Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

MONITORIZACIÓN ECO-EFICIENTE Y PARALELA CON DISPOSITIVOS MÓVILES. MANUAL DEL PROGRAMADOR.

D. VILLALÓN FERNÁNDEZ, Alberto

TUTOR: D. RANILLA PASTOR, José

FECHA: Julio de 2020

Tabla de contenido

Índice de Tablas

Índice de Figuras

Manual del programador

1.1 Prototipo Android para monitorizar la actividad y frecuencia cardiaca

Una vez desarrollados los sistemas de monitorización de la frecuencia cardiaca y la actividad física, se desarrolla un prototipo Android para integrar ambos sistemas y proporcionar al usuario alertas sobre posibles anomalías detectadas en su frecuencia cardiaca. Además, se presenta al usuario información acerca de su frecuencia cardiaca en relación con la actividad física realizada.

El prototipo cuenta con dos partes diferenciadas para gestionar la información obtenida a través de la pulsera y la información que proviene del reloj inteligente. Se organiza utilizando un *TabLayout*, que es un componente que proporciona la capacidad para albergar diferentes pestañas. De este modo se utiliza una pestaña para gestionar la pulsera y otra pestaña para gestionar el reloj. Es posible realizar una transición entre ambas pestañas, pulsando el botón relativo a cada pestaña situado en la parte superior de la aplicación o realizando un gesto de *scroll* horizontal.

El prototipo cuenta con la posibilidad de conectarse con un dispositivo que cuenta con tecnología Bluetooth de bajo consumo (*BluetoothLE*), que es el encargado de obtener los diferentes datos del usuario para la monitorización. Para ello se implementó un servicio *BuscarPulseraService* que una vez iniciado, se encarga de buscar los dispositivos cercanos que cuentan con BluetoothLE. Para ello se utiliza el método *startLeScan()* de la clase *BluetoothAdapete*r, que retorna los resultados a través de un *callback*. Cada dispositivo encontrado es un objeto de tipo *BluetoothDevice* que contiene información necesaria para poder realizar una conexión posteriormente como su nombre o su dirección *MAC (Media Access Control)*. Este proceso tiene un consumo de batería elevado y por ello, tras 10 segundos de búsqueda se llama a *stopLeScan()* para detener el escaneo de dispositivos. En

cualquier caso, el usuario podrá solicitar otro escaneo si no ha encontrado el dispositivo con el que quiere realizar la conexión.

Una vez obtenida la lista de dispositivos cercanos es posible conectarse a uno de ellos. Para gestionar este procedimiento y los posteriores intercambios de información entre el prototipo Android y el dispositivo BluetoothLE se implementó un servicio *BluetoothLeService*, que contiene diferentes métodos para establecer y gestionar dichos intercambios. Para realizar la conexión con el dispositivo se llama al método *connect()* del servicio BluetoothLeService. Este método recibe como único argumento la dirección MAC del dispositivo y llama al método *BluetoothDevice.connectGatt()* para solicitar la conexión. Uno de los parámetros que recibe este método es un objeto de tipo *BluetoothGattCallback* que detecta si existe algún cambio en el estado de la conexión. De esta forma es posible conocer si la conexión se realizó con éxito.

A continuación, para poder recibir la información de la frecuencia cardiaca en la aplicación es necesario activar las notificaciones de la característica BluetoothLE correspondiente a la medición del ritmo cardiaco. Esta característica es *Medición ritmo cardiaco*, cuyo identificador UUID se encuentra en la Tabla 1.1. Para ello se utiliza el método *setCharacteristicNotification()* implementado en el servicio BluetoothLeService que se encarga modificar el *flag* de las notificaciones de la característica a *true*. Además, de forma especial para esta característica es necesario acceder al descriptor *Client Characteristic Configuration* de la característica para activar las notificaciones.

Una vez realizado este procedimiento es posible solicitar información al dispositivo BluetoothLE y obtener dicha información para su procesamiento. En concreto para obtener la frecuencia cardiaca del usuario con la pulsera *Xiaomi Smart Band 4* es necesario, como se indica en [1], escribir los bytes *0x15, 0x01, 0x01* en la característica *Punto de control de la frecuencia cardiaca.* Tras realizar esta operación el sensor encargado de medir la frecuencia cardiaca se activa durante 30 segundos para realizar la estimación. Durante este

tiempo puede retornar un número indeterminado de valores de la frecuencia cardiaca del usuario. Cada uno de estos valores se obtienen a través del método *onCharacteristicRead()* del BluetoothGattCallback que se había pasado como argumento al realizar la conexión con el dispositivo. El formato en el que se devuelve la información puede ser UINT8 o UINT16. En la propia respuesta, como se indica en [2], se incluyen 8 bits de flags. El bit que se encuentra en la posición 0 indica el formato de la respuesta y tiene el valor 0 si la respuesta está en formato UINT8 y 1 si la respuesta está en formato UINT16. Una vez obtenida dicha información es posible obtener el valor de la frecuencia cardiaca en pulsaciones por minuto. Si la respuesta se encuentra en formato UINT8, el valor se encuentra en el campo 1 de la respuesta, mientras que si la respuesta tiene el formato UINT16 se encuentra en el campo 2.

Nombre	**UUID**
(Característica) Medición ritmo cardiaco	00002a37-0000-1000-8000-00805f9b34fb
(Característica) Punto de control de la frecuencia cardiaca	00002a39-0000-1000-8000-00805f9b34fb
(Servicio) Custom FEE0	0000fee0-0000-1000-8000-00805f9b34fb
(Característica) Steps	00000007-0000-3512-2118-0009af100700

Tabla 1.1. Identificadores UUID utilizados.

Por otro lado, la pulsera Xiaomi Smart Band 4 incorpora un servicio *custom* como se indica en [1], que no se encuentra dentro de la especificación BluetoothLE a través del cual es posible obtener información sobre el número de pasos que ha dado el usuario en el día, la distancia recorrida y las calorías que ha quemado. Dicho servicio es el *Custom Service FEE0.* Como se mencionó en el apartado 3.6 de la memoria, un servicio puede incluir diferentes características. En concreto este servicio tiene la característica *Steps,* a través de la cual se obtienen los 3 valores relacionados con la actividad del usuario anteriormente mencionados. Para obtener la información se debe leer dicha característica. La respuesta obtenida, como se indica en [3], se encuentra en formato UINT32 en formato *little-endian*, donde los bytes menos significativos se encuentran en la parte izquierda. El número de pasos dados durante el día se encuentran en los bytes 1-4, la distancia en metros en los bytes 5-8 y las calorías quemadas en los bytes 9-12. Con esta información es posible

obtener los valores en formato entero para utilizarlos posteriormente en las diferentes funcionalidades de la aplicación.

La principal funcionalidad de la aplicación consiste en realizar una monitorización continua tanto de la actividad física del usuario como de su frecuencia cardiaca. Para ello se realizan peticiones periódicas a la pulsera para obtener el número de pasos y la frecuencia cardiaca del usuario. Mediante la clase *Alarm* se crea un objeto *AlarmManager* que es el encargado de enviar un mensaje por *broadcast* de forma periódica. Además, es posible especificar el intervalo de repetición. Ese mensaje se recoge en la propia clase Alarm. A su recepción se envía de nuevo otro mensaje por broadcast. Este mensaje es recibido por el *Fragment* que gestiona las funcionalidades de la pulsera. Con cada recepción de dicho mensaje se invoca a las funciones que se encargan de leer el número de pasos y de activar el sensor de frecuencia cardiaca. De esta forma se obtienen mediciones periódicas de ambos valores que serán utilizados en funcionalidades como determinar el tipo de actividad que se encuentra realizando el usuario.

La pestaña del prototipo destinada al reloj inteligente se encarga de mostrar la información recopilada por dicho dispositivo. Para ello se hace uso de la API (*Application Programming Interface*) *DataLayer* que es parte de los servicios de *Google Play* y que ofrece un canal de comunicación. De esta forma es posible comunicar una aplicación para teléfonos Android con una aplicación para relojes inteligentes con el sistema operativo Wear OS. Dicha comunicación se utiliza para solicitar desde la aplicación del teléfono los registros de los últimos 14 días recogidos por la aplicación instalada en el reloj. De esta forma es posible visualizar estadísticas sobre la frecuencia cardiaca relacionada con el nivel de actividad física en la pantalla del teléfono, que es considerablemente más grande.

El primer paso consiste en enviar un mensaje desde el prototipo Android para solicitar los datos. Para ello se utiliza un objeto de tipo *MessageClient* que se recibe en el servicio *TransferService* que forma parte del prototipo implementado en el reloj. Dicho mensaje se

recibe en un *listener* de tipo *MessageClient.OnMessageReceivedListener*. A su recepción se invoca un método que se encarga de enviar los registros de los últimos 14 días, que están contenidos en 14 ficheros, uno para cada día. Dicho método utiliza un objeto de tipo *DataClient* para realizar el envío. En total se realizan 14 envíos, uno para el registro de cada día. En cada envío se crea un *PutDataMapRequest* que contiene tres pares clave/valor. En el primero se envía el nombre del fichero, que contiene la fecha del registro. En el segundo se envía el contenido del fichero cuyo nombre se corresponde con el del primer par clave/valor. Por último, en el tercero se envía una cadena de texto que indica si todavía quedan más registros por enviar o si el envío actual es el último que se va a realizar.

Todos los envíos provocan un cambio en la capa de datos. Estos cambios son detectados por el listener implementado en el Fragment donde se gestionan todas las funcionalidades correspondientes al reloj. Cada vez que recibe el registro correspondiente a un día del reloj, se guarda en el espacio de almacenamiento del teléfono dicho registro en un fichero con el nombre que aparece en el primer par clave/valor. De esta forma la aplicación del teléfono tiene disponibles los datos del reloj para poder procesarlos y mostrarlos al usuario de forma resumida en un gráfico donde se visualiza la frecuencia cardiaca media de cada día, y en una tabla que muestra los valores medios, máximos y mínimos para cada nivel de actividad en cada día.

Para la implementación del gráfico donde se pueden visualizar las pulsaciones medias de los últimos 14 días se ha utilizado la librería MPAndroidChart [4] creada específicamente para Android. Permite crear diferentes tipos de gráficos como gráficos de líneas, gráficos de barras o gráficos circulares. Tiene numerosas opciones de configuración y permite interactuar con los diferentes elementos del gráfico. De esta forma es posible, por ejemplo, pulsar sobre un valor en el gráfico y ejecutar una determinada acción. Esto se consigue mediante un listener de tipo *OnChartValueSelectedListener.*

1.2 Prototipo Android para la separación de sonidos corazón-pulmón.

En Android es posible integrar código escrito en C/C++ en una aplicación utilizando el *kit de desarrollo nativo (Native Development Kit, NDK)* [5]. En él se incluyen un conjunto de herramientas para compilar código C/C++ en una biblioteca nativa y empaquetarlo en el archivo APK (*Android Application Package*) utilizando *Gradle* (herramienta de automatización de la construcción de código usada en Android). El APK es el archivo que contiene los datos necesarios para instalar una aplicación y para poder ejecutarse.

El NDK resulta útil en casos donde se necesite obtener un rendimiento mayor del dispositivo ejecutando aplicaciones que requieren un alto nivel computacional. También permite el uso de bibliotecas C/C++ de otros desarrolladores, como las mencionadas en la sección 3.8 de la memoria de este trabajo. NDK proporciona además la herramienta *CMake* para compilar la biblioteca nativa.

Existen dispositivos Android que utilizan diferentes tipos de CPU, por ello el NDK proporciona soporte para CPU'S basadas en ARM de 32 bits y en ARMv8-A (de 64 bits). También tiene soporte para CPUs que admiten los conjuntos de instrucciones x86 y x86-64. Por defecto la compilación se realiza para todos los tipos de CPU. De esta forma se obtiene un archivo APK multiplataforma. También es posible especificar en el fichero Gradle de la aplicación mediante *abiFilters* los tipos de CPU para los que se debe realizar la compilación. Así se puede reducir el tamaño del archivo APK.

Dentro del NDK se incluyen una serie de bibliotecas que se pueden utilizar directamente en el código C/C++ utilizando la sentencia *#include* en el encabezado del código. Para el código en C se incluyen encabezados de la biblioteca estándar C11, mientras que para el código en C++ se incluye compatibilidad con C++17.

Para este trabajo es necesario compilar, además del código desarrollado en C para la separación de sonidos cardio-pulmonares, las bibliotecas FFTW, OpenBLAS y OpenMP. En

las últimas versiones de NDK se ha eliminado el soporte para gcc/g++ y por ello la compilación de estas bibliotecas es más complicada, ya que hay que utilizar el compilador Clang. OpenMP presenta un grado de integración mayor, Clang lo soporta de forma nativa con lo que simplemente hay que modificar los *flags* de compilación. En el caso de FFTW, la propia biblioteca incluye los archivos necesarios para su compilación mediante CMake. En este caso hay que configurar los flags de compilación e indicar la ruta donde se encuentra la biblioteca en el archivo de compilación de la aplicación (*CMakeLists.txt).* Por último, la biblioteca OpenBLAS no incluye archivos para su compilación mediante CMake. Por ello es necesario utilizar las herramientas explicadas en [6] que proporciona Google. De esta forma es posible realizar la compilación de la biblioteca en una arquitectura diferente a la del dispositivo donde se realizará la ejecución. Las arquitecturas destino soportadas son ARM de 32 y 64 bits, x86 y x86-64.

Una vez compilado el código nativo de una biblioteca es posible llamar a las funciones nativas incluidas en dicha biblioteca utilizando la *Interfaz Nativa de Java (Java Native Interface, JNI).* De esta forma es posible ejecutar código Java en la Máquina Virtual Java que interactúe con bibliotecas escritas en C/C++ y código ensamblador.

1.3 Estimación de la frecuencia cardiaca a partir del sonido

En la sección 3.7 de la memoria se explicó el proceso para realizar la separación de sonidos cardio-pulmonares, a partir de una mezcla donde ambas fuentes sonoras se encuentran solapadas tanto en el dominio de la frecuencia como en el del tiempo. Realizada la separación es posible utilizar las señales, del corazón o de los pulmones, para diversos fines, entre ellos, la estimación de la frecuencia cardiaca.

Para ello, existen diferentes alternativas, algunas de ellas son las siguientes:

- A partir del espectrograma reconstruido de la señal correspondiente a los sonidos procedentes del corazón se calcula la densidad espectral de potencia (PSD). Con ello se obtiene la información que se muestra en Figura 1.1. Posteriormente, se buscan

los máximos locales que se encuentran por encima del valor 0,4W/Hz, obteniendo, de esta forma, el número de ruidos del corazón. Nótese que, cada latido del corazón está compuesto principalmente por dos ruidos (R1 y R2), tal y como se menciona en el apartado 2.2 de la memoria. Conocido el número de ruidos y la longitud del sonido analizado se obtiene, finalmente, el número de pulsaciones por minuto.

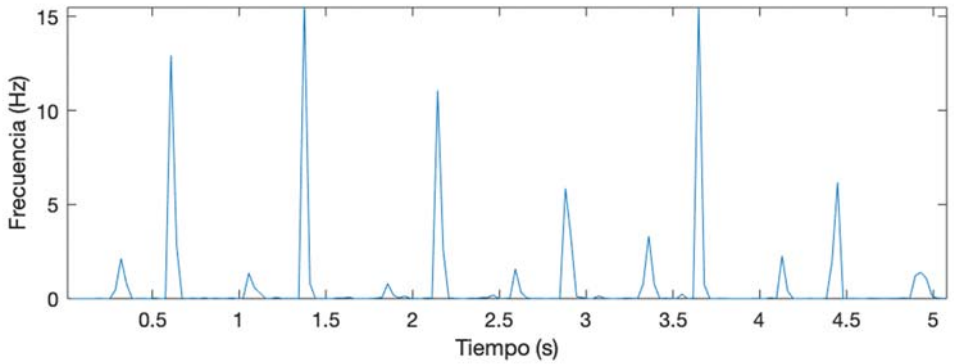

Figura 1.1. Densidad espectral de potencia de la señal del corazón.

- A partir de la información que se obtiene con el método anterior (la PSD de la señal del corazón) se buscan los máximos locales que se encuentran por encima de los 2,5W/Hz. De esta forma se obtiene únicamente ruidos que se producen en cada latido del corazón. Al igual que en el método anterior, con el número de ruidos y la longitud del sonido analizado es posible obtener las pulsaciones por minuto.
- Conceptualmente es un planteamiento diferente porque para la estimación utiliza información temporal, y no del dominio de la frecuencia. Calcula la correlación de las activaciones cardiacas estimadas, ordena los picos por la fuerza de la periodicidad y selecciona el pico con mayor periodicidad, dentro de los límites de ritmos cardiacos mínimo/máximo en el ser humano (entre 40 y 190 pulsaciones por minuto). Este modelo elimina la necesidad de seleccionar picos mediante umbrales y, además, al usar límites de ritmos cardiacos, evita errores de correlación o pulsaciones sin sentido. En la Figura 1.2 se muestran los picos ordenados y el pico seleccionado. Con esa información ya es posible calcular la frecuencia cardiaca en pulsaciones por minuto.

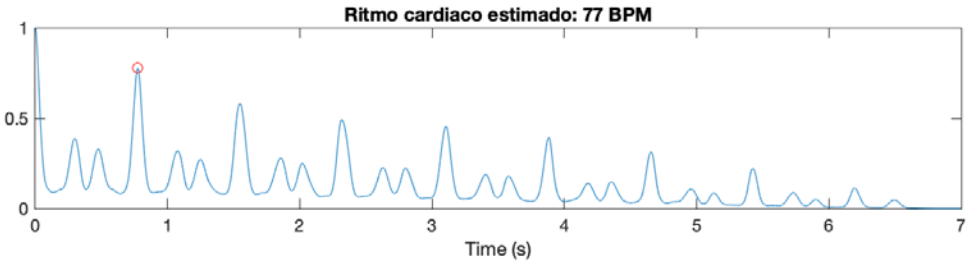

Figura 1.2. Picos ordenados por prominencia de periodicidad.

Los tres métodos expuestos obtienen, en la práctica, los mismos resultados. Las pruebas se han realizado con el archivo de audio de 7 segundos y sus derivados. Son necesarios experimentos adicionales, con audios reales obtenidos directamente por el prototipo construido, para determinar cuál de los tres métodos es más estable y preciso.

Bibliografía

[1] A. Nikishaev, «Medium,» [En línea]. Available: https://medium.com/machine-learning-world/how-i-hacked-xiaomi-miband-2-to-control-it-from-linux-a5bd2f36d3ad. [Último acceso: 20 06 2020].

[2] «Bluetooth,» [En línea]. Available: https://www.bluetooth.com/xml-viewer/?src=https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.heart_rate_measurement.xml#tree0:;. [Último acceso: 27 06 2020].

[3] M. Chmielewski, «WithIntent,» [En línea]. Available: https://withintent.com/blog/introduction-to-bluetooth-le-on-ios-mi-band-2-case-study/. [Último acceso: 20 06 2020].

[4] P. «GitHub,» [En línea]. Available: https://github.com/PhilJay/MPAndroidChart. [Último acceso: 20 06 2020].

[5] Google, «Google Developers,» [En línea]. Available: https://developer.android.com/ndk/guides/concepts?hl=es. [Último acceso: 23 06 2020].

[6] Google, «Google Developers,» [En línea]. Available: https://developer.android.com/ndk/guides/other_build_systems?hl=es-419. [Último acceso: 23 06 2020].