanged(float value);
public void speedChanged(float value);
}

private ICallback mCallback;

public void registerCallback(ICallback cb) {
mCallback = cb;
//mStepDisplayer.passValue();
//mPaceListener.passValue();
}

private int mDesiredPace;
private float mDesiredSpeed;

public void setDesiredPace(int desiredPace) {
mDesiredPace = desiredPace;
if (mPaceNotifier != null) {
mPaceNotifier.setDesiredPace(mDesiredPace);
}
}
public void setDesiredSpeed(float desiredSpeed) {
mDesiredSpeed = desiredSpeed;
if (mSpeedNotifier != null) {
mSpeedNotifier.setDesiredSpeed(mDesiredSpeed);
}
}

public void reloadSettings() {
mSettings = PreferenceManager.getDefaultSharedPreferences(this);

if (mStepDetector != null) {
mStepDetector.setSensitivity(
Float.valueOf(mSettings.getString("sensitivity", "10"))
);
}

if (mStepDisplayer != null) mStepDisplayer.reloadSettings();
if (mPaceNotifier != null) mPaceNotifier.reloadSettings();
if (mDistanceNotifier != null) mDistanceNotifier.reloadSettings();
if (mSpeedNotifier != null) mSpeedNotifier.reloadSettings();
if (mSpeakingTimer != null) mSpeakingTimer.reloadSettings();
}

public void resetValues() {
mStepDisplayer.setSteps(0);
mPaceNotifier.setPace(0);
mDistanceNotifier.setDistance(0);
mSpeedNotifier.setSpeed(0);
}
```

```
 private StepDisplayer.Listener mStepListener = new StepDisplayer.Listener() {
 public void stepsChanged(int value) {
 mSteps = value;
 passValue();
 }
 public void passValue() {
 if (mCallback != null) {
 mCallback.stepsChanged(mSteps);
 }
 }
 };
 private PaceNotifier.Listener mPaceListener = new PaceNotifier.Listener() {
 public void paceChanged(int value) {
 mPace = value;
 passValue();
 }
 public void passValue() {
 if (mCallback != null) {
 mCallback.paceChanged(mPace);
 }
 }
 };
 private DistanceNotifier.Listener mDistanceListener = new
DistanceNotifier.Listener() {
 public void valueChanged(float value) {
 mDistance = value;
 passValue();
 }
 public void passValue() {
 if (mCallback != null) {
 mCallback.distanceChanged(mDistance);
 }
 }
 };
 private SpeedNotifier.Listener mSpeedListener = new SpeedNotifier.Listener() {
 public void valueChanged(float value) {
 mSpeed = value;
 passValue();
 }
 public void passValue() {
 if (mCallback != null) {
 mCallback.speedChanged(mSpeed);
 }
 }
 };


 private void showNotification() {
 CharSequence text = getText(R.string.app_name);
 Notification notification = new Notification(R.drawable.ic_notification, null,
 System.currentTimeMillis());
 notification.flags = Notification.FLAG_NO_CLEAR |
Notification.FLAG_ONGOING_EVENT;
 Intent pedometerIntent = new Intent();
 pedometerIntent.setComponent(new ComponentName(this, Pedometer.class));
```

```
 pedometerIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
 PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
 pedometerIntent, 0);
 notification.setLatestEventInfo(this, text,
 getText(R.string.notification_subtitle), contentIntent);

 mNM.notify(R.string.app_name, notification);
 }


 // BroadcastReceiver for handling ACTION_SCREEN_OFF.
 private BroadcastReceiver mReceiver = new BroadcastReceiver() {
 @Override
 public void onReceive(Context context, Intent intent) {
 // Check action just to be on the safe side.
 if (intent.getAction().equals(Intent.ACTION_SCREEN_OFF)) {
 // Unregisters the listener and registers it again.
 StepService.this.unregisterDetector();
 StepService.this.registerDetector();
 if (mPedometerSettings.wakeAggressively()) {
 wakeLock.release();
 acquireWakeLock();
 }
 }
 }
 };

 private void acquireWakeLock() {
 PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
 int wakeFlags;
 if (mPedometerSettings.wakeAggressively()) {
 wakeFlags = PowerManager.SCREEN_DIM_WAKE_LOCK |
PowerManager.ACQUIRE_CAUSES_WAKEUP;
 }
 else if (mPedometerSettings.keepScreenOn()) {
 wakeFlags = PowerManager.SCREEN_DIM_WAKE_LOCK;
 }
 else {
 wakeFlags = PowerManager.PARTIAL_WAKE_LOCK;
 }
 wakeLock = pm.newWakeLock(wakeFlags, TAG);
 wakeLock.acquire();
 }
}
```

Utilidades:

```
package wk.tesis.podometro;

import java.util.Locale;

import android.app.Service;
import android.speech.tts.TextToSpeech;
import android.text.format.Time;
```

```
import android.util.Log;

public class Utils implements TextToSpeech.OnInitListener {
 private static final String TAG = "Utils";
 private Service mService;

 private static Utils instance = null;

 private Utils() {
 }

 public static Utils getInstance() {
 if (instance == null) {
 instance = new Utils();
 }
 return instance;
 }

 public void setService(Service service) {
 mService = service;
 }

 public static long currentTimeInMillis() {
 Time time = new Time();
 time.setToNow();
 return time.toMillis(false);
 }
}
```

Editar medición:

```
package wk.tesis.podometro.preferences;

import android.content.Context;
import android.os.Bundle;
import android.preference.EditTextPreference;
import android.preference.PreferenceManager;
import android.text.InputType;
import android.util.AttributeSet;
import android.view.View;
import android.widget.EditText;

abstract public class EditMeasurementPreference extends EditTextPreference {
 boolean mIsMetric;

 protected int mTitleResource;
 protected int mMetricUnitsResource;

 public EditMeasurementPreference(Context context) {
 super(context);
 initPreferenceDetails();
 }
 public EditMeasurementPreference(Context context, AttributeSet attr) {
 super(context, attr);
```

```
 initPreferenceDetails();
 }
 public EditMeasurementPreference(Context context, AttributeSet attr, int
defStyle) {
 super(context, attr, defStyle);
 initPreferenceDetails();
 }

 abstract protected void initPreferenceDetails();

 protected void showDialog(Bundle state) {
 mIsMetric =
PreferenceManager.getDefaultSharedPreferences(getContext()).getString("units",
"metric").equals("metric");
 setDialogTitle(
 getContext().getString(mTitleResource) +
 " (" +
 getContext().getString(
 mIsMetric
 ? mMetricUnitsResource
 : mMetricUnitsResource) +
 ")"
 );

 try {
 Float.valueOf(getText());
 }
 catch (Exception e) {
 setText("25");
 }

 super.showDialog(state);
 }
 protected void onAddEditTextToDialogView (View dialogView, EditText editText) {
 editText.setRawInputType(
 InputType.TYPE_CLASS_NUMBER | InputType.TYPE_NUMBER_FLAG_DECIMAL);
 super.onAddEditTextToDialogView(dialogView, editText);
 }
 public void onDialogClosed(boolean positiveResult) {
 if (positiveResult) {
 try {
 Float.valueOf(((CharSequence)(getEditText().getText())).toString());
 }
 catch (NumberFormatException e) {
 this.showDialog(null);
 return;
 }
 }
 super.onDialogClosed(positiveResult);
 }
}
```

Longitud del paso:

```
package wk.tesis.podometro.preferences;
import wk.tesis.podometro.R;
import android.content.Context;
import android.preference.EditTextPreference;
import android.util.AttributeSet;
public class StepLengthPreference extends EditMeasurementPreference {

 public StepLengthPreference(Context context) {
 super(context);
 }
 public StepLengthPreference(Context context, AttributeSet attr) {
 super(context, attr);
 }
 public StepLengthPreference(Context context, AttributeSet attr, int defStyle) {
 super(context, attr, defStyle);
 }
 protected void initPreferenceDetails() {
 mTitleResource = R.string.step_length_setting_title;
 mMetricUnitsResource = R.string.centimeters;
 }
}
```

Anexo 2: Tabla 1 “Mediciones patrones y reales”.

Sujeto	Pasos reales	Pasos contados	Error absoluto	Módulo de error	Error relativo (%)
1	111	110	1	1	0,90
2	294	294	0	0	0,00
3	192	194	-2	2	1,04
4	152	153	-1	1	0,66
5	500	498	2	2	0,40
6	357	358	-1	1	0,28
7	128	130	-2	2	1,56
8	116	117	-1	1	0,86
9	455	453	2	2	0,44
10	167	167	0	0	0,00
11	263	264	-1	1	0,38
12	139	141	-2	2	1,44
13	100	101	-1	1	1,00
14	227	227	0	0	0,00
15	179	180	-1	1	0,56
16	333	330	3	3	0,90
17	122	123	-1	1	0,82
18	135	137	-2	2	1,48
19	417	417	0	0	0,00
20	250	249	1	1	0,40
21	114	116	-2	2	1,75
22	102	102	0	0	0,00
23	192	193	-1	1	0,52
24	156	158	-2	2	1,28
25	143	142	1	1	0,70
26	106	107	-1	1	0,94
27	125	126	-1	1	0,80
28	385	386	-1	1	0,26
29	208	209	-1	1	0,48
30	313	314	-1	1	0,31
31	172	173	-1	1	0,58
32	238	240	-2	2	0,84
33	104	105	-1	1	0,96
34	132	134	-2	2	1,54
35	119	118	1	1	0,84
36	278	279	-1	1	0,35
37	217	217	0	0	0,00
38	147	148	-1	1	0,68
39	109	109	0	0	0,00
40	200	199	1	1	0,50
41	161	160	1	1	0,62
42	227	227	0	0	0,00

Continuación de la Tabla 1.

Sujeto	Pasos reales	Pasos contados	Error absoluto	Módulo de error	Error relativo (%)
43	185	184	1	1	0,54
44	172	171	1	1	0,58
45	385	385	0	0	0,00
46	313	313	0	0	0,00
47	116	118	-2	2	1,72
48	139	140	-1	1	0,72
49	104	106	-2	2	1,92
50	263	260	3	3	1,14
51	250	250	0	0	0,00
52	152	151	1	1	0,66
53	143	143	0	0	0,00
54	455	454	1	1	0,22
55	357	355	2	2	0,56
56	417	419	-2	2	0,48
57	294	294	0	0	0,00
58	106	106	0	0	0,00
59	122	120	2	2	1,64
60	102	103	-1	1	0,98

Anexo 3: Tabla 2 “Evaluación de la precisión y la exactitud (sujeto 10)”.

Sujeto 10 (200 pasos)	Medición	Exactitud (%)
1	201	99,50
2	201	99,50
3	201	99,50
4	202	99,00
5	200	100,00
6	201	99,50
7	202	99,00
8	201	99,50
9	202	99,00
10	201	99,50
11	199	99,50
12	200	100,00
13	200	100,00
14	201	99,50
15	201	99,50
16	201	99,50
17	201	99,50
18	202	99,00
19	199	99,50
20	200	100,00
21	200	100,00
22	201	99,50
23	201	99,50
24	201	99,50
25	201	99,50
26	202	99,00
27	201	99,50
28	200	100,00
29	202	99,00
30	201	99,50
31	201	99,50
32	201	99,50
33	201	99,50
34	201	99,50
35	202	99,00
36	201	99,50
37	201	99,50
38	200	100,00

Continuación de la Tabla 2.

Sujeto 10 (200 pasos)	Medición	Exactitud (%)
39	199	99,50
40	199	99,50
41	200	100,00
42	201	99,50
43	201	99,50
44	201	99,50
45	201	99,50
46	202	99,00
47	201	99,50
48	201	99,50
49	200	100,00
50	200	100,00
Precisión (%)		99,60

Anexo 4: Tabla 3 “Evaluación de la precisión y la exactitud (sujeto 40)”.

Sujeto 40 (200 pasos)	Medición	Exactitud (%)
1	202	99,00
2	201	99,50
3	201	99,50
4	201	99,50
5	200	100,00
6	200	100,00
7	201	99,50
8	201	99,50
9	201	99,50
10	202	99,00
11	199	99,50
12	199	99,50
13	202	99,00
14	201	99,50
15	201	99,50
16	201	99,50
17	200	100,00
18	200	100,00

Continuación de la Tabla 3.

Sujeto 40 (200 pasos)	Medición	Exactitud (%)
19	201	99,50
20	203	98,50
21	201	99,50
22	201	99,50
23	201	99,50
24	202	99,00
25	201	99,50
26	202	99,00
27	201	99,50
28	201	99,50
29	201	99,50
30	201	99,50
31	200	100,00
32	201	99,50
33	202	99,00
34	201	99,50
35	201	99,50
36	200	100,00
37	200	100,00
38	200	100,00
39	201	99,50
40	201	99,50
41	201	99,50
42	201	99,50
43	202	99,00
44	201	99,50
45	201	99,50
46	201	99,50
47	201	99,50
48	202	99,00
49	201	99,50
50	201	99,50
Precisión (%)		99,52