

Proyecto Fin de Grado

Ingeniería de las Tecnologías Industriales

Desarrollo de aplicación de asistencia remota para
personas con necesidades especiales.

Autor: Tomás Acosta Almeda

Tutor: Manuel Perales Esteve

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015



Proyecto Fin de Grado
Ingeniería de Tecnologías Industriales

Desarrollo de aplicación de asistencia remota para personas con necesidades especiales.

Autor:

Tomás Acosta Almeda

Tutor:

Manuel Perales Esteve

Profesor Contratado Doctor

Dep. de Ingeniería electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Carrera: Desarrollo de aplicación de asistencia remota para personas con necesidades especiales.

Autor: Tomás Acosta Almeda

Tutor: Manuel Perales Esteve

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mi familia y a mis amigos.

Agradecimientos

A mi familia y a mis amigos por ayudarme a terminar esta etapa.

El proyecto consiste en el desarrollo de una aplicación de asistencia para personas con necesidades especiales. Se basa en un reloj programable que recoge los datos de movimiento de estas personas y es capaz de detectar caídas. Una vez detectada una caída el reloj da un tiempo de seguridad, si se pasa este tiempo y no hay respuesta por parte del usuario, entonces el reloj se pone en contacto con un móvil con sistema operativo Android (al que está conectado mediante Bluetooth) y éste envía un e-mail de aviso a una persona de confianza. El e-mail consiste en una notificación de caída y un mapa con las coordenadas del lugar donde se ha producido la caída.

En el proyecto se va a describir el proceso de aprendizaje de los distintos lenguajes de programación que han sido necesarios para llevarlo a cabo: C en el entorno de Pebble y Java en el entorno de Android. Así pues el proyecto en sí es el aprendizaje de ambos lenguajes y entornos y posteriormente la aplicación final.

Se va a describir cada parte por separado, primero Pebble y después la parte de Android, por último la unión de ambas en la aplicación final.

Abstract

This project is based in the development of a remote assistance app for people with special needs. It's based on the programmable smartwatch Pebble. Basically, the goal of the smartwatch is to monitorize the movement of these people (through the accelerometer sensor built in the watch) and detect a fall if it happened. In the case of a fall the watch will display a 15 seconds screen, when that time passes without user interaction the watch should connect, through Bluetooth technology, with an Android mobile device (phone, tablet, etc) which will send an emergency e-mail. These e-mail will warn about the possible fall and will have the coordinates of the fall.

This project will describe the process of learning two different programming environments: Pebble (C language) and Android (Java). The project should be considered as a whole : first the learning of those two environments (with different examples) and then the mix between the two, resulting in the final application.

Each part will be described separately and finally together for the final app.

Índice

Agradecimientos	ix
Resumen	xi
Abstract	xiii
Índice	xiv
Índice de Tablas	xvi
Índice de Figuras	xviii
Notación	xxi
1 Introducción	2
2 El reloj inteligente Pebble	6
2.1. <i>Elección del reloj</i>	6
2.1.1 Hardware	6
2.1.2 Software	7
2.1.3 Programación en Pebble	8
2.1.4 Estructura y particularidades	9
2.2. <i>Reconocimiento de caídas</i>	10
2.2.1 El sensor	10
2.2.2 Introducción a la detección de caídas	11
2.2.3 Pruebas para acotar los estados	12
2.2.4 Conclusiones extraídas de las pruebas	20
2.2.5 Determinando los umbrales	21
2.3. <i>Algoritmo de reconocimiento</i>	23
2.4. <i>Comunicación con el dispositivo Android</i>	25
2.5. <i>El reloj en funcionamiento normal</i>	26
2.6. <i>Problemas encontrados</i>	26

3	Dispositivo Android	29
3.1.	<i>Elección del teléfono</i>	29
3.1.1.	Hardware	29
3.1.2.	Software	30
3.1.3	Programación en Android	30
3.2.	<i>Tareas a realizar por el teléfono</i>	31
3.2.1.	Ubicación	31
3.2.2.	Conexión con Pebble	31
3.2.3	Envío automático de correo electrónico	31
4	Conclusiones y posibles mejoras	34
5	Anexos	36
A	<i>Código en Pebble</i>	36
A.1.	Pantalla principal "fd2.c"	36
A.2.	Pantalla de cuenta atrás "window2.c"	53
B	<i>Código en Android</i>	67
B.1.	MainActivity.java	79
	Referencias	80
	Índice de Conceptos	81
	Glosario	82

ÍNDICE DE TABLAS

Tabla 1. Listado de tareas para cada plataforma	3
Tabla 2. Datos de la prueba número 2	16
Tabla 3. Datos de la prueba número 3	17
Tabla 4. Datos de la prueba número 4	17
Tabla 5. Datos de la prueba número 5	18
Tabla 6. Datos de la prueba número 6	19
Tabla 7. Datos de comportamiento normal para una persona de 23 años	21
Tabla 8. Datos de comportamiento normal para una persona de 67 años	22
Tabla 9. Valores medios, máximos y mínimos sacados de las pruebas	22

ÍNDICE DE FIGURAS

Figura 1. El reloj Pebble	5
Figura 2. Distintos relojes inteligentes con pantallas táctiles	6
Figura 3. Reloj con pantalla de cristal rota	7
Figura 4. EZ430, Pebble y Sony Smartwatch	7
Figura 5. Captura de pantalla del sitio CloudPebble	8
Figura 6. Captura de pantalla del sitio developers.pebble.com	9
Figura 7. Estructura de pila	10
Figura 8. Distribución de ejes en el reloj	11
Figura 9. Forma típica de la aceleración durante una caída	12
Figura 10. Prueba de caída número 1	13
Figura 11. Detalle de la prueba 1	14

Figura 12. Gráfico de la prueba 2	15
Figura 13. Gráfico de la prueba 3	16
Figura 14. Gráfico de la prueba 4	17
Figura 15. Gráfico de la prueba 5	18
Figura 16. Gráfico de la prueba 6	19
Figura 17. Gráfica tipo	20
Figura 18. Gráfica tipo con zonas	21
Figura 19. Gráfica con estados acotados	22
Figura 20. Diagrama de flujo del algoritmo	24
Figura 21. Sistema de comunicación bidireccional	25
Figura 22. Captura de pantalla	26
Figura 23. Huawei Y330	29
Figura 24. Libro seguido por el alumno	30
Figura 25. Página de desarrolladores de Google	31
Figura 26. Apariencia del correo	32
Figura 27. Enlace de Google Maps	33

Notación

acel	Módulo del vector aceleración (sacado del acelerómetro del reloj)
API	Librería (en referencia a la librería JavaMail)
SDK	Software Development Kit, herramienta de desarrollo.
■	Como queríamos demostrar
e.o.c.	En cualquier otro caso
e	número e
Re	Parte real
Im	Parte imaginaria
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\sin^x y$	Función seno de x elevado a y
$\cos^x y$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
$\partial y \partial x$	Derivada parcial de y respecto
x°	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
SNR	Signal-to-noise ratio

MSE	Minimum square error
:	Tal que
<	Menor o igual
>	Mayor o igual
\	Backslash
⇔	Si y sólo si

1 INTRODUCCIÓN

El mundo y el sistema en el que vivimos han aportado grandes cosas a los humanos como civilización, tenemos tecnología de todo tipo, avances médicos, etc . Sin embargo, el acelerado ritmo de la vida nos lleva, a veces, a dejar de lado a las personas que no pueden seguirlo. Hay quien considere esto algo paradójico, el mismo sistema que favorece que cada vez vivamos más y con más comodidades es el mismo que nos obliga a apartar de nuestra vida, por ejemplo, a familiares de la tercera edad porque se convierten en un estorbo. Estas personas acaban solas y aisladas y por tanto más vulnerables a contratiempos.

Uno de estos contratiempos son las caídas domésticas. Las caídas involuntarias suponen la segunda causa de muerte por lesiones en el mundo, siendo el colectivo de la tercera edad el más perjudicado, según la Organización Mundial de la Salud, OMS. En la actualidad existen métodos para reconocer estas caídas y mandar una señal de ayuda, siendo el más utilizado el del collar detector, sin embargo estos métodos necesitan un estímulo por parte del usuario (hay que pulsar un botón en el collar para pedir ayuda) y por tanto no cubren el caso en el que el usuario no puede interaccionar (si está inconsciente por ejemplo).

Actualmente los “wearable” están en auge, son un tipo de dispositivo que llevamos puesto. Se caracterizan por su reducido tamaño y por ofrecer una serie de prestaciones que “supuestamente” nuestros teléfonos no nos pueden dar. Ahora mismo está por ver si estos dispositivos triunfarán y encontrarán un hueco en nuestras vidas, y si, así como pasó con los smartphones, sustituirán a los relojes normales. La apuesta personal del autor es que estos dispositivos fracasarán a la hora de llegar al público general (no tendremos todos un smartwatch como todos tenemos un teléfono con internet), pero triunfarán al especializarse en ámbitos concretos como el deporte o la salud.

El autor del proyecto piensa que la tecnología y la ingeniería deben tener una finalidad social y que ya que ha estudiado tantos años prefiere intentar hacer algún bien al conjunto de la humanidad antes que poner su intelecto a trabajar para que podamos ver los mensajes de “whatsapp” en tu muñeca antes que en tu teléfono.

El objetivo del alumno en este proyecto ha sido aprender los lenguajes de programación necesarios para llevar a cabo el proyecto. De hecho en un principio no se consideró aprender Java y Android, pero dadas las limitaciones del reloj (procesador poco potente e incapacidad de acceder a internet por su propia cuenta) al final se incluyeron en el proyecto y acabaron siendo partes principales de él (en cuanto a importancia y horas de trabajo)

La finalidad de este proyecto es presentar una aplicación de reconocimiento de caídas que las detecte y avise en consecuencia, sin intervención del usuario. La aplicación está destinada a personas de la tercera edad o personas con necesidades especiales que vivan solas y puedan necesitar asistencia. La aplicación final funciona de la siguiente manera: La persona que necesita asistencia lleva consigo, en la muñeca, el reloj inteligente, cuando se produce una caída dicho reloj la detecta y si no se trata de un falso positivo, tras 15 segundos el reloj contacta con un dispositivo Android con conexión a internet. Dicho dispositivo manda un email de ayuda a una persona de contacto, con las coordenadas exactas de la caída.

Por tanto podría dividirse el proyecto en dos grandes secciones, las tareas a realizar por el reloj y las que tiene

que realizar el dispositivo Android.



<i>Reloj Inteligente</i>	<i>Dispositivo Android</i>
1- Actuar como un reloj en funcionamiento normal, mientras que se atiende a los datos del acelerómetro.	1- Esperar una señal por parte del reloj, mientras tanto no hacer nada.
2- Comprobar que la conexión reloj-móvil sigue activa (mediante Bluetooth)	2- Obtener las coordenadas del dispositivo mediante GPS.
3- Aplicar un algoritmo de reconocimiento de caídas a los datos del acelerómetro.	4- Atender una señal del reloj y saber gestionarla.
5- Comunicar al dispositivo una posible caída si así ha determinado el algoritmo.	4- Enviar el mensaje de ayuda.

Tabla 1 – Listado de tareas para cada plataforma

En el proyecto se explicarán cada uno de estos bloques en su sección correspondiente y en el anexo se incluirá el código comentado.

2 EL RELOJ INTELIGENTE PEBBLE

“Stay connected to things that matter”

Sitio web del reloj Pebble

SE ha escogido para la elaboración de este proyecto, un reloj inteligente llamado Pebble, que traducido del inglés significa piedra pequeña, haciendo alusión al reducido peso y tamaño del mismo. En la figura 1 podemos ver el modelo escogido finalmente para llevar a cabo el proyecto:



Fig. 1 El reloj Pebble

De los relojes programables que había disponibles en el mercado en el momento se escogió este modelo por una serie de razones que se explican a continuación.

2.1. Elección del reloj

2.1.1 Hardware

Había dos condicionantes principales a la hora de escoger el modelo, el primero, la pantalla. La mayoría de los relojes inteligentes que existen utilizan pantallas táctiles (LCD, OLED, etc) retroiluminadas, como las de los smartphones pero a menor escala. Estas pantallas presentaban un gran inconveniente: el consumo de batería. La mayoría de los smartwatches actuales necesitan una recarga de batería diaria, lo cual no es en absoluto práctico para y da lugar a situaciones adversas, como que se acabe la batería y se produzca una caída o que a la persona que necesita usar el reloj se le olvide ponérselo después de cargarlo. El modelo escogido soluciona ese problema usando una pantalla de tinta inteligente (e-ink) de bajo consumo y sin problemas de luminosidad en el exterior. La batería del reloj dura alrededor de una semana, lo cual es mucho más conveniente.



Fig. 2 Distintos relojes inteligentes con pantallas táctiles.

Otro de los condicionantes que existían era la propia naturaleza del proyecto, que involucra caídas, no sólo en su funcionamiento final de cara al usuario, sino sobretodo en el periodo de prueba que se ha tenido que llevar a cabo para recopilar datos y ver si funcionaba la aplicación. De ahí que no se haya escogido un reloj con carcasa de metal ni con partes de cristal, el modelo escogido es de plástico entero y ha aguantado sin problema todas las pruebas de caídas que se le han hecho.



Fig. 3 Reloj con pantalla de cristal rota.

Dentro de las consideraciones de hardware también hay que destacar que el reloj escogido es resistente al agua, con lo cual el usuario no tiene por qué quitárselo en la ducha (que es justamente donde se producen la mayoría de los accidentes domésticos).

Además el Pebble era de los relojes más asequibles del mercado a la hora de comprarlo, costando menos de la mitad que muchos de sus competidores.

2.1.2 Software

El principal condicionante es que fuese programable, porque si no, no se podría desarrollar ninguna aplicación sin convertirse en desarrollador oficial de alguna compañía. Por tanto se descartaron la mayoría de los modelos que había disponibles, pues casi todos eran plataformas cerradas. Finalmente se consideraron 3 modelos distintos: Sony Smartwatch, Pebble y el EZ-430Chronos de Texas instrument:



Fig. 4 EZ430, Pebble y Sony Smartwatch

Se descartó el modelo de Sony porque realmente se lanzó al mercado como un reloj no programable, pero Sony decidió abrir el SDK a todo el mundo y hacerlo programable, pero debido a que esto había ocurrido hace poco tiempo se desestimó usarlo por falta de soporte y de comunidad activa programando para la plataforma.

El Reloj de Texas Instrument también se descartó por tener una interfaz de usuario y programación mucho menos atractivas que las del Pebble.

Y es que esa fue la principal baza de esta plataforma, de cara a los programadores parecía un entorno muchísimo más atractivo y simple, con una gran cantidad de personas desarrollando aplicaciones así como muchísima información, tutoriales y guías para empezar.

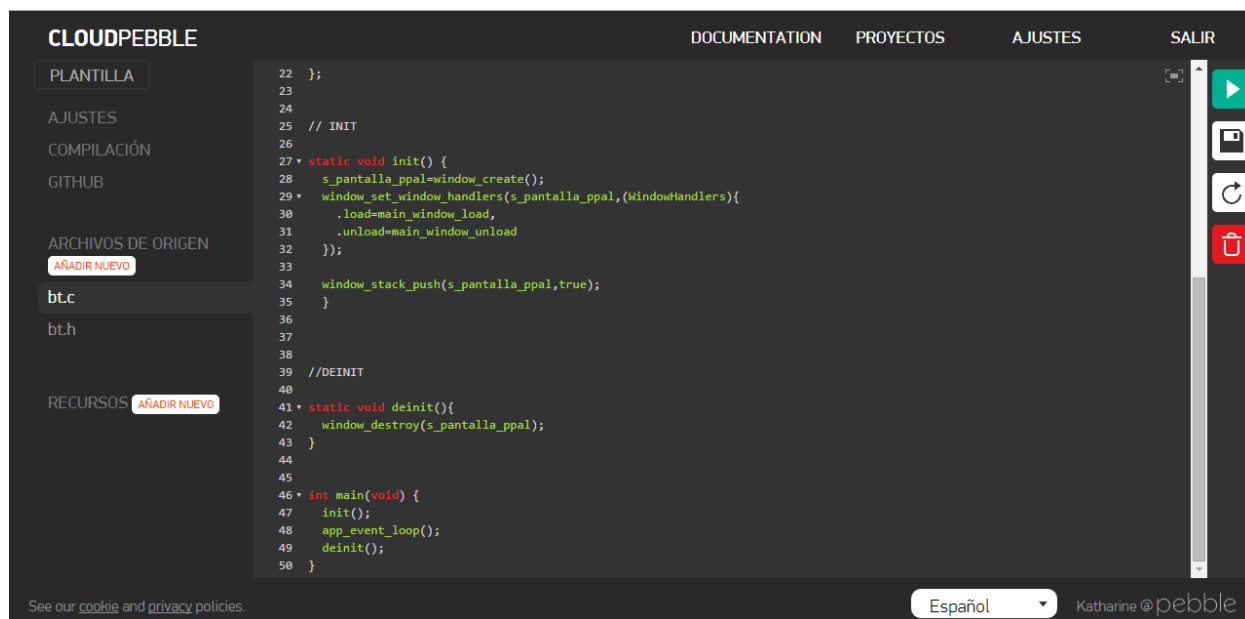
Hay que tener en cuenta también que el Pebble se programa en lenguaje C, que yo ya conocía y había estudiado en la carrera. Además disponen de un compilador en línea con un emulador del reloj, con lo que pude probarlo un poco antes de decidirme completamente.

El reloj contaba con los elementos principales del proyecto: Bluetooth para las comunicaciones y un acelerómetro para detectar caídas.

Por todas estas razones se acabó escogiendo el reloj que se puede ver en la figura 1, por un precio cercano a los 100 euros, fue comprado a través de internet, pero también se puede adquirir en tiendas de electrónica o grandes almacenes.

2.1.3 Programación en Pebble

Pebble utiliza el lenguaje de programación C, con algunas particularidades. Como durante la carrera se utiliza bastante este lenguaje (y uno de los objetivos del Proyecto Fin de Grado es poner en práctica los conocimientos adquiridos) no resultó complicado familiarizarse con el entorno de programación. Todos los programas de este proyecto se han hecho a través del compilador online que ofrece la plataforma: www.cloudpebble.net. Como nota negativa cabe señalar que actualmente no disponen de SDK para Windows, y por tanto hay que utilizar un emulador de Linux o similar para poder usarlo. El principal problema del compilador en línea es que la depuración de errores no es buena, el compilador no te “ayuda” mucho por así decirlo, dando lugar a situaciones en las que no se avanza por un simple error de sintáxis o similar.



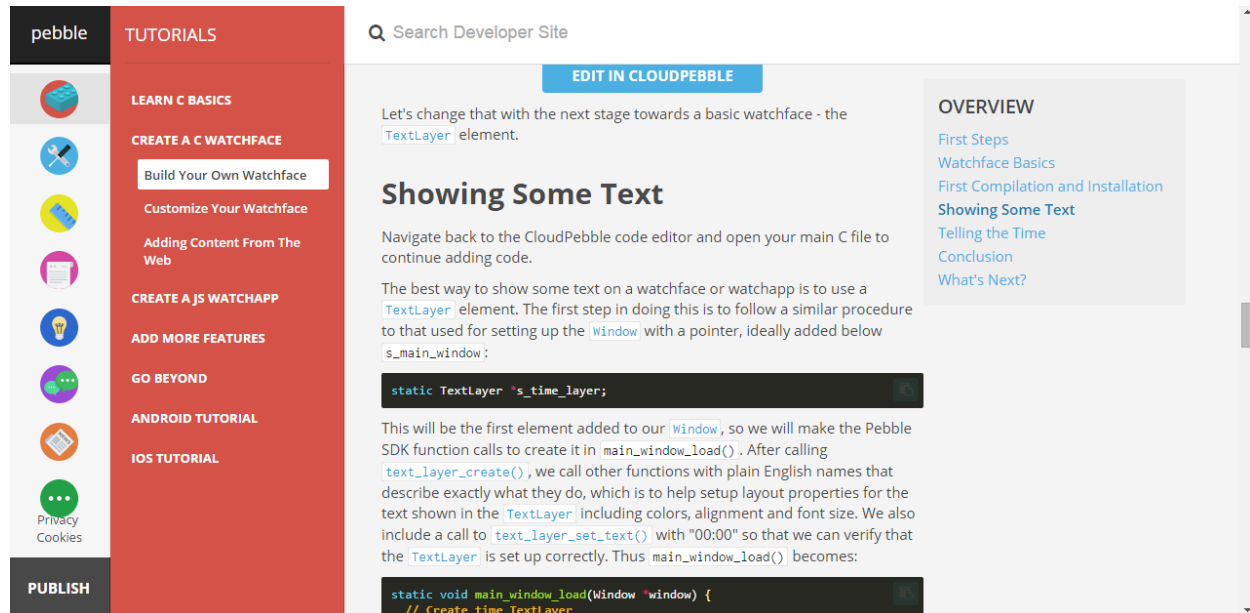
```
22  };
23
24
25  // INIET
26
27  static void init() {
28      s_pantalla_ppal=window_create();
29      window_set_window_handlers(s_pantalla_ppal,(WindowHandlers){
30          .load=main_window_load,
31          .unload=main_window_unload
32      });
33
34      window_stack_push(s_pantalla_ppal,true);
35  }
36
37
38
39  //DEINIT
40
41  static void deinit(){
42      window_destroy(s_pantalla_ppal);
43  }
44
45
46  int main(void) {
47      init();
48      app_event_loop();
49      deinit();
50  }
```

Fig. 5 Captura de pantalla del sitio CloudPebble

Como ya se ha comentado antes, Pebble se basa en C, pero con una estructura particular para los programas así como una manera especial de gestionar los recursos gráficos y el uso de sensores. Una persona que sólo sepa C

puede leer un código de Pebble y reconocer algunas estructuras, pero no entender lo que está haciendo el programa. Por tanto es necesario un periodo de aprendizaje.

Para aprender sobre este entorno y empezar a programar se usó la bibliografía disponible en la propia página de la empresa. Se empezó con los tutoriales más básicos y después con las guías más avanzadas.

The image is a screenshot of the Pebble Developer Site. On the left, there is a navigation menu with categories like 'LEARN C BASICS', 'CREATE A C WATCHFACE', 'CREATE A JS WATCHAPP', 'GO BEYOND', 'ANDROID TUTORIAL', and 'IOS TUTORIAL'. The main content area is titled 'Showing Some Text' and includes an 'EDIT IN CLOUDPEBBLE' button. The text explains how to use the `TextLayer` element and shows a code snippet:

```
static TextLayer *s_time_layer;
```

 and

```
static void main_window_load(Window *window) {  
    // Create time TextLayer
```

. An 'OVERVIEW' sidebar on the right lists links like 'First Steps', 'Watchface Basics', and 'Showing Some Text'.

Fig. 6 Captura de pantalla del sitio developers.pebble.com

A continuación se va a explicar la estructura básica de una aplicación y las particularidades de esta plataforma, para que se pueda entender el resto del proyecto. Más adelante se explicará una de las aplicaciones que desarrollaron inicialmente, con un poco más de complejidad (el código estará en los apéndices del proyecto).

2.1.4 Estructura y particularidades.

La estructura de un programa en Pebble es similar a la de uno en C. Primero se empieza incluyendo las librerías y definiendo las constantes y variables globales. Después van las definiciones de funciones de todo tipo. La estructura general se basa en un bucle principal y en funciones de iniciación y deiniciación. Dentro del bucle principal se pueden crear nuevas ventanas, capas de texto, animaciones, llamar a servicios, etc... Las particularidades de este sistema son:

Es necesario iniciar una ventana principal para mostrar cualquier cosa por pantalla, incluso si no vamos a mostrar nada, la pantalla debe existir. Cada pantalla podría entenderse como un hilo de ejecución (de hecho, cuando una ventana no está activa su código no se ejecuta hasta que se vuelva a ella). Para cambiar entre ventanas activas y no activas se utiliza un sistema de “push” y “pop”, es decir, un sistema de pila.

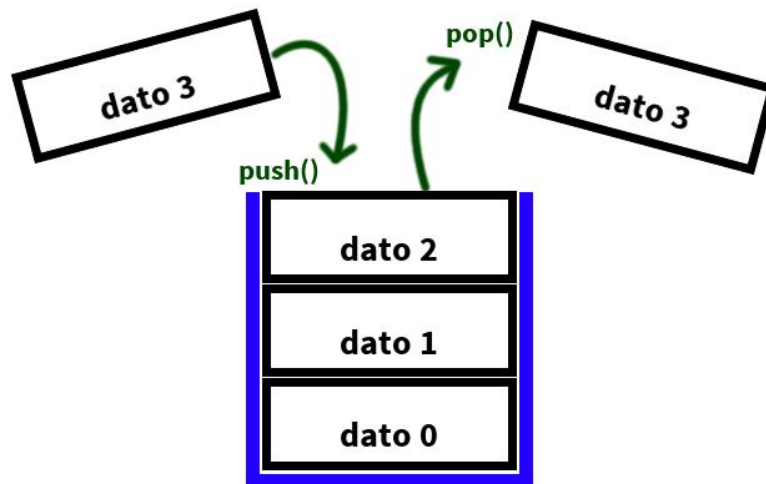


Fig. 7 Estructura de pila

Para mostrar cualquier cosa por pantalla hay que crear una capa, puesto que Pebble funciona por capas. Si se quiere mostrar un mensaje de “HOLA MUNDO”, primero hay que definir una capa, asignarla a una ventana, especificar cuánto espacio de la pantalla va a ocupar dicha capa y por último decir qué va a ir ahí (en nuestro caso el mensaje antes comentado).

Para poder utilizar cualquiera de los sensores o botones del reloj, hay que suscribirse a ese servicio y utilizar las funciones predefinidas para manejarlos (por ejemplo, el manejador de datos del acelerómetro está ya definido, lo que no quita que nosotros podamos hacer una llamada a otra función o modificar esa como nosotros queramos). En el momento en el que no se quiera seguir usando esos servicios hay que anular la suscripción.

2.2. Reconocimiento de caídas

En este proyecto el reconocimiento de caídas va a realizarse a través de un algoritmo implementado en el reloj. El reloj recibe y atiende los datos del acelerómetro y él mismo los gestiona y les aplica el algoritmo.

Se va a explicar en qué consiste dicho algoritmo así como el fundamento teórico en el que se basa. También se explicará cómo se ha investigado y las pruebas que se han hecho para acotar su funcionamiento.

2.2.1 El sensor.

El reloj escogido tiene un acelerómetro de 3 ejes incorporado, con un rango de actuación de $\pm 4000\text{mG}$. El Gal o “G” a secas es una unidad de medida de la aceleración y se define como $1\text{Gal} = 0.001 \text{ m/s}^2$.

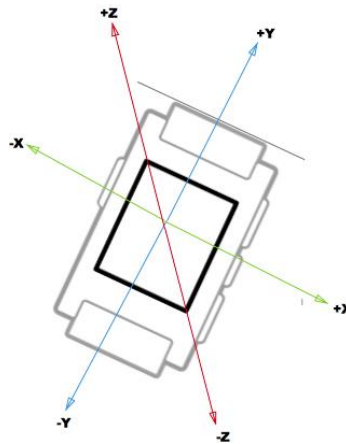


Fig. 8 Distribución de ejes en el reloj.

Se ha utilizado este sensor para detectar las caídas por la falta de otro método mejor, el reloj no dispone de un sensor de velocidad o posición, con lo cual todo el algoritmo se ha basado en este. Durante el periodo de aprendizaje varias aplicaciones resultaron inviables debido a esto, por ejemplo, se desestimó una aplicación para medir la velocidad cuando se va en bicicleta, ya que al disponer sólo de valores de aceleración y no de velocidad, los resultados integrados no eran para nada precisos.

El acelerómetro tiene 3 ejes y por tanto 6 direcciones posibles, sin embargo, debido a la naturaleza impredecible de una caída, no se tienen en cuenta las direcciones para detectarla. Una persona puede caerse de distintas maneras, puede mover o no los brazos, hacer movimientos bruscos en una u otra dirección, por tanto éste no debería ser un factor decisivo.

Lo que nos deja con nuestro único recurso disponible para la detección: el módulo de la aceleración. Más adelante se va a explicar cómo se ha conseguido esto.

En cuanto a software, el sistema es el siguiente: Hay que suscribirse al servicio y determinar una serie de parámetros, como a cada cuánto tiempo se quiere que se lean los valores de aceleración (en el código esto es el “sampling rate”) o qué número de muestras se quiere en cada ciclo. También hay que definir la función manejadora a la que se llama cada vez que se leen los datos, esta función es además la encargada de aplicar el algoritmo.

Hay una razón por la que se ha decidido implementar y gestionar el algoritmo en el reloj y no en el teléfono. A primera vista sería más provechoso mandar los datos recogidos al teléfono y procesarlos en él, ya que disponemos de un procesador mucho más potente y más recursos. Sin embargo, existe un problema con el software del reloj, concretamente con la “API” de conexión con otros dispositivos “AppMessage”. Esta aplicación tiende a fallar si se envían muchos datos, y mientras más datos se envían más errores comete. Por ello la comunidad de Pebble en general recomienda gestionar los datos en el reloj o comprimir los datos y después mandarlos. Como tenemos que detectar caídas no nos podemos permitir ninguna clase de retraso con los datos, así que se ha decidido gestionar los datos en el reloj. Este fallo en la API todavía no ha sido resuelto, se entiende que es uno de los problemas asociados a usar una tecnología relativamente nueva.

2.2.2 Introducción a la detección y al algoritmo.

Tenemos pues que detectar una caída utilizando el módulo de la aceleración del reloj. Para hacerlo con los recursos que tenemos disponibles primero tenemos que entender como funciona el sensor y qué datos va a proporcionar.

Para ello se programó una aplicación que recogía los datos del sensor y calculaba la suma del cuadrado de sus componentes. El sistema Pebble es un poco limitado en cuanto a mostrar datos por pantalla ya que no se pueden mostrar números decimales, así que en vez de calcular el módulo de la aceleración todos los datos están hechos con el módulo al cuadrado, es decir, la suma del cuadrado de cada componente (si hiciese la raíz daría un número decimal y no podría verlo por pantalla). Este inconveniente va a hacer que tengamos que operar con números

bastante grandes y aparatosos de manejar.

El programa que se hizo no sólo dejaba observar los datos en tiempo real, sino que mostraba el valor de cada componente y además tenía un sistema para registrar dichos datos y verlos uno a uno, pudiendo ver valores anteriores o posteriores. Es decir, se guardaban los valores en un registro y después permitía ver ese registro ordenadamente. Esto se hizo para poder analizar mejor los datos y poder reconocer momentos concretos, por ejemplo, si hago un movimiento brusco, puedo saber en qué momento ha pasado (hay un contador para organizar mejor el registro) y puedo reconocer el movimiento porque los valores aumentan mucho.

¿Cómo se reconoce una caída sólo con un acelerómetro? Una de las respuestas a esta pregunta es: por etapas. Estructurando una caída por etapas concretas e identificándolas en orden. En este proyecto y tras investigar sobre el tema se decidió que para la detección se usarían 4 etapas distintas, 4 condiciones distintas que deben darse en orden y en tiempos establecidos. Estas etapas son:

- 1- Comportamiento normal.
- 2- Caída libre.
- 3- Impacto.
- 4- Vuelta a comportamiento normal.

2.2.3 Pruebas para acotar los estados.

Por establecer cierta conexión con el código, vamos a llamar a nuestro valor del acelerómetro (la suma del cuadrado de cada componente) como “*acel*”. Este es un valor siempre positivo, y aunque en teoría podría ser cero, realmente nunca llega a serlo. Cómo sólo tenemos este valor la cuestión es acotar los umbrales en los que se mueve según cada etapa.

Además también es importantísimo el tiempo, por ello se ha cogido la frecuencia de muestreo más pequeña que permite el reloj, que es un valor nuevo cada 0.1 segundos. Así pues nuestro periodo de muestreo o salto va a ser de 0.1 segundos.

El paso inicial fue investigar sobre el tema y familiarizarse con la forma que tiene una gráfica de caída (basada en el valor del módulo de la aceleración). Esta es la forma típica:

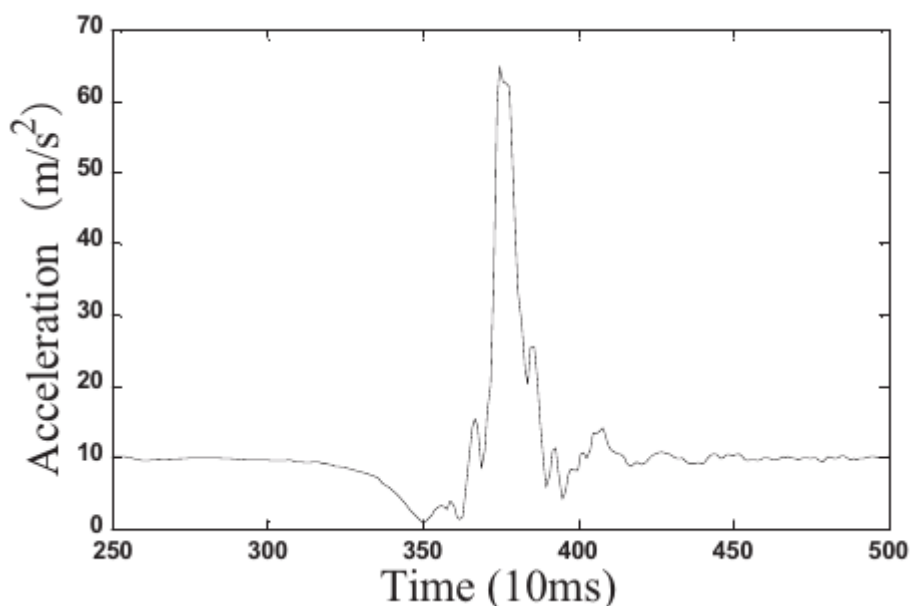


Fig. 9 Forma típica de la aceleración durante una caída.

Las unidades son indiferentes ahora mismo, lo prioritario es la forma de la gráfica. Se observa que empieza y

acaba en una línea (no completamente recta) que está alrededor del valor de aceleración de 10 metros por segundo al cuadrado, esos valores corresponden a lo que hemos llamado funcionamiento normal. Un poco después de los 300 milisegundos se observa una bajada brusca del valor y acto seguido un valor muy alto y muy brusco, eso correspondería a la caída libre (valor cercano a cero) seguido del impacto (valor momentáneo pero muy alto). Después del impacto volvemos a comportamiento normal.

Después de haber estudiado un poco este tema decidí comprobar por mi cuenta si esta gráfica se cumplía en la realidad, para lo cual me descargué una aplicación gratuita de la tienda de aplicaciones de Pebble, que mostraba por pantalla esta clase de gráficas. Además con mi propia aplicación de registro de datos pude dibujar el conjunto de gráficas que viene a continuación. Las pruebas se hicieron imitando, en la medida de lo posible, una caída fortuita (de diversas maneras, hacia adelante, hacia atrás, de manera brusca, etc). En todos los casos se trata de una caída de como mínimo medio metro de altura (la altura media hasta la cintura).

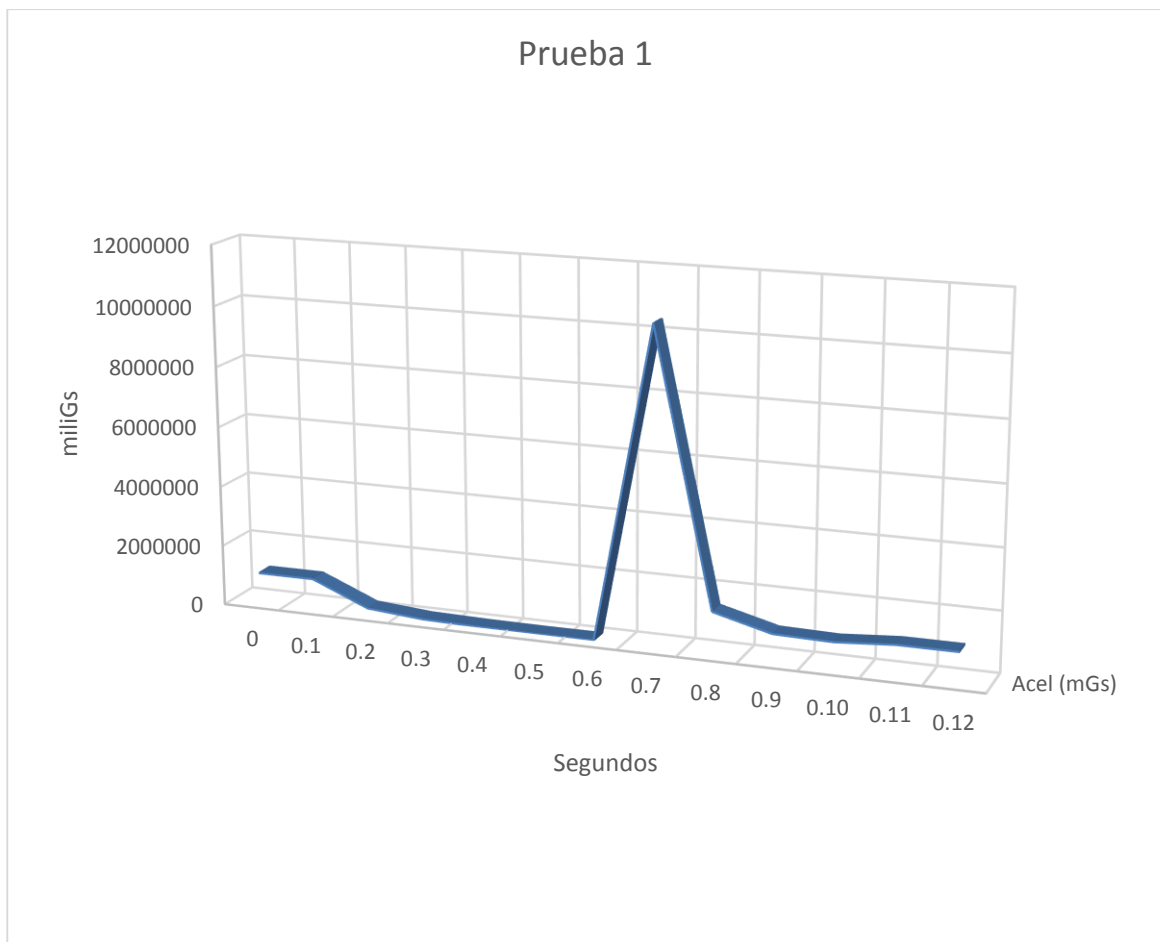


Fig. 10 Prueba de caída número 1.

Se puede observar una forma similar, no se parece tanto debido a dos factores. Primero, la escala no ayuda a ver la forma claramente, esto es debido a que el valor del impacto (el pico que llega a los 20 millones) hace que se pierda detalle de todo lo demás, concretamente de la parte de caída libre (tramo de 0.2 a 0.7 segundos), que se ve mucho menos pronunciada de lo que es en realidad. Para no perder esa información, adjunto un detalle de la zona de caída libre:

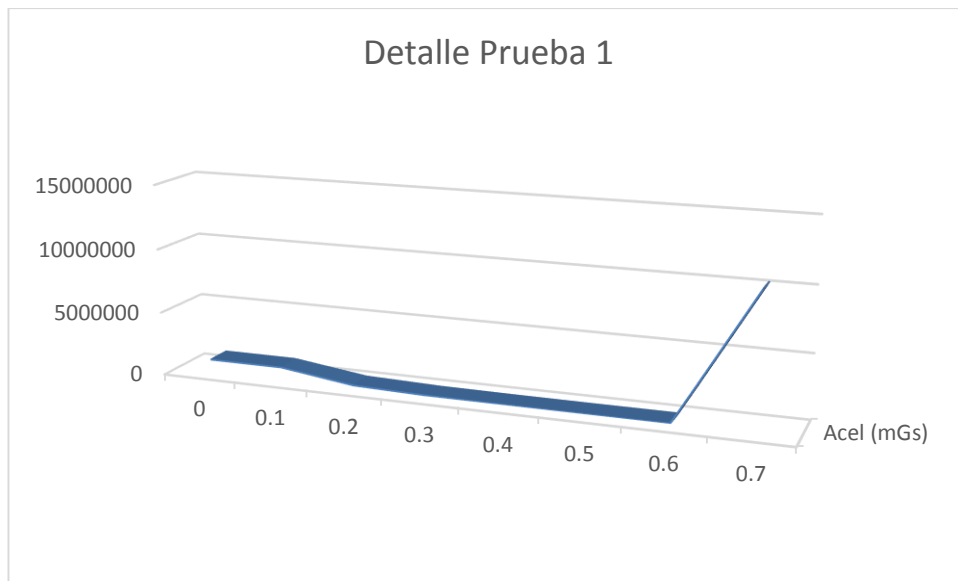


Fig. 11 Detalle de la prueba 1.

Si además observamos los datos en los que se ha basado esta gráfica:

TIEMPO	ACEL
0	1000000
0.1	945000
0.2	576000
0.3	294000
0.4	35000
0.5	3900
0.6	704
0.7	8900
0.8	20000000
0.9	3000000
0.10	1000090
0.11	1020000
0.12	1000300

Tabla 1 Datos de la prueba 1

Se puede ver que el comportamiento normal ronda cerca 1 millón (1M), y que alrededor de los 0.3 segundos es cuando se está produciendo la caída libre, que alcanza su mínimo para $t = 0.6$ segundos, dos pasos después se

produce el impacto (20M) y después se vuelve poco a poco al comportamiento normal.

A continuación voy a presentar los datos de las pruebas que se realizaron para entender la forma de la gráfica y la naturaleza del proceso.

2.2.3.1 Prueba número 2

Para las subsiguientes pruebas se ha establecido como límite del eje vertical el valor 3M, para poder observar mejor el resto de la gráfica sin perder detalle.

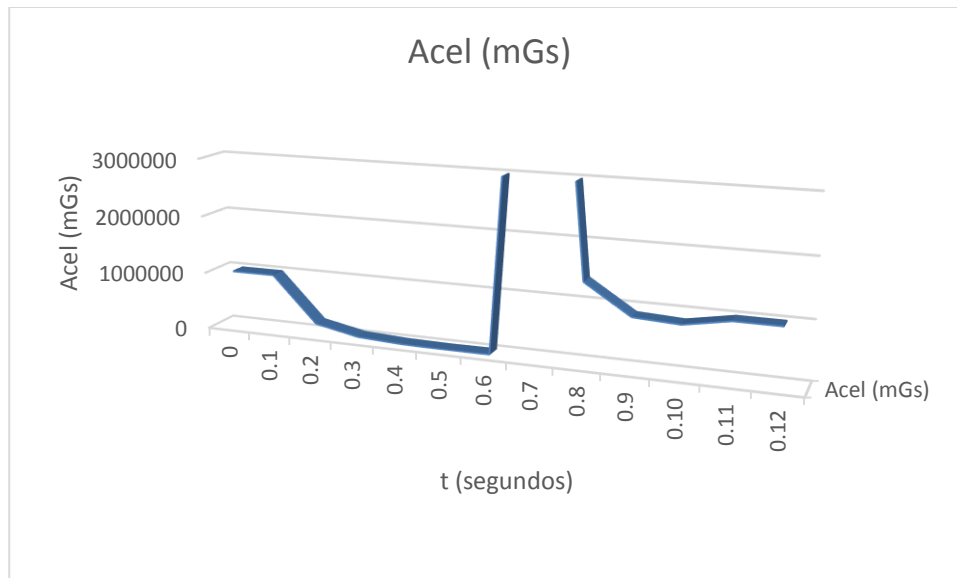


Fig. 12 Gráfico prueba 2

Tiempo(s)	Acel (mGs)
0	1000000
0.1	3400
0.2	5600
0.3	294000
0.4	4400
0.5	576
0.6	1000
0.7	136000
0.8	24000000
0.9	1400000
0.10	1000090
0.11	1020000

0.12	1000300
------	---------

Tabla 2 Datos de la prueba número 2.

Hay que recordar que la gráfica está cortada, pero podemos ver en los datos que el impacto real llega hasta el valor de 24M.

2.2.3.2 Prueba número 3

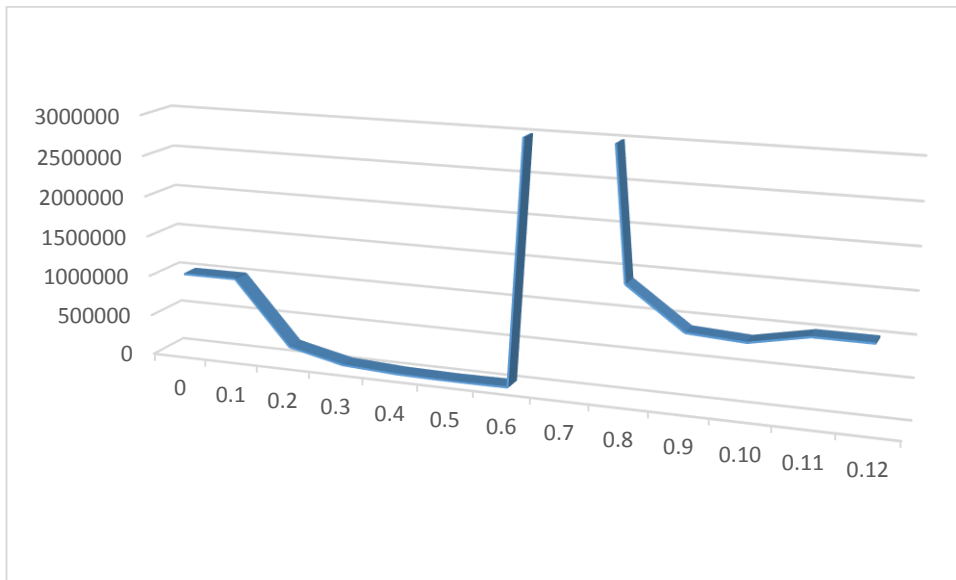


Fig. 13 Gráfico prueba 3

Tiempo(s)	Acel (mGs)
0	1000000
0.1	1200000
0.2	102000
0.3	643000
0.4	115000
0.5	32000
0.6	8800
0.7	2900
0.8	2100
0.9	23900000
0.10	4700000

0.11	1020000
0.12	1000300

Tabla 3 Datos de la prueba número 3.

2.2.3.3 Prueba número 4

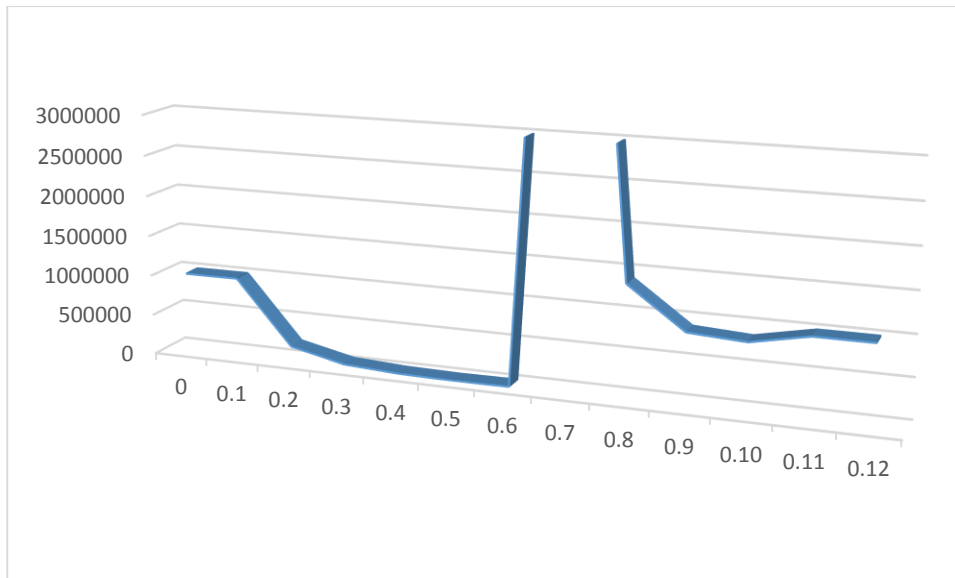


Fig. 14 Gráfico de la prueba 4

Tiempo(s)	Acel (mGs)
0	1000000
0.1	1020000
0.2	750000
0.3	263000
0.4	105000
0.5	12000
0.6	8100
0.7	9700
0.8	384
0.9	2100
0.10	12000000

0.11	2400000
0.12	1000000

Tabla 4 Datos de la prueba número 4.

2.2.3.4 Prueba número 5

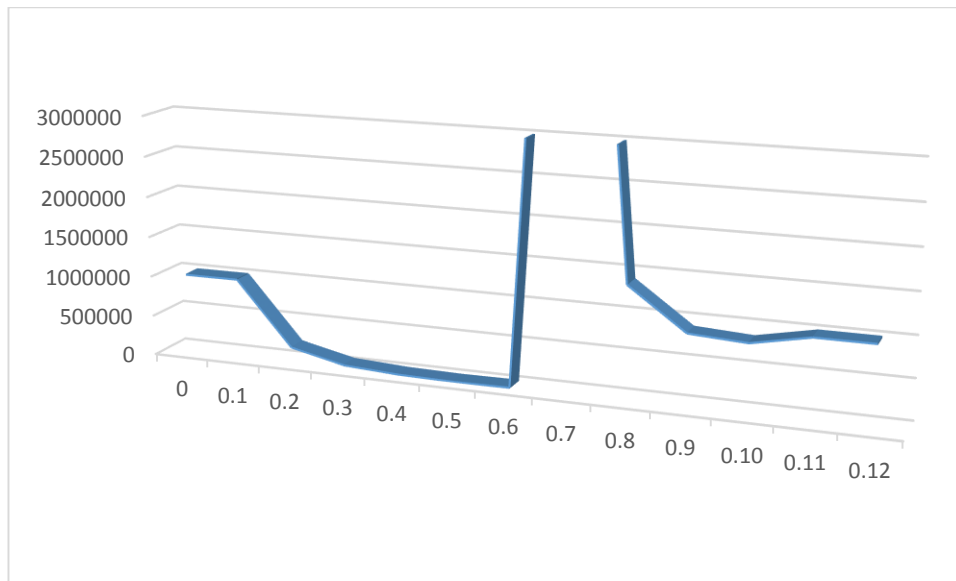


Fig. 15 Gráfico de la prueba 5.

Tiempo(s)	Acel (mGs)
0	1000000
0.1	1002000
0.2	326000
0.3	13000
0.4	5700
0.5	4900
0.6	1800
0.7	4700
0.8	1500000
0.9	3000000

0.10	1900000
0.11	800000
0.12	1000000

Tabla 5 Datos de la prueba número 5.

2.2.3.5 Prueba número 6

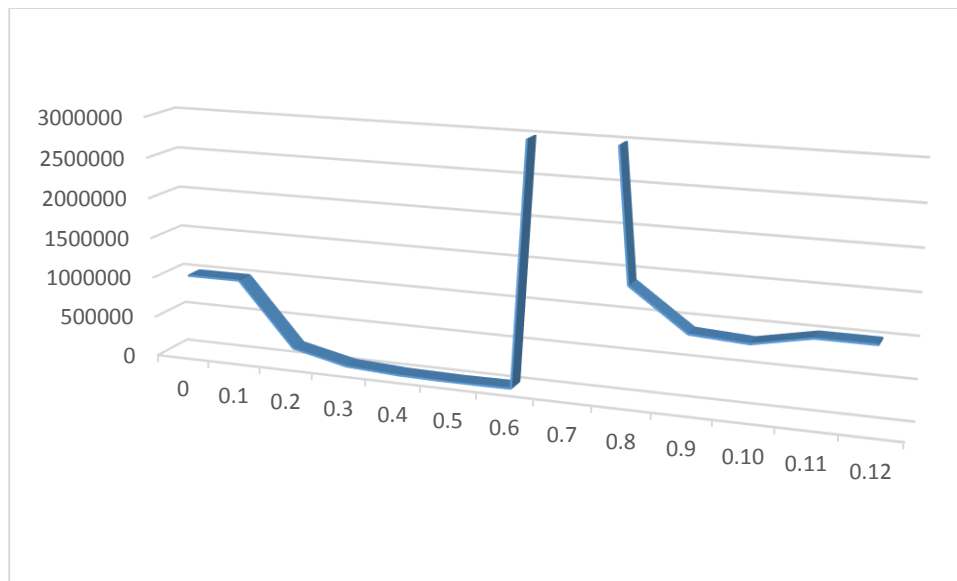


Fig. 16 Gráfico de la prueba 6

Tiempo(s)	Acel (mGs)
0	1000000
0.1	1002000
0.2	210000
0.3	58000
0.4	20000
0.5	9500
0.6	16000
0.7	10400000
0.8	1400000
0.9	900000
0.10	860000

0.11	1000000
0.12	1000000

Tabla 6 Datos de la prueba número 6.

2.2.4 Conclusiones extraídas de las pruebas.

Con las pruebas realizadas podemos entender como funciona una caída cuando el reloj está en la muñeca del usuario. Además de identificar las distintas partes de la caída, podemos estimar también su duración aproximada, cosa que será clave para el algoritmo de reconocimiento.

Podemos dar por identificadas las siguientes etapas, tomando como referencia una gráfica tipo (donde el máximo del impacto se ha minimizado para no estropear la escala de la gráfica):

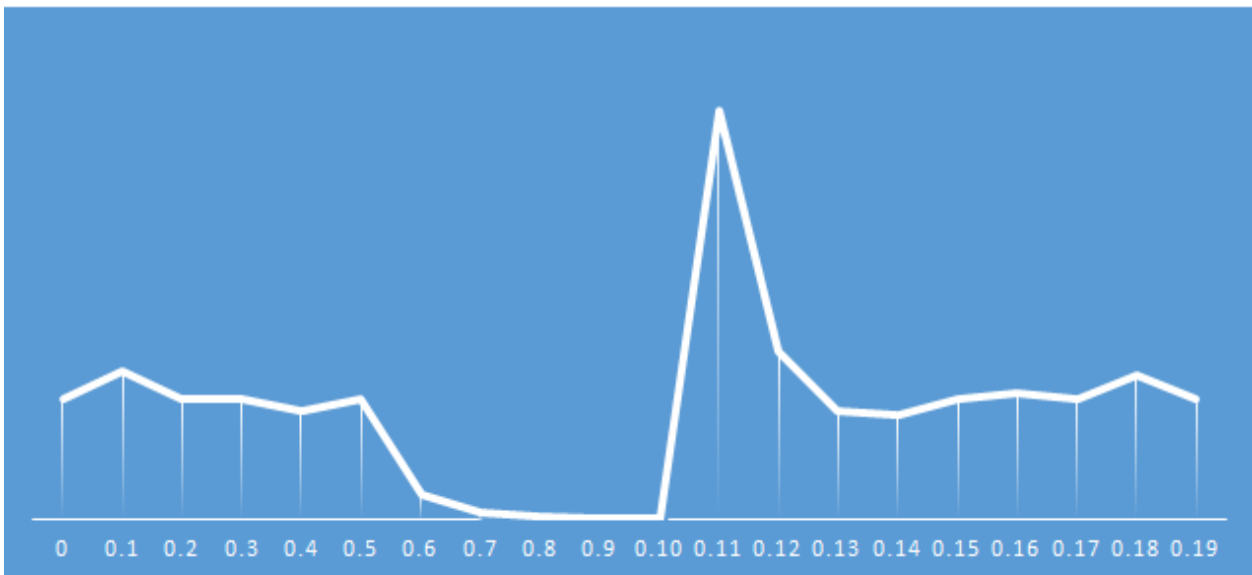


Fig. 17 Gráfica tipo.

Podemos definir las zonas como:

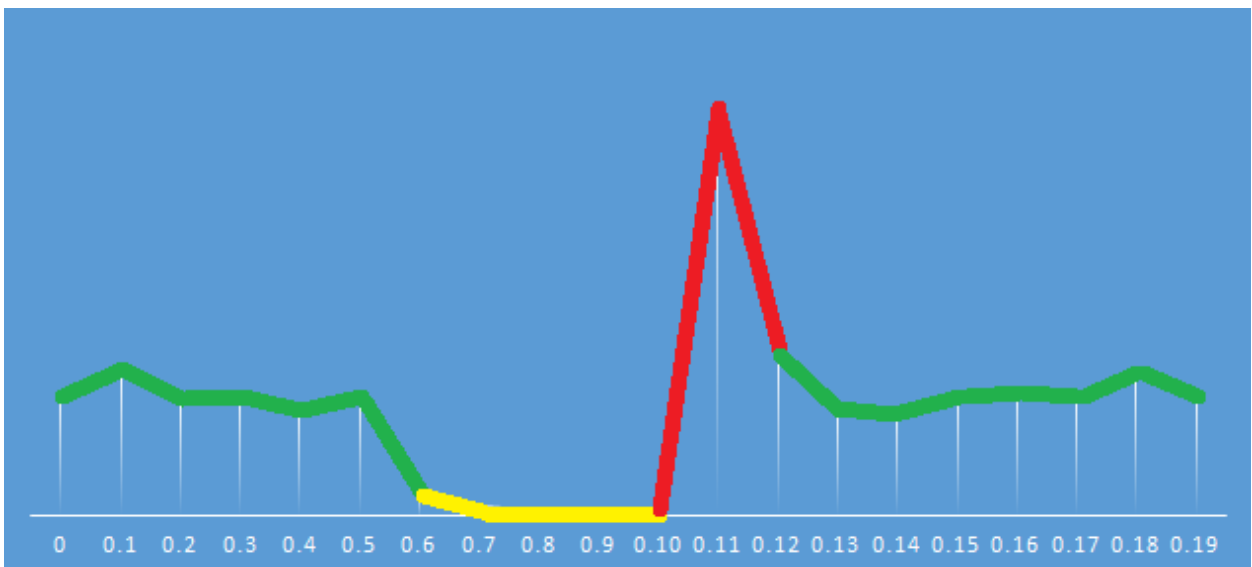


Fig. 18 Gráfica tipo con zonas.

- **Verde:** Comportamiento normal, todavía hay que delimitar cuales son los umbrales superior e inferior. En un principio *no tiene limitación de tiempo*.
- **Amarillo:** Caída libre, dura *en torno a 5 pasos de muestreo* y tiene unos valores muy bajos, se delimitarán más adelante.
- **Rojo:** Impacto, dura *aproximadamente 2 pasos de muestreo*.

Estos valores aproximados de la duración de cada etapa se han tomado teniendo en cuenta las pruebas anteriores y pueden ser modificados en el código si conviene.

2.2.5 Determinando los umbrales de cada etapa.

Una vez tenemos la forma de la gráfica y la duración aproximada de cada etapa, sólo queda determinar qué entendemos por cada etapa, es decir determinar las cotas que deciden cuando un valor de acel pertenece a una etapa u otra.

Para la zona de funcionamiento normal se ha usado una vez más el software creado por el alumno, pero esta vez con una ligera modificación: sólo toma 2000 muestras y además muestra por pantalla el valor máximo y mínimo de acel (como estamos buscando umbrales, es lo que nos interesa). Se han hecho diversas pruebas, tanto por una persona de 23 años (activa) como por una de 67 (movimientos menos bruscos) y estos son los resultados respectivamente:

Prueba nº	1	2	3	4	5
Máx	6,12M	13,06M	2,44M	7,209M	4,86M
Min	18,75k	291,2k	269,120k	193,4k	187,76k

Tabla 7 Datos de comportamiento normal para persona 23 años.

Prueba nº	1	2	3	4	5
Máx	1,411M	1.94M	2.51M	1.75M	1.917M
Min	462.4k	558.1k	272.57k	367.68k	415.2k

Tabla 8 Datos de comportamiento normal para persona 67 años.

Las pruebas consistían en actividades cotidianas, como andar de una habitación de la casa a otra, leer un libro, coger el ordenador, dar un manotazo a una mosca, abrir el frigorífico, etc

Tras hacer las medias de los valores máximos y mínimos y añadiendo cierto margen de seguridad, se ha decidido poner los umbrales superior e inferior de comportamiento normal en 2M la cota superior y 500k la cota inferior. Estos valores son generales y válidos para distintos tipos de usuario, de todas formas son ajustables en el código.

Para las cotas de caída libre e impacto sencillamente se cogieron los datos de las pruebas de caídas y se sacó lo

siguiente:

	Mínimo	Máximo	Media
Caída Libre	384	9.5k	2.51k
Impacto	15M	24M	22M

Tabla 9 Valores medios, máx. y mín. sacados de las pruebas.

Añadiendo los márgenes de seguridad pertinentes, se ha decidido definir la zona de caída libre como un valor de acel entre 0 y 40k , y la zona de impacto como un valor mayor de 10M (no es relevante como de grande es el impacto una vez pasada esa zona).

Finalmente la gráfica de caída por etapas quedaría definida de la siguiente manera:

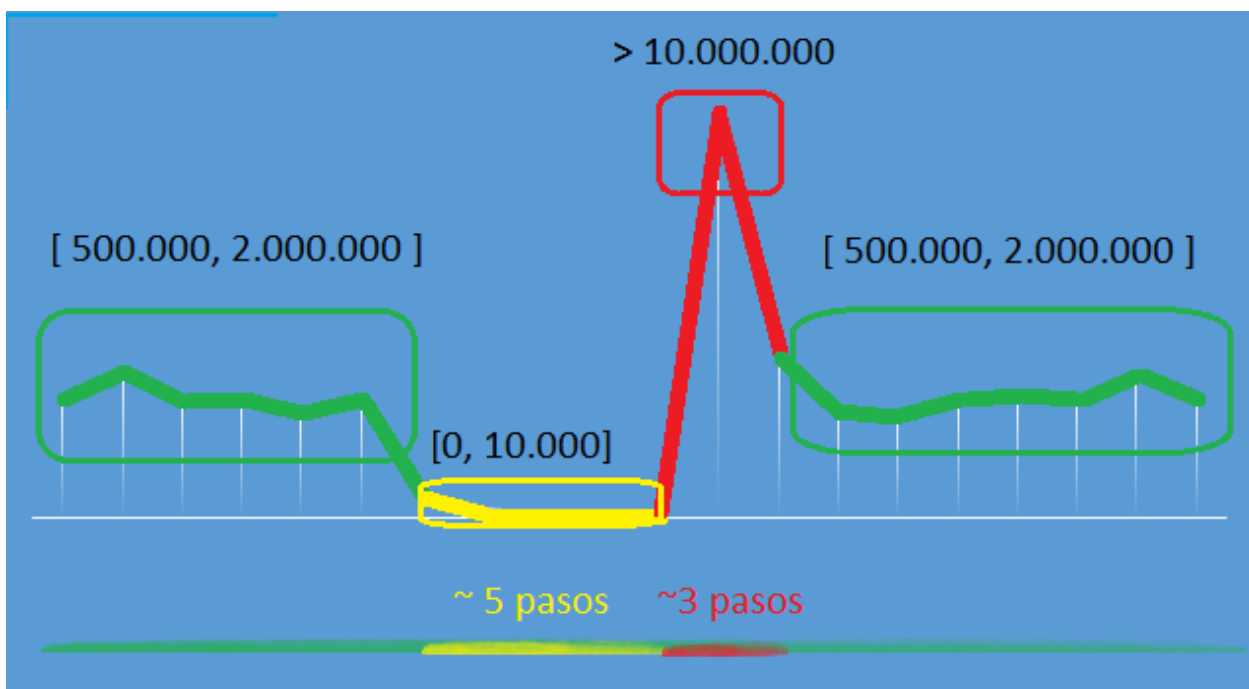


Fig 19 Gráfica de estados acotados.

2.3. Algoritmo de reconocimiento de caídas.

Este algoritmo tiene que discernir si pasamos por los 4 estados conocidos, con los valores que le llegan de acel cada 0.1 segundos. Debido a las limitaciones técnicas del reloj no podemos dejar de atender a los datos del acelerómetro (no hay ejecución en paralelo). Cada vez que llega un dato, en cada paso, la función de manejo de datos es llamada. Por tanto, la solución a la que se ha llegado es optar por un proceso de criba y ver si vamos avanzando al siguiente estado o no mediante variables bandera o flags. Es decir, dentro de la misma función gestionamos los datos del sensor y aplicamos el algoritmo.

Se ha añadido un estado intermedio entre comportamiento normal y caída libre, que es posible caída, en caso de que acel baje de 200.000, sólo con el fin de estar preparado si se acaba produciendo la caída.

El funcionamiento básico es el siguiente, en cada iteración observo si sobrepaso algún umbral (superior o inferior, depende del estado), si lo paso entonces veo si para las próximas iteraciones estoy dentro de la siguiente etapa y así sucesivamente.

Por ejemplo, si estamos en comportamiento normal (que es donde suponemos que vamos a estar siempre) y acel

pasa a valer 60.000, entonces veo si para las próximas 4 o 5 iteraciones a_{cel} está dentro de las cotas que definen la etapa de caída libre, si resulta que sí, pues sigo viendo si luego pasamos a impacto y posteriormente volvemos a comportamiento normal, en cuyo caso se habría detectado una caída, en caso contrario reseteo todos los flags y sigo analizando los datos actuales.

Es un algoritmo sencillo una vez que se conocen las etapas de la caída, pero cuesta ajustarlo correctamente para que no de demasiados falsos positivos y para que reconozca las caídas satisfactoriamente. Se incluye una versión en diagrama de flujo para su mejor comprensión:

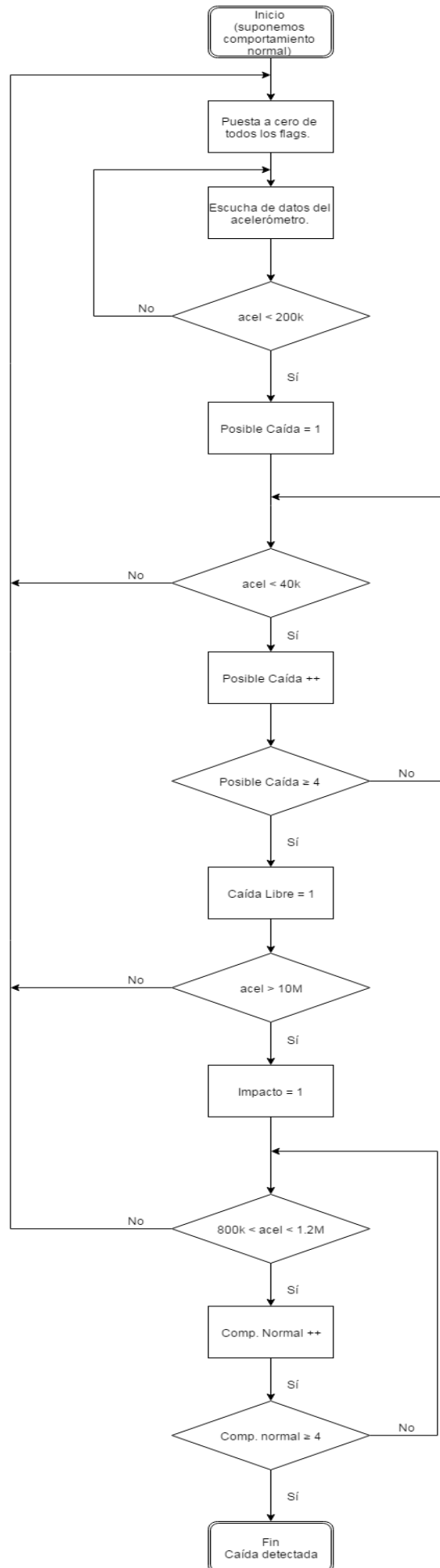


Fig. 20 Diagrama de flujo del algoritmo.

Los umbrales de este algoritmo son ajustables en función de la actividad y el movimiento del usuario.

2.4. Comunicación con el dispositivo Android.

Una vez reconocida una caída con éxito, el reloj muestra por pantalla una cuenta atrás de 15 segundos, en los últimos 5 segundos se añade también vibración para llamar la atención del usuario. Si se pulsa cualquier botón se vuelve al funcionamiento normal, en caso de no pulsarse el reloj, conectado vía Bluetooth con el teléfono, manda un mensaje de caída.

El envío de mensajes se hace con un sistema definido en el entorno de programación de Pebble, consiste en crear un diccionario (compartido por el reloj y teléfono) en el que se introducen “tuplas” o pares de valores. Una tupla consiste de una etiqueta (numérica) que define qué tipo de dato es, y un valor, para cuantificar ese dato. Por ejemplo, puedo tener el siguiente par de valores [2, 24], donde 2 es, según un diccionario que he definido arbitrariamente, un valor de temperatura, y 24 es el valor, en grados Celsius.

El diccionario a usar se define arbitrariamente, pero debe ser el mismo para ambas plataformas para poder entenderse correctamente.

En este proyecto no hay que enviar más que un tipo de dato, el dato de aviso de caída, por tanto siempre vamos a utilizar el valor 0 para definir nuestro mensaje. Tampoco recibimos nada del teléfono por medio del sistema de mensajes, la comunicación es unidireccional (en cuanto a mensajes sólo, porque para saber si hay conexión bluetooth tiene que haber cierta reciprocidad, pero para ello no necesitamos el sistema de mensajes).

Nuestro par de valores de aviso es [0,0].

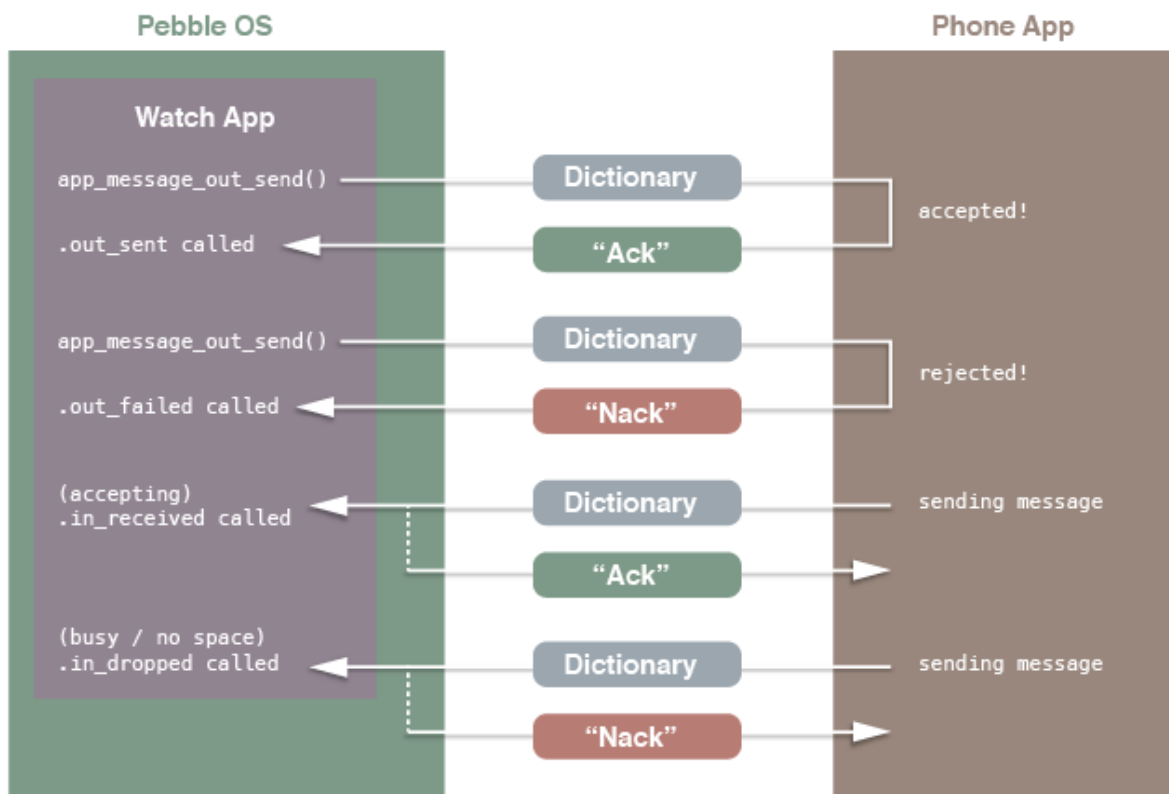


Fig. 21 Sistema de comunicación bidireccional

En nuestro código no vamos a necesitar un sistema bidireccional y los detalles de la comunicación están explicados en el código comentando.

2.5. El reloj en funcionamiento normal

Durante el funcionamiento rutinario del reloj, este está comprobando los datos de aceleración, la conexión bluetooth y mientras muestra la hora como se ve en la siguiente imagen:



Fig. 22 Captura de pantalla del funcionamiento normal (en el emulador)

Los valores que se observan en la esquina inferior izquierda son las variables bandera o flag para ver si se detecta una caída o no (por orden corresponden a Caída libre, Impacto y Comportamiento Normal). Se han decidido dejar en el código ya que este proyecto tiene una finalidad académica, y con esos 3 valores se puede comprobar como de bien funciona el algoritmo.

Si se pulsa el botón izquierdo se sale de la aplicación (es el botón de “atrás” del reloj), desafortunadamente los desarrolladores no pueden cambiar esto. Si se pulsa cualquiera de los otros tres botones no sucede nada, a no ser que se deje pulsado cualquiera de ellos durante más de 0.7 segundos, en cuyo caso se activa la ventana de cuenta atrás. Se ha añadido esta funcionalidad a modo de “botón de pánico”.

2.6. Problemas encontrados

Aparte del problema relativo al envío de mensajes que ya se ha comentado (la API de envío de mensajes falla y tiende a fallar mientras más mensajes se envíen, se recomienda por parte de los desarrolladores de Pebble no utilizarla para enviar muchos datos) se han encontrado otros problemas:

La primera generación de relojes Pebble tiene un fallo de software (reconocido por los propios desarrolladores de la empresa, código de error “0xfe504502”) que inhabilita el dispositivo por completo. Para solucionarlo hay que esperar que la batería se descargue completamente, poner el reloj en modo recuperación y restaurarlo a valores de fábrica (por tanto se pierden todos los datos y programas que se tenían). Durante la realización del proyecto me encontré con este error y fui capaz de solucionarlo sin tener que cambiar el reloj. Según la empresa esto ocurre aproximadamente al 3% de los dispositivos de la primera generación. Este error complica mucho el futuro del reloj para una aplicación comercial de este proyecto.

Ya se ha comentado, pero al no tener SDK propio en Windows, ha resultado más difícil depurar el código con el compilador online.

3 DISPOSITIVO ANDROID

Para la realización del proyecto se ha escogido el teléfono Huawei Y330, con versión Android 4.2. En este capítulo se detalla por qué se ha escogido este modelo y cual es su función en el proyecto.



Fig. 23 Huawei Y330

3.1 Elección del teléfono

3.1.1 Hardware

Se escogió este teléfono por ser asequible en cuanto precio y ser representativo de los teléfonos de gama baja de Android. Si el proyecto funciona sin problemas en los terminales de menor calidad, no debe tener ningún impedimento en el resto. El teléfono fue comprado por el alumno por un precio cercano a los 60 euros. Es un teléfono asequible. Sólo hay un inconveniente: el teléfono no está liberado y por tanto no puede hacer llamadas ni usar las redes móviles para acceder a internet, esto no es un requisito necesario pero más adelante se verá que

podría haber sido algo beneficioso.

Disponía de Bluetooth y conexión WiFi, con lo cual cumplía los requerimientos del proyecto.

3.1.2 Software

Para funcionar correctamente con la plataforma Pebble, se requiere como mínimo la versión 4.0 de Android. El modelo escogido cumple este requisito.

3.1.3 Programación en Android

Android es un lenguaje en alza en la actualidad y junto con la plataforma de Apple, se dividen el mercado global de teléfonos inteligentes. Se basa en el lenguaje de programación Java, también muy usado a día de hoy.

Como no sabía programar en Java decidí seguir las recomendaciones de mi tutor del proyecto y busqué un libro en el catálogo online de la universidad. Como Android es un lenguaje puntero y con una gran comunidad detrás, encontré un libro que enseñaba a programar en Java y después a aplicarlo a Android: “Android How To Program, second edition”

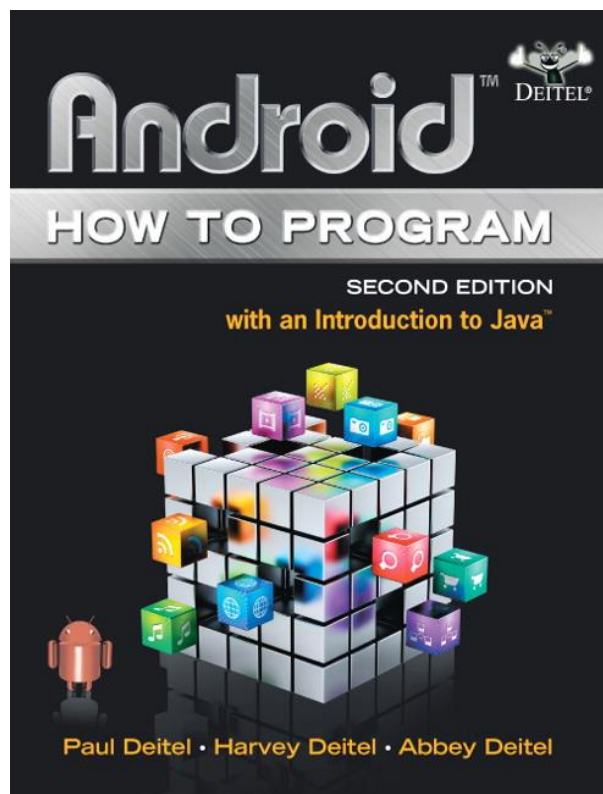


Fig. 24 Libro seguido por el alumno

Gracias a este libro me familiaricé con Java e hice algunos programas de ejemplo (sugeridos en el libro), hasta sentirme cómodo con el lenguaje. Dado que es un libro un poco antiguo, no lo seguí para aprender Android, ya que la herramienta de desarrollo ha cambiado mucho con los años (antes se usaba un programa llamado Eclipse, ahora se usa Android Studio). Para aprender Android me basé en la página oficial de google (www.developer.android.com) para desarrolladores y en diversos foros de internet.

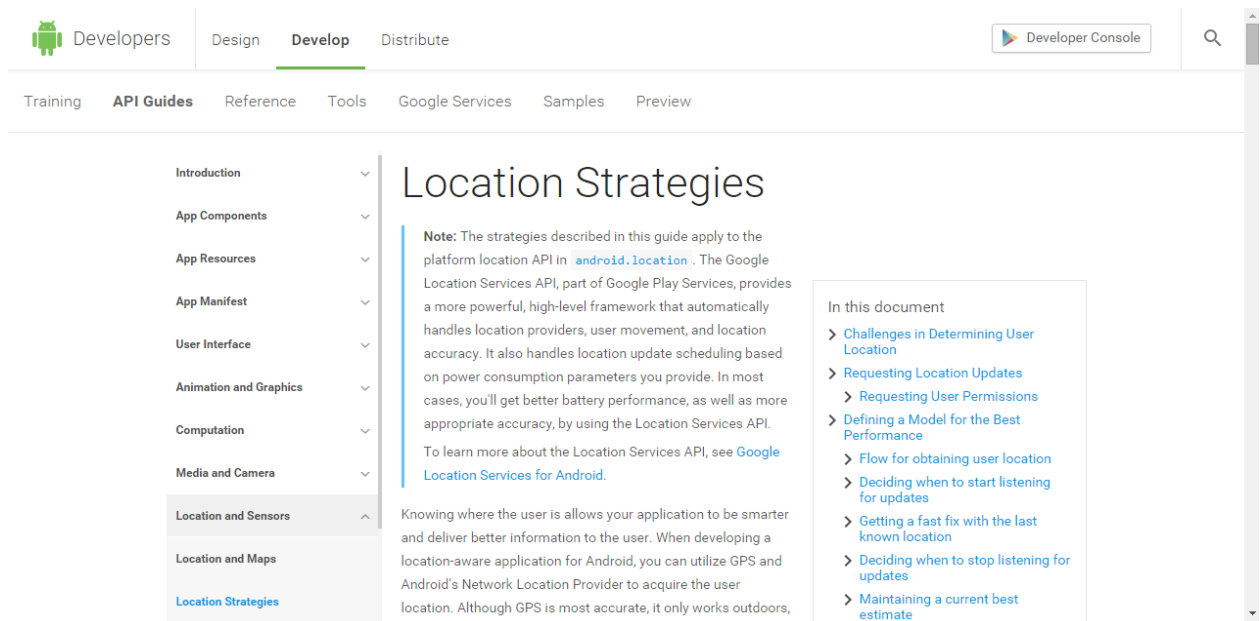


Fig. 25 Página de desarrolladores de Google.

La página oficial tiene muchas guías y tutoriales y fue más que suficiente para resolver las dudas que fueron surgiendo. Para programar se usó el programa Android Studio 1.2.1.1.

3.2 Tareas a realizar por el teléfono.

3.2.1 Ubicación

Para el correcto funcionamiento del proyecto es necesario que el teléfono disponga de los datos de geolocalización más recientes. Para ello se han usado las herramientas de localización que nos da Android que son distintas funciones tipo manejador para gestionar dichos datos. La aplicación obtiene las coordenadas de latitud y longitud del teléfono, con un margen de error de aproximadamente 10 metros (según las pruebas que se han hecho en localizaciones conocidas). Para acceder a la ubicación es necesario que el dispositivo esté conectado a Internet y tenga la localización GPS activada (para mayor precisión). Como se dijo antes, no se dispone de redes móviles, y estas son mucho más rápidas que la conexión WiFi a la hora de determinar una posición, lo que lleva a que algunas veces la aplicación tarde varios minutos en tener la ubicación.

3.2.2 Conexión con Pebble

Debe estar conectado el Bluetooth, cuando la conexión es correcta el sistema Android lo notificación en la barra superior, además el reloj tiene un mensaje que indica si hay algún problema en este aspecto.

Como ya se explicó, el envío de mensajes en este proyecto es unidireccional, del reloj al teléfono. El teléfono usa el diccionario que definimos anteriormente, sólo contiene el par de valores [0,0], que significa mensaje de caída. Cuando llega este mensaje se muestra por pantalla (del teléfono) un mensaje que lee “caída detectada, enviando mail de aviso”.

En caso de que se quiera enviar el correo electrónico directamente sin tener que pasar por el reloj, también se incluye un botón en la pantalla para enviar. Se usó en un principio para hacer pruebas y se ha dejado añadido como funcionalidad.

3.2.3 Envío automático de correo electrónico

En Android es bastante sencillo enviar un correo electrónico, podemos escribir nosotros el cuerpo, destinatario,

asunto, etc previamente o no. Se hace a través de una estructura conocida como “intent” o intención, que se puede llamar cuando convenga (al pulsar un botón, cuando llega la conexión con el reloj, etc). Sin embargo, este sistema tiene una gran tara para nuestro proyecto: Necesita obligatoriamente intervención del usuario. Incluso si definimos nosotros el correo, al ejecutarse el intent aparecerá una ventana de diálogo en el teléfono en la que nos dice que debemos escoger el cliente de correo que queremos usar (Gmail, aplicación de e-mails nativa del teléfono u otra). Por tanto necesita que el usuario pulse la pantalla, lo cual no es aceptable en un proyecto de estas características.

Para sortear este problema se ha hecho uso de una librería Java llamada Javamail. Una librería en Java se puede entender como un conjunto de funciones (clases, métodos, etc) que ya han sido programados y están a nuestra disposición. Por ejemplo, para comunicar el reloj con el teléfono, hay que añadir la librería de Pebble. Es una forma estructurada de reutilizar código.

La instalación de la librería de Javamail (hay que añadir 3 archivos “.jar” además de la librería) y su correcto funcionamiento ha sido uno de los escollos más grandes de todo el proyecto. Esto es debido a que Android Studio es un programa un poco lento y para ciertas cosas, poco amigable para el usuario (no es intuitivo añadir librerías) y llegado a este punto el archivo del proyecto tenía muchas líneas de código y subarchivos. Además hay que añadir permisos extra para utilizar el GPS, internet o los correos electrónicos.

Además tampoco hay tutoriales en esta materia en las páginas de desarrolladores de Google, lo cual tiene cierta lógica, se facilita que se pueda enviar un email de manera manual, pero se oculta cómo hacerlo de manera automática o programática, ya que esta funcionalidad se podría utilizar para mandar correos de forma masiva a espaldas del usuario y favorecer la proliferación de correos basura. Para poder enviar correos desde una cuenta de gmail (he usado la mía personal tomasacostaalmeda@gmail.com) hay que autorizarlo específicamente en los ajustes.

En los foros de internet también cuesta encontrar la solución a este problema, además la solución ha ido cambiando conforme la herramienta de desarrollo ha ido evolucionando, con lo cual, una solución de hace por ejemplo 2 o 3 años puede no ser válida hoy en día.

Pero tras una investigación extensa y una detallada búsqueda, así como de muchos ensayos y errores, se consiguió enviar correos de forma automática. El propio correo se puede definir en el código (cuerpo, destinatario, asunto, etc). El correo tiene además un enlace a google maps donde usamos las coordenadas de la localización para mostrar por pantalla la situación exacta de la caída:

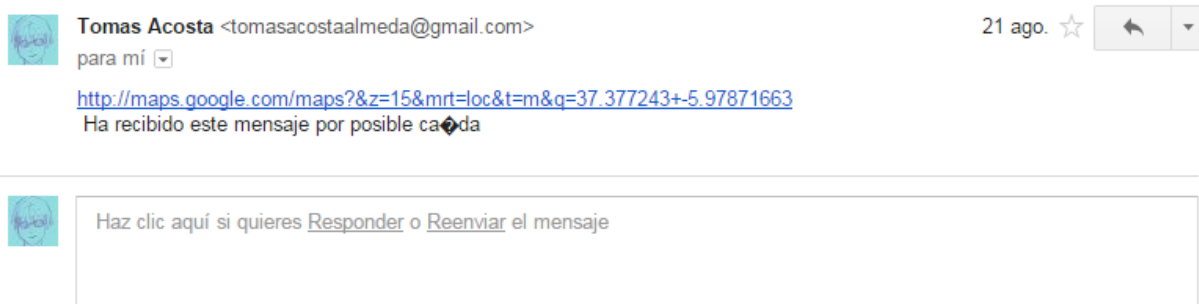


Fig. 26 Apariencia del correo

Si se pincha en el enlace :

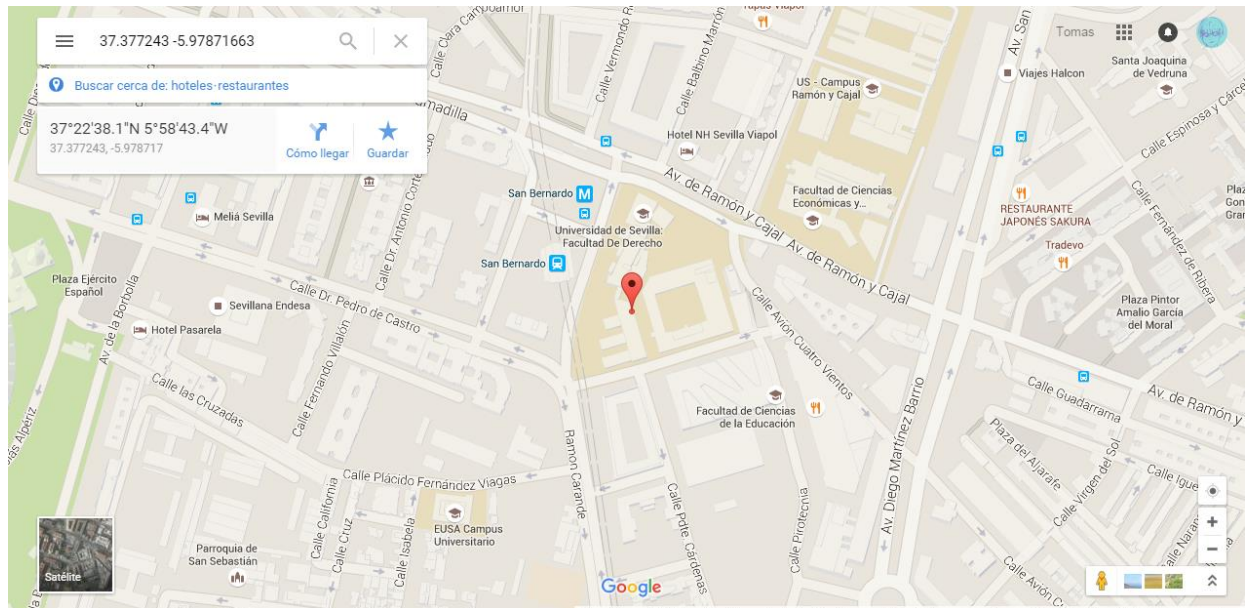


Fig. 27 Enlace de Google Maps

4 CONCLUSIONES Y POSIBLES MEJORAS

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Claude Shannon, 1948

Tras terminar una versión funcional del proyecto se estuvo probando su eficacia. Se ajustaron los valores límites o umbrales del algoritmo de reconocimiento para minimizar los falsos positivos. Se ha dado prioridad a reconocimiento de caídas. Es decir se prefiere un falso positivo antes que una caída no reconocida. Se proponen una serie de mejoras posibles para el proyecto:

Utilizar un teléfono liberado y con acceso a las redes móviles para un reconocimiento de la ubicación mucho más rápido.

Usar un reloj programable con un procesador más rápido, lo que permitiría un tiempo de muestreo muchísimo más pequeño, lo que llevaría a una detección más precisa. Las versiones más nuevas del sistema Pebble tienen un procesador mejor.

5 ANEXOS

Se incluyen en los siguientes anexos el código del proyecto. Está comentado para facilitar la comprensión del lector.

A Código en Pebble

A.1 Pantalla principal "fd2.c"

```

/*
 * main.c
 * Constructs a Window housing an output TextLayer to show data from
 * either modes of operation of the accelerometer.
 */

#include <pebble.h>
#include "window2.c" // La ventana 2 es una cuenta atrás

#define TAP_NOT_DATA false

#define KEY 0 // Claves de conexión reloj -> teléfono ( tupla )
#define CAIDA 0

// Declaración de variables a usar en el código

static int acel; // Valor absoluto del vector aceleración
static int x,y,z; // Variable donde voy a guardar los valores en los 3 ejes del
acelerómetro
static int cont=0; // Contador para el data handler
static int i=0; // índice para el upclick handler ( no se usa)
static int sum=0; // unused
static int max=0; // Variables auxiliares para establecer los máximos y mínimos de
acel.
static int min=100000000;
static int index;
static int PC=0; //PC = Posible caída
static int CaLib=0; // Caída Libre , flag.
static int impacto=0; // flag
static int normal=0; //flag de comportamiento normal tras caída e impacto.
static int CN=0; // Comportamiento Normal
static int aux=0; //índice auxiliar.

```



```

text_layer_set_text(s_output_layer2, char_buffer2);
i=i+1;
}

static void down_click_handler(ClickRecognizerRef recognizer, void *context) { //
MANEJADOR DE CLICK (no se usa)
    i=i-1;

    snprintf(char_buffer, sizeof(char_buffer), "Cont: %d\n %d", i, int_buffer[i]);
    text_layer_set_text(s_output_layer, char_buffer);
    snprintf(char_buffer2, sizeof(char_buffer2), "Max: %d\n Min: %d", max, min);
    text_layer_set_text(s_output_layer2, char_buffer2);
}*/

void up_long_click_handler(ClickRecognizerRef recognizer, void *context) {

    // Window *window = (Window *)context;
}

void up_long_click_release_handler(ClickRecognizerRef recognizer, void *context) {

    //Window *window = (Window *)context;
    window_stack_push(window2, true/*animated*/); // Paso a la ventana de la cuenta
atrás.
}

void down_long_click_handler(ClickRecognizerRef recognizer, void *context) {

    // Window *window = (Window *)context;
}

void down_long_click_release_handler(ClickRecognizerRef recognizer, void *context)
{

    //Window *window = (Window *)context;
    window_stack_push(window2, true/*animated*/); // Paso a la ventana de la cuenta
atrás.
}

void select_long_click_handler(ClickRecognizerRef recognizer, void *context) {

    // Window *window = (Window *)context;

```

```
}

void select_long_click_release_handler(ClickRecognizerRef recognizer, void
*context) {

    //Window *window = (Window *)context;
    window_stack_push(window2, true/*animated*/); // Paso a la ventana de la cuenta
    atrás.
}

static void click_config_provider(void *context) {
    // Register the ClickHandlers
    window_long_click_subscribe(BUTTON_ID_UP, 700, up_long_click_handler,
up_long_click_release_handler);
    window_long_click_subscribe(BUTTON_ID_DOWN, 700, down_long_click_handler,
down_long_click_release_handler);
    window_long_click_subscribe(BUTTON_ID_SELECT, 700, select_long_click_handler,
select_long_click_release_handler);
}

// MANEJADOR DE DATOS DE ACELEROMETRO
// Se encarga de recibir los datos del acelerómetro cada ciclo y de procesarlos.
Debido a la complejidad de la
// programación concurrente en el sistema Pebble ( hay que recordar que tiene un
procesador bastante limitado)
// el mismo manejador de datos es el que los procesa y aplica el algoritmo de
selección de caídas.

static void data_handler(AccelData *data, uint32_t num_samples) { // data_handler
está ya definido, es así
    // Definiciones : Guardamos cada componente en una de las variables ( estas
variables son globales ).
    x=data[0].x;
    y=data[0].y;
    z=data[0].z;

    acel=x*x+y*y+z*z; // acel es el módulo de la aceleración.
    int_buffer[cont]=acel; // Guardo dicho módulo en un vector, este paso es
importante para ir viendo la evolución de
        // los valores del acelerómetro.

    // Mostrar por pantalla los datos : Esta parte se ha comentado puesto que sólo
servía en el periodo de investigación
```

```

// Básicamente mostraba por pantalla los valores de x, y, z, un contador de los
ciclos del acelerómetro y el valor de
// acel. Se usaba para ver en tiempo real qué iba marcando el sensor. Mediante
los botones se podía ver paso por paso
// los valores de cada ciclo, para estudiar con mayor detenimiento qué estaba
pasando.

/*snprintf(char_buffer, sizeof(char_buffer),
           "Cont x y z\n %d %d %d %d",
           cont,x,y,z
           );
text_layer_set_text(s_output_layer,char_buffer);

snprintf(char_buffer2,sizeof(char_buffer2),"  %d",acel);
text_layer_set_text(s_output_layer2,char_buffer2);*/

// Almacenamos los valores máximo y mínimo del vector de acel.
if(int_buffer[cont]>max){
    max=int_buffer[cont];
}
if(int_buffer[cont]<min){
    min=int_buffer[cont];
}

// Algoritmo de selección de caídas: Deben darse 4 condiciones consecutivas para
determinar caída
// Comprtamiento normal -> Caída libre -> Impacto -> Comportamiento normal
// El funcionamiento y la base de este algoritmo deben estar descritos con
detalle en otras partes de este proyecto.
// Los valores umbrales de cada paso están sacados de muchas y diversas pruebas
con el acelerómetro y
// pueden ajustarse según la sensibilidad que se quiera ( cada persona tiene una
movilidad distinta ).

// CAIDA LIBRE
// Se tiene que dar un valor pequeño de acel, seguido de al menos 3 ciclos o
iteraciones con valores muy pequeños
// Cuando se supera el límite inferior de 200k lo que hacemos es observar si se
trata de una posible caída o
// sencillamente es un valor pequeño.
if((acel<200000)){
    if(PC==0){PC=1;} // Una iteración menor de 200k -> Empezamos a ver si estamos
en una posible caída o no.
    if(acel<40000){ // Al menos 3 iters menores que 40k.
        PC=PC+1;
    }
}

```



```
    if(PC==4){ //Se confirma caída libre.
        CaLib=1; // Activamos el flag de caída libre.
    }
}
else{PC=0;} // Si se trataba sencillamente de un valor bajo de acel, reseteamos
el flag de posible caída y volvemos
// a funcionamiento normal.

// IMPACTO
if(CaLib==1){ // Si estoy en caída libre confirmada, espero al IMPACTO
    if(accel>10000000){impacto=1;} // Valor de impacto medio, sacado de las pruebas
con el reloj. Es ajustable.
}
// COMPORTAMIENTO NORMAL (post caída e impacto)
// Lo que hacemos es ver si en las próximos 4 ciclos estamos entre los valores
límites de comportamiento normal.

if(impacto==1){
    aux=aux+1;
    if(aux>4){
        if(accel<1200000){
            if(accel>800000){
                normal=normal+1;
            }
            else{normal=0;}
        }
        else{normal=0;}
    }
    if(normal>4){
        // Una vez detectada la caída reseteamos todos los flags y mandamos la
        ventana de la cuenta atrás (window2).
        CN=1;
        //accel_data_service_unsubscribe();
        PC=0;
        CaLib=0;
        impacto=0;
        normal=0;
        CN=0;
        aux=0;
        window_stack_push(window2, true/*animated*/); // Paso a la ventana de la
        cuenta atrás.
    }
}

// Mostrar por pantalla el estado actual:
// Muestra el valor de 3 de los flags del algoritmo para saber en qué estado
estamos

snprintf(char_buffer1,sizeof(char_buffer1),"CL: %d\n Impacto: %d\n CN:
%d",CaLib,impacto,CN);
text_layer_set_text(s_output_layer3,char_buffer1);
```

```

// Actualizar contador:
cont=cont+1;

/*if(cont>400){ // Esto se usaba para recopilar información, ahora no sirve
para nada.
    cont=0;
    accel_data_service_unsubscribe();
}*/
}

static void tap_handler(AccelAxisType axis, int32_t direction) { // No se usa pero
hay que ponerlo.

}

static void main_window_load(Window *window) {
    Layer *window_layer = window_get_root_layer(window);
    GRect window_bounds = layer_get_bounds(window_layer);

    // Los 3 siguientes párrafos se encargaban de la creación de 3 capas de gráficos
y texto para ver los valores de
    // las componentes y el módulo de la aceleración. Se usaba en la investigación
del acelerómetro. Lo deajo comentado
    // por si puede resultar útil.

    s_output_layer = text_layer_create(GRect(5, 0, 200, 200));
    text_layer_set_font(s_output_layer, fonts_get_system_font(FONT_KEY_GOTHIC_24));
    //text_layer_set_text(s_output_layer, "No data yet.");
    //text_layer_set_overflow_mode(s_output_layer, GTextOverflowModeWordWrap);
    layer_add_child(window_layer, text_layer_get_layer(s_output_layer));

    s_output_layer2 = text_layer_create(GRect(5, 60, window_bounds.size.w - 10,
window_bounds.size.h));
    text_layer_set_font(s_output_layer2, fonts_get_system_font(FONT_KEY_GOTHIC_24));
    //text_layer_set_text(s_output_layer2, "Probando .");
    //text_layer_set_overflow_mode(s_output_layer2, GTextOverflowModeWordWrap);
    layer_add_child(window_layer, text_layer_get_layer(s_output_layer2));

    s_output_layer3 = text_layer_create(GRect(5, 100, window_bounds.size.w - 10,
window_bounds.size.h));
    text_layer_set_font(s_output_layer3, fonts_get_system_font(FONT_KEY_GOTHIC_14));
    //text_layer_set_text(s_output_layer3, "Probando .");
    //text_layer_set_overflow_mode(s_output_layer3, GTextOverflowModeWordWrap);
    layer_add_child(window_layer, text_layer_get_layer(s_output_layer3));

```

```
// Capa de texto para el bluetooth
s_output_layer1 = text_layer_create(GRect(0, 0, 150, 25));
text_layer_set_text_alignment(s_output_layer1, GTextAlignmentCenter);
layer_add_child(window_layer, text_layer_get_layer(s_output_layer1));

// Manejador de bluetooth que nos enseña el estado actual
bt_handler(bluetooth_connection_service_peek());

// Capa de texto para el reloj
s_time_layer = text_layer_create(GRect(0, 55, 144, 50));
text_layer_set_background_color(s_time_layer, GColorClear);
text_layer_set_text_color(s_time_layer, GColorBlack);
text_layer_set_text(s_time_layer, "00:00");
text_layer_set_font(s_time_layer,
fonts_get_system_font(FONT_KEY_BITHAM_42_BOLD));
text_layer_set_text_alignment(s_time_layer, GTextAlignmentCenter);
layer_add_child(window_get_root_layer(window),
text_layer_get_layer(s_time_layer));

// Llamada al reloj, lo ponemos en main_window_load para que el tiempo aparezca
desde el principio.
update_time();
}

static void main_window_unload(Window *window) {
// Destruir la capa de texto.
text_layer_destroy(s_output_layer);
}

// Manejador de "ticks", llama a nuestro propio manejador de tiempo.
static void tick_handler(struct tm *tick_time, TimeUnits units_changed) {
update_time();
}

static void init() {
// Hay que registrar todas las llamadas a funciones:
app_message_register_inbox_received(inbox_received_handler);
app_message_register_inbox_dropped(inbox_dropped_handler);
app_message_register_outbox_failed(outbox_failed_handler);
app_message_register_outbox_sent(outbox_sent_handler);

// Abrir AppMessage
app_message_open(app_message_inbox_size_maximum(),
app_message_outbox_size_maximum());
```

```

// Creación de main_window
// init() y deinit() tienen siempre esta estructura, que está dada por el sistema
pebble. A la hora de programar
// en esta plataforma es mejor crearse una plantilla con estas funciones, así
como main window load y unload.

s_main_window = window_create();
window_set_window_handlers(s_main_window, (WindowHandlers) {
    .load = main_window_load,
    .unload = main_window_unload
});

// Create window2, no significa que aparezca, solo asigno handlers y demás
window2 = window_create();
window_set_click_config_provider(window2, click_config_provider2);
window_set_window_handlers(window2, (WindowHandlers) {
    .load = window_load2,
    .appear=window_appear2,
    .disappear=window_disappear2,
    .unload = window_unload2,
});

window_stack_push(s_main_window, true);

// Elección del servicio de acelerómetro que quiero usar, nosotros usamos el data
service, que nos va dando los valores
// en los 3 ejes cada ciclo. El tap service te da una respuesta cada vez que se
agita el reloj.
if (TAP_NOT_DATA) {
    // Tap service
    accel_tap_service_subscribe(tap_handler);
} else {
    // Data service
    int num_samples = 1; // Sólo quiero un sample por vez
    accel_data_service_subscribe(num_samples, data_handler);

    window_set_click_config_provider(s_main_window, click_config_provider);

    // Frecuencia de muestreo. Todas las pruebas se han hecho con esta frecuencia y
    el algoritmo está basado en esta
    // frecuencia, por tanto no se debería cambiar nunca de 10Hz.
    accel_service_set_sampling_rate(ACCEL_SAMPLING_10HZ);

    // Suscribirse al servicio de bluetooth
    bluetooth_connection_service_subscribe(bt_handler);

    // Suscribirse al servicio de tiempo.

```

```
    tick_timer_service_subscribe(MINUTE_UNIT, tick_handler);
}
}

static void deinit() {
    // Destruir ventana principal.
    window_destroy(s_main_window);

    if (TAP_NOT_DATA) {
        accel_tap_service_unsubscribe();
    } else {
        accel_data_service_unsubscribe();
    }
}

int main(void) {
    init();
    app_event_loop();
    deinit();
}
```

A.2 Pantalla de cuenta atrás "window2.c"

```
#include
<pebble.h>

static int count_down=15; // Los segundos de espera ante falso positivo,
pasado este tiempo se envía la señal de socorro

static TextLayer *text_layer, *note_layer, *note_layer2; // Las distintas
capas de texto

// En esta ventana sí usamos los botones, ya que ante cualquier pulsación se
debe anular la cuenta atrás
static void select_click_handler2(ClickRecognizerRef recognizer, void
*context);
static void up_click_handler2(ClickRecognizerRef recognizer, void *context);
static void down_click_handler2(ClickRecognizerRef recognizer, void *context);
static void count_down_handler(struct tm *tick_time, TimeUnits units_changed);
```

```

static void window_load2(Window *window2) {

    // PARTE GRÁFICA DE LA VENTANA 2:
    // La capa de texto que va a ir mostrando los segundos que quedan.
    Layer *window_layer = window_get_root_layer(window2);
    GRect bounds = layer_get_bounds(window_layer);
    //text_layer = text_layer_create((GRect) { .origin = { 0, 72 } });
    text_layer = text_layer_create((GRect) { { 5, 0 }, { bounds.size.w - 2*5,
bounds.size.h } });
    text_layer_set_text_alignment(text_layer, GTextAlignmentCenter);
    text_layer_set_font(text_layer,
fonts_get_system_font(FONT_KEY_ROBOTO_BOLD_SUBSET_49));
    layer_add_child(window_layer, text_layer_get_layer(text_layer));
    text_layer_set_text(text_layer, "15");

    // Capa de texto para mostrar el mensaje de "Pulsa un boton"
    note_layer = text_layer_create((GRect) { .origin = { 0, 72 }, .size = {
bounds.size.w, 28 } });
    text_layer_set_font(note_layer,
fonts_get_system_font(FONT_KEY_ROBOTO_CONDENSED_21));
    text_layer_set_text(note_layer, "Pulsa un boton");
    text_layer_set_text_alignment(note_layer, GTextAlignmentCenter);
    layer_add_child(window_layer, text_layer_get_layer(note_layer));

    // Capa de texto para mostrar el mensaje de "para parar"
    note_layer2 = text_layer_create((GRect) { .origin = { 0, 100 }, .size = {
bounds.size.w, 28 } });
    text_layer_set_font(note_layer2,
fonts_get_system_font(FONT_KEY_ROBOTO_CONDENSED_21));
    text_layer_set_text(note_layer2, "para parar");
    text_layer_set_text_alignment(note_layer2, GTextAlignmentCenter);
    layer_add_child(window_layer, text_layer_get_layer(note_layer2));

}

// Manejador de ticks, es una función que se dispara cada vez que pasa un
segundo, es la que usamos para ir actualizando
// la cuenta atrás.
static void count_down_handler(struct tm *tick_time, TimeUnits units_changed){
    static char buf[] = "12";
    snprintf(buf, sizeof(buf), "%d", count_down--);
    text_layer_set_text(text_layer, buf);
}

```

```
    if(count_down<5){          // Si la cuenta atrás llega a los 5 segundos, se
    aumenta la presión sobre el usuario al
        vibes_short_pulse(); // vibrar por cada segundo.
    }
    if(count_down==0){        // Si pasan los 15 segundos sin intervención del
    usuario, se envía el mensaje.
        //ENVÍO DEL MENSAJE
        // Primero se define un diccionario (con el que me comunico con el
    teléfono), se le asigna el valor de la tupla
        // correspondiente y se envía. Para enviar primero abrimos la bandeja de
    salida y después enviamos.
        DictionaryIterator *iter;
        app_message_outbox_begin(&iter);
        Tuplelet value = TupleletInteger(0,0); // Como sólo hay un tipo de mensaje, la
    tupla no tiene mucho sentido.
        dict_write_tuplet(iter, &value);
        app_message_outbox_send();

        vibes_long_pulse(); // Vibración larga para señalar que se ha enviado el
    mensaje.
        window_stack_pop(true); // Volvemos a la ventana principal.
    }
}

static void window_appear2(Window *window2){
    tick_timer_service_subscribe(SECOND_UNIT, count_down_handler);
    count_down_handler( NULL, SECOND_UNIT);
    count_down=15;
    vibes_long_pulse();
}

static void window_disappear2(Window *window2){
    tick_timer_service_unsubscribe();
}

static void window_unload2(Window *window2) {
    text_layer_destroy(text_layer);
    text_layer_destroy(note_layer);
    text_layer_destroy(note_layer2);
}

static void click_config_provider2(void *context) {
    window_single_click_subscribe(BUTTON_ID_SELECT, select_click_handler2);
    window_single_click_subscribe(BUTTON_ID_UP, up_click_handler2);
    window_single_click_subscribe(BUTTON_ID_DOWN, down_click_handler2);
}
```

```
// MANEJADORES DE BOTONES
// En caso de que se pulse cualquiera de los 3 botones físicos del reloj, se
// finaliza la cuenta atrás y se vuelve
// a la pantalla principal.
static void down_click_handler2(ClickRecognizerRef recognizer, void *context){
    window_stack_pop(true);
}

static void up_click_handler2(ClickRecognizerRef recognizer, void *context) {
    window_stack_pop(true);
}

static void select_click_handler2(ClickRecognizerRef recognizer, void
*context){
    window_stack_pop(true);
}
```


B Código en Android

Se incluye el código del archivo principal, MainActivity.java.

B.1 MainActivity.java

```
package com.example.tomas.mail2;

// Listado de librerías y APIs que se han usado
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.AsyncTask;
import android.os.Handler;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.getpebble.android.kit.Constants;
import com.getpebble.android.kit.PebbleKit;
import com.getpebble.android.kit.util.PebbleDictionary;

import com.getpebble.android.kit.PebbleKit;

import java.util.UUID;

import java.io.UnsupportedEncodingException;
import java.util.Date;
import java.util.Properties;
import javax.activation.CommandMap;
import javax.activation.DataHandler;
```

```

import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.activation.MailcapCommandMap;
import javax.mail.BodyPart;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

// Como el objetivo de la aplicación es simple, se ha decidido incluir todas
// las funcionalidades
// dentro de una sola clase : MainActivity.
public class MainActivity extends Activity implements LocationListener {
    private static final String username = "tomasacostaalmeda@gmail.com"; //
Nombre de usuario del correo : usuario@gmail.com
    private static final String password = "..."; // Contraseña (obviamente
no se incluye en este código)

    private TextView text1;
    private LocationManager locationManager; // Manejador de localización
    private String provider;

    public double lon;
    public double lat;

    private static final UUID WATCHAPP_UUID = UUID.fromString("31abf20f-1ba1-
4225-907c-b66b28d8cf7e");
    // Necesito la identidad de la aplicación con la que me comunico, eso es
la UUID
    private static final String WATCHAPP_FILENAME = "ReceiveData.c"; // Y
este es el nombre de la aplicación.

    private static final int

        KEY = 0,
        CAIDA = 0,
        KEY_VIBRATE = 1;

    // Como ya se ha explicado se va a usar el par de valores [0,0] para las
comunicaciones

    private Handler handler = new Handler();
    private PebbleKit.PebbleDataReceiver appMessageReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

```

```

super.onCreate (savedInstanceState) ;
setContentView (R.layout.activity_main) ;

text1 = (TextView) findViewById (R.id.textView) ;

// LOCALIZACION
locationManager = (LocationManager)
getSystemService (Context.LOCATION_SERVICE) ;
provider = LocationManager.GPS_PROVIDER ;
text1.setText ("buscando localización...") ;
locationManager.requestLocationUpdates (provider, 400, 1, this) ;

//emailEdit = (EditText) findViewById (R.id.email) ;
//subjectEdit = (EditText) findViewById (R.id.subject) ;
//messageEdit = (EditText) findViewById (R.id.message) ;
Button sendButton = (Button) findViewById (R.id.Button) ;

sendButton.setOnClickListener (new View.OnClickListener () {
    @Override
    public void onClick (View view) {
        // Se implementa un botón en la pantalla para forzar un envío
de correo a modo de boton de panico.

        // INICIALIZAR MAIL @@@@@@@@@@@@@@@@@@@@@@
        // Esta es la forma que tendrá nuestro correo de aviso:

        String email = "tomasacostaalmeda@gmail.com";
        String subject = "Mensaje de aviso de caída";
        String longitude = String.valueOf (lon) ;
        String latitude = String.valueOf (lat) ;
        String gmaps =
"http://maps.google.com/maps?&z=15&mrt=loc&t=m&q=";
        String message = gmaps + latitude + "+" + longitude + "  \n
Ha recibido este mensaje por posible caída";
        /* Formato de envío de mails :
        este es el formato a seguir si se quiere que el enlace de
Google Maps muestre la posición
con un pin rojo.
http://maps.google.com/maps?&z=15&mrt=loc&t=m&q=37.37747828+-
5.97867533
        */

        sendMail (email, subject, message) ;
        //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    }
});
}

protected void onResume () {
super.onResume () ;
locationManager.requestLocationUpdates (provider, 400, 1, this) ;

// Comportamiento de AppMessage (el sistema de comunicación con el
reloj)
if (appMessageReciever == null) { // Si no existia antes, lo creo.

```



```

        };
    }
};

// Añadir las funcionalidades de AppMessage
PebbleKit.registerReceivedDataHandler(this, appMessageReciever);
}

@Override
protected void onPause() {
    super.onPause();
    locationManager.removeUpdates(this);
    // Eliminar suscripción a AppMessage
    if(appMessageReciever != null) {
        unregisterReceiver(appMessageReciever);
        appMessageReciever = null;
    }
}

@Override
public void onLocationChanged(Location location) {
    // Cuando cambie la localizacion vuelvo a obtener las coordenadas.
    text1.setText(location.getLatitude() + ", " +
location.getLongitude());
    lon=location.getLongitude();
    lat=location.getLatitude();
}

public void onStatusChanged(String provider, int status, Bundle extras) {
    text1.setText("onStatusChanged: " + status);
}

public void onProviderEnabled(String provider) {
    text1.setText("Enabled new provider " + provider);
}

public void onProviderDisabled(String provider) {
    text1.setText("Disabled provider " + provider);
}

private void sendMail(String email, String subject, String messageBody) {
    Session session = createSessionObject();

    try {
        Message message = createMessage(email, subject, messageBody,
session);
        new SendMailTask().execute(message);
    }

    catch (AddressException e) {
        e.printStackTrace();
    } catch (MessagingException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

private Message createMessage(String email, String subject, String

```

```

messageBody, Session session) throws MessagingException,
UnsupportedEncodingException {
    Message message = new MimeMessage(session);
    message.setFrom(new InternetAddress("tomasacostaalmeda@gmail.com",
"Tomas Acosta"));
    message.addRecipient(Message.RecipientType.TO, new
InternetAddress(email, email));
    message.setSubject(subject);
    message.setText(messageBody);
    return message;
}

private Session createSessionObject() {
    Properties properties = new Properties();
    properties.put("mail.smtp.auth", "true");
    properties.put("mail.smtp.starttls.enable", "true");
    properties.put("mail.smtp.host", "smtp.gmail.com");
    properties.put("mail.smtp.port", "587");

    return Session.getInstance(properties, new javax.mail.Authenticator()
{
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    });
}

private class SendMailTask extends AsyncTask<Message, Void, Void> {
    private ProgressDialog progressDialog;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        //progressDialog = ProgressDialog.show(MainActivity.this, "Please
wait", "Sending mail", true, false);
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        //progressDialog.dismiss();
    }

    @Override
    protected Void doInBackground(Message... messages) {
        try {
            Transport.send(messages[0]);
        } catch (MessagingException e) {
            e.printStackTrace();
        }
        return null;
    }
}
}

```

REFERENCIAS

[1] Página de desarrolladores de Pebble: <http://developer.getpebble.com/>

[2] Foros de desarrollo del sistema Pebble: <http://forums.getpebble.com/>

[3] Comunidad de programadores StackOverflow: <http://stackoverflow.com/>

[4] Android How To Program, Second edition, by Paul, Harvey and Andy Deitel: <http://0-proquest.safaribooksonline.com.fama.us.es/book/programming/android/9780133802092>

[5] Repositorios de código de la plataforma GitHub:

- <https://github.com/Hitheshaum/fall-detection>

- [https://github.com/xPret/Android-](https://github.com/xPret/Android-JavaMailSenderUtil/blob/master/src/com/ivanpretel/droid/utills/javamail/MainActivity.java)

[JavaMailSenderUtil/blob/master/src/com/ivanpretel/droid/utills/javamail/MainActivity.java](https://github.com/xPret/Android-JavaMailSenderUtil/blob/master/src/com/ivanpretel/droid/utills/javamail/MainActivity.java)

- <https://github.com/pebble-examples/pebblekit-android-example>

[6] Información sobre envío automático de Mails en Android:

- http://www.jondev.net/articles/Sending_Emails_without_User_Intervention_%28no_Intents%29_in_Android

- <http://www.tiemenschut.com/how-to-send-e-mail-directly-from-android-application/>

[7] Información de desarrollo de Pebble:

- <http://es.slideshare.net/pebbledev/pebble-development-app>

- <https://ninedof.wordpress.com/2013/07/11/pebble-watch-face-sdk-tutorial-6-2-way-communication-with-android/>

[8] Página de desarrolladores de Android: <http://developer.android.com/index.html>

ÍNDICE DE CONCEPTOS

GLOSARIO
