TECNIACUSTICA VALENCIA · 2015

46º CONGRESO ESPAÑOL DE ACÚSTICA
ENCUENTRO IBÉRICO DE ACÚSTICA
EUROPEAN SYMPOSIUM ON VIRTUAL ACOUSTICS
AND AMBISONICS

GESTOR DE APLICACIONES DE PROCESADO DE SONIDO SOBRE ANDROID PARA LA CREACIÓN DE REDES DE NODOS ACÚSTICOS CON DISPOSITIVOS COMERCIALES

PACS: 43.60.Dh.

Estreder, Juan; Antoñanzas, Christian; Piñero, Gema; De Diego, María
Instituto de Telecomunicaciones y Aplicaciones Multimedia (iTEAM)
Universitat Politècnica de València
Camino de Vera s/n; Edificio 8G
46022 Valencia (España)
Tel.: +34 963 879 761
E-mail: juaescam@iteam.upv.es, chanma@iteam.upv.es

ABSTRACT

This article describes the development of an application on Android able to manage Wi-Fi connections between mobile devices (smartphones and tablets) and Bluetooth connections between each device and a wireless speaker. In this way it is possible to create a network of two or more acoustic nodes using commercial devices, avoiding the design of ad-hoc acoustic nodes. The ability to use speakers connected via Bluetooth gives a great versatility to the network deployment and offers better performance in applications that need good quality of the reproduced sound as transaural reproduction or generalized cross-talk cancellation.

RESUMEN

En este artículo se presenta el desarrollo de una aplicación sobre Android capaz de gestionar conexiones Wi-Fi entre dispositivos comerciales (Smartphone o Tablet) y conexiones Bluetooth entre cada dispositivo y un altavoz inalámbrico. De esta forma se consigue crear una red de dos o más nodos acústicos con dispositivos comerciales sin necesidad de diseñar un nodo ad-hoc. El hecho de poder usar altavoces inalámbricos (mediante conexión Bluetooth) permite gran versatilidad en el despliegue de la red y mejor rendimiento en aplicaciones de campo sonoro como son la reproducción transaural o la cancelación generalizada de cross-talk.

INTRODUCCIÓN

Las redes de sensores inalámbricos (*Wireless Sensor Networks*, WSN) son una solución barata, flexible y eficiente para la vigilancia del medio ambiente y el hábitat, así como para el seguimiento y mantenimiento de equipos industriales [Akyildiz2002, Martinez2004, Puccinelli2005] entre otras tareas. Ellas han sido objeto de investigación desde hace más de

diez años, aunque su uso comercial no está tan extendido como se esperaba. Desde sus inicios las redes de sensores han sido también usadas en aplicaciones acústicas [Maroti2004, Chen2004], dando lugar a lo que se denomina redes de sensores acústicos inalámbricos (*Wireless Acoustic Sensor Networks*, WASN), cuya característica particular con respecto a una WSN es que utilizan micrófonos, o cualquier otro tipo de sensor acústico. En una WASN, el nodo consiste en uno o más micrófonos conectados a un procesador con algún tipo de comunicación y con capacidad de procesado. Las aplicaciones que hacen uso de este tipo de nodos acústicos son numerosas, ver [Bertrand2001, Bertrand2015] y sus referencias. En general, las WASN se centran en la estimación de una señal o parámetro común que es medido por todos los nodos, o en la estimación de señales específicas de cada nodo que comparten algunas propiedades o parámetros comunes con el resto. Otra característica típica de este tipo de nodos tiene que ver con su electrónica, que además de dedicarse a la grabación del sonido y a tareas de control y comunicación, normalmente también puede ejecutar algoritmos de procesamiento de señal.

Sin embargo, si se tienen en cuenta aplicaciones de control y reproducción de campo sonoro, entonces las WASN deben estar compuestas por nodos capaces de emitir sonidos a través de un altavoz o actuador. Por otra parte, la red no sólo debe hacer tareas de estimación, sino que también tiene que generar las señales adecuadas que alimentarán los altavoces y controlarán el campo sonoro [Ferrer2015]. En este sentido, a lo largo de este artículo consideraremos un nodo acústico genérico como una unidad de procesado que puede grabar y reproducir señales sonoras, procesarlas y comunicarse con otros nodos para intercambiar información local y estado de la red. Por tanto, una WASN no solo monitoriza su propio entorno sonoro, sino que también es capaz de modificarlo.

Desde un punto de vista práctico, una WASN podría implementarse utilizando *smartphones* o tabletas como nodos acústicos. En general este tipo de dispositivos tienen uno o dos micrófonos, un altavoz, y sus procesadores exhiben una potencia similar a los de los ordenadores portátiles (incluso mejor en muchos modelos recientes). También pueden comunicarse con otros dispositivos a través de conexiones inalámbricas Wi-Fi IEEE 802.11 o Bluetooth. En este sentido, el número de aplicaciones de procesamiento de sonido en dispositivos móviles ha ido en aumento durante los últimos años. La mayoría de ellas están pensadas para el entretenimiento, pero no sólo. Algunos ejemplos recientes son [Kim2014], donde varios dispositivos móviles reproducen el mismo sonido simultáneamente con el fin de proporcionar una experiencia de sonido envolvente como si estuvieran dentro de un teatro, y [Kardous2014], en el que se examina el rendimiento de una amplia gama de aplicaciones móviles capaces de medir el nivel de presión sonora emulando un sonómetro.

En este artículo se explican los procedimientos necesarios para el desarrollo de una WASN implementada sobre Android con capacidad de gestionar conexiones Wi-Fi entre varios dispositivos y conexiones Bluetooth entre cada dispositivo y un altavoz inalámbrico. El hecho de poder usar altavoces inalámbricos (mediante conexión Bluetooth) permite gran versatilidad en el despliegue de la red y mejor rendimiento en aplicaciones de control de campo sonoro debido a la mejor respuesta en frecuencia de dichos altavoces respecto a los embebidos. A lo largo del artículo se desarrollarán los pasos necesarios para gestionar y realizar las conexiones Wi-Fi y Bluetooth de los dispositivos. Se tendrá en cuenta la sincronización y se explicará también como realizar el procedimiento para estimar los canales acústicos de la WASN.

En este punto conviene establecer el contexto en el que se lleva a cabo este desarrollo:

1. Para la programación en Android (Java), se ha hecho uso del entorno *Eclipse* [eclipse].

2. Aunque este desarrollo está pensado para una red con varios nodos, por sencillez y comodidad se ha desarrollado en una primera etapa para dos nodos. Cada nodo dispone de su propio micrófono y está conectado a un altavoz inalámbrico.

3. La conexión entre dispositivos móviles se realiza mediante Wi-Fi y la conexión entre un dispositivo móvil y su altavoz mediante Bluetooth. La WASN resultante se muestra en la Figura 1. En ella se señalan también las señales grabadas, $x_i(n)$, reproducidas, $s_i(n)$, y los caminos acústicos correspondientes, $\mathbf{c}_{ij}$.

4. La conexión Wi-Fi se realiza con el protocolo TCP en el puerto 6555. Por otra parte, la conexión Bluetooth utiliza el perfil A2DP [A2DP2012] que es el que define la transmisión de un *stream* de audio a través de la conexión establecida.

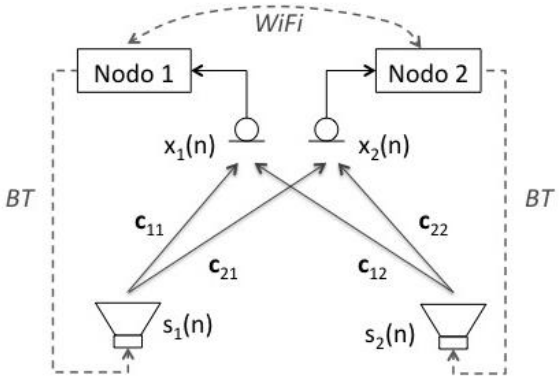

Fig. 1. Conexiones de una WASN con dos nodos mediante Wi-Fi y Bluetooth.

El resto de secciones están organizadas de esta forma: En la sección 2 se explican los procedimientos para gestionar las conexiones inalámbricas de los dispositivos entre sí y con sus respectivos altavoces. La sección 3 explica los procedimientos básicos necesarios para la estimación de los distintos canales acústicos y para el cálculo del retardo inherente a la conexión Bluetooth de los altavoces, mientras que la sección 4 trata de procedimientos de ajuste en la sincronización de los dispositivos a la hora de emitir sonidos. Por último, la sección 5 recoge las conclusiones más relevantes de este trabajo.

GESTIÓN DE LAS CONEXIONES

2.1 Conexión Wi-Fi

Como ya se ha comentado anteriormente, la conexión se establece mediante el protocolo TCP. Para gestionarla se usa la librería ***java.net***, la cual posee las clases[1] ***ServerSocket*** y ***Socket***. En primer lugar se establecen los siguientes roles: un dispositivo deberá actuar como servidor y el otro como cliente. Los roles son necesarios únicamente durante el proceso de conexión, una vez establecida dicha conexión ambos dispositivos tendrán los mismos atributos a la hora de enviar y recibir información.

[1] Es una plantilla implementada en software que describe un conjunto de objetos con atributos (variables) y comportamiento similares.

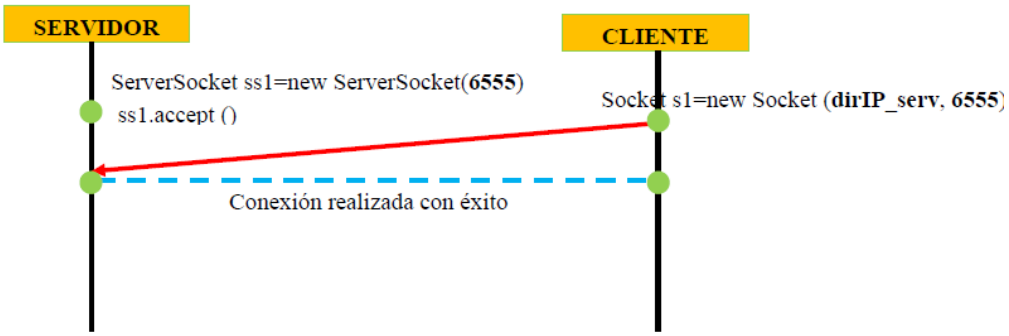

Fig. 2. Proceso de conexión cliente-servidor en Wi-Fi.

La Figura 2 muestra el proceso de conexión. Se definen dos dispositivos (**A** y **B**), donde **A** actúa como servidor y **B** como cliente. El dispositivo **A** tendrá que usar el método ***accept()*** de la clase ***ServerSocket*** para habilitar la escucha de peticiones de conexión por parte del cliente. Este método es bloqueante, lo que significa que hasta que no se haya establecido una conexión no se ejecutará la siguiente línea de código. Por otra parte el dispositivo **B** realizará la petición de conexión mediante la clase ***Socket*** y el método que usa también es bloqueante.

Como se puede observar en la Figura 2, se conectan a un puerto cuya elección la realiza el programador sin restricciones de ningún tipo, puede ser cualquier puerto libre. Una vez realizada la conexión, los dos dispositivos podrán comunicarse y enviarse información de forma bidireccional.

2.2 Conexión Bluetooth

La conexión Bluetooth es más sencilla que la anterior ya que solo se programa un dispositivo. Para la conexión se utiliza el perfil A2DP [A2DP2012] de Bluetooth, concretamente el método ***connect()*** de la clase ***BluetoothA2dp***, el cual se encuentra en la librería ***android.bluetooth***. Este método se tiene que llamar de manera especial (en Java se denomina **Reflection**) ya que el método no se muestra en el API, pero aparece en el código fuente de la clase ***BluetoothA2dp***. Una vez invocado, la conexión Bluetooth queda establecida. El uso de este perfil consigue que el audio se reproduzca por el altavoz inalámbrico anulando la reproducción por los altavoces del propio dispositivo móvil.

Cabe aclarar que en Bluetooth antes de conectarse a un dispositivo hay que vincularse a él. Un dispositivo conectado siempre está vinculado[2], pero un dispositivo vinculado no tiene por qué estar conectado. Por lo tanto, si se quiere conectar un dispositivo que no esté vinculado previamente, la aplicación lo vincula de manera transparente al usuario.

Procesado del sonido

En este apartado se describe el proceso de estimación de las distintas Respuestas al Impulso del Canal Acústico (RICA), es decir, la obtención de las $\mathbf{c}_{ij}$ i,j=1,2 de la Figura 1. También se describe el cálculo del retardo del canal asociado a la conexión Bluetooth, entendido como el tiempo que el sistema operativo tarda en reproducir un fichero de audio desde que el programa le ordena que lo reproduzca por el altavoz inalámbrico hasta que efectivamente lo hace. Este retardo depende del hardware y software utilizado. En pruebas preliminares para un sistema operativo Android 4.4 o superior, una versión de Bluetooth 2.1 o superior, y distintas marcas de

[2] La vinculación Bluetooth es un mecanismo de seguridad que sirve para restringir el acceso a usuarios autorizados.

altavoces, este retardo está entre los 150 y los 300 ms. Dado que cada RICA presentará un retardo Bluetooth distinto, y este es un dato ajeno al camino acústico en sí, la aplicación desarrollada utiliza el retardo Bluetooth de cada RICA para poder obtener una misma referencia temporal en todas ellas, tal y como se explica más adelante.

3.1 Cálculo del Retardo Bluetooth

Para calcular el retardo sufrido por el sistema la aplicación utiliza la correlación entre las señales reproducida y grabada. La señal a reproducir debe tener las mismas características que un ruido blanco en cuanto a su función de correlación. En nuestra aplicación hacemos uso de Secuencias de Máxima Longitud (*Maximum Length Sequences*, MLS) [Rife1989].

Se define la correlación como:

$$R_x(n_0) = \sum_{n=0}^{T-n_0} xrep(n+n_0)xrec(n) \qquad (1)$$

donde n_0 es el índice de la muestra temporal, *xrep* es la señal que reproduce el altavoz, *xrec* es la señal que se capta con el micrófono del dispositivo móvil y T es la longitud de estas dos señales. Una vez calculada (1), se obtiene la posición del valor máximo de la correlación y a partir de ella se calcula el retardo en segundos como:

$$\tau = (\arg\{\max_{n_0}\{R_x(n_0)\}\}) \cdot T_s \qquad (2)$$

donde T_s es el periodo de muestreo. La función "arg{max}" indica el índice de n_0 en el que está situado el máximo de R_x. Si se desea obtener el retardo en muestras, simplemente se calcularía (2) sin multiplicar por T_s. Los retardos así obtenidos se utilizarán en el siguiente apartado para estimar las RICAs de la WASN con una única referencia temporal. Téngase en cuenta que para cada canal acústico $\mathbf{c}_{ij}$ se calcula un retardo Bluetooth τ_{ij}.

Respecto a la programación de este algoritmo en Android, hay que tener en cuenta que una de las limitaciones de este lenguaje es la pobre gestión de los arrays (vectores y matrices), ya que para realizar operaciones entre ellos hay que hacerlas componente a componente.

3.2 Cálculo de la Respuesta al Impulso del Canal Acústico (RICA)

El otro algoritmo de procesado de sonido incluido en la aplicación es la estimación de los coeficientes de los canales acústicos $\mathbf{c}_{ij}$. Esta estimación se realiza mediante un procedimiento de identificación de sistemas y hace uso del algoritmo adaptativo *Least Mean Squares* (LMS), tal y como se muestra en la Figura 3.

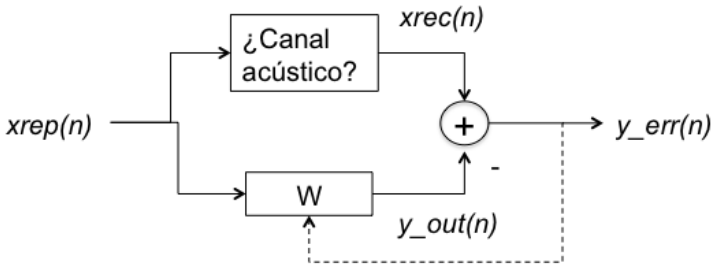

Fig. 3. Procedimiento usado para estimar la RICA mediante LMS.

El algoritmo en Android hace uso de las siguientes variables:

- Un array llamado *Buf_fA* de longitud L.
- Una variable llamada *y_out* de salida del filtro adaptativo W.
- Una variable llamada *y_err* que mide el error en cada iteración del algoritmo.
- Una variable llamada *mu* constante y de pequeño valor.
- Un array llamado *W* de longitud L que contiene los coeficientes de la RICA estimada en esa iteración.

A continuación se explica el procedimiento iterativo realizado sobre Android:

1. Se refresca la variable *Buf_fA* mediante:
   >> Buf_fA(2:L)= Buf_fA(1:L-1)
   >> Buf_fA(1)=xrep(i)
2. Se calcula *y_out* como:
   *>> y_out= w*Buf_fA*
3. A continuación se calcula:
   >> y_err=xrec(i)-y_out
4. Y por último se almacena la estimación del canal acústico en w:
   *>> w=w+Buf_fA*y_err*mu*

Estos cuatro pasos se realizan desde i=1:Lx, donde Lx es la longitud de las señales *xrec* y *xrep*, o la longitud mínima de ambas. Como se ha comentado anteriormente, las operaciones con arrays en Android se hacen componente a componente, por lo que dentro del bucle principal, hay otros dos más correspondientes a las instrucciones 2 y 4. Por último, en la instrucción 1 se copia un vector a otro y en este caso Android hace uso del método ***System.arraycopy*** que sí es un método eficiente.

Una vez calculados los canales acústicos $\mathbf{c}_{ij}$, la aplicación los recorta a partir de una misma referencia temporal para todos ellos, eliminando el retardo inherente a las conexiones Bluetooth. Sea t_{min} el valor mínimo de todos los retardos τ_{ij}. calculados mediante la ecuación (2). A partir de dicho valor mínimo la aplicación recorta todas las RICAs estimadas entre $t_{min} - L_p*T_s$ y $t_{min} + L_c*T_s$, siendo L_p una longitud de guarda para no recortar componentes relevantes del canal, y L_c la longitud de los canales acústicos. Nótese que el algoritmo de estimación usa una longitud L que debe ser siempre mayor que la longitud de canal acústico real L_c debido a los retardos propios de la conexión Bluetooth.

Sincronización

La aplicación contempla dos tipos de sincronización:

- Sincronización grabación-reproducción.
- Sincronización entre dispositivos.

La primera es la utilizada durante el proceso de estimación de la RICA y cálculo del retardo, ya que la señal de reproducción *xrep* y la grabada *xrec* deben empezar lo más simultáneamente posible, aunque nunca antes la reproducción que la grabación. La segunda sincronización se refiere a la conexión Wi-Fi, y trata de que el retardo de esta conexión y el offset de tiempo entre dispositivos afecte lo menos posible a la estimación de los canales acústicos cruzados ($\mathbf{c}_{ij}$, con *i* distinto de *j*).

4.1 Sincronización Grabación - Reproducción

En primer lugar es importante explicar qué es un hilo de ejecución en Android, denominado **Thread**: es la unidad de procesamiento más pequeña que permite que una aplicación pueda hacer tareas concurrentemente. De esta forma, si se desea reproducir y grabar a la vez, es necesario usar dos hilos de ejecución, uno para grabar (**Recording Thread**) y otro para reproducir (**Reproduction Thread**).

Los dos métodos de Android que se usan en esta operación son:

- *wait()* : El hilo que lo llama espera indefinidamente hasta que le notifican que pare, o bien haya una interrupción inesperada.
- *notify()*: Se usa para hacer despertar (o dejar de esperar) al hilo que se llamó mediante *wait(),* y tiene que ser llamado por otro hilo.

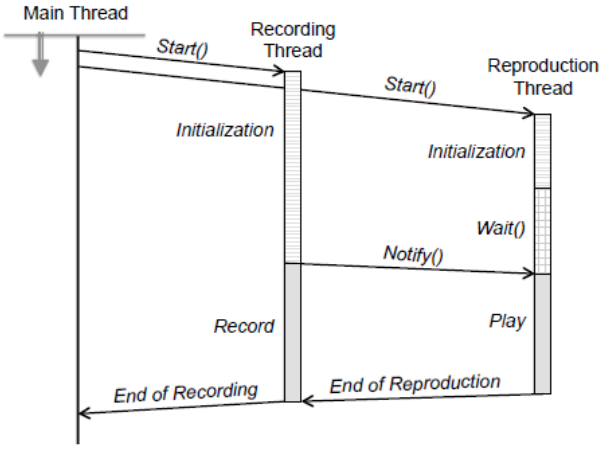

Fig. 4. Procedimiento de sincronización grabación-reproducción.

En general las inicializaciones del **Recording Thread** son más costosas en tiempo que el **Reproduction Thread**, siendo el primero el que utiliza *notify()* y el segundo *wait()*. El esquema de la Figura 4 muestra ambos hilos y sus relaciones.

4.2 Sincronización Entre Dispositivos

Esta sincronización ayuda a que la reproducción de un altavoz y la grabación del dispositivo móvil que no está conectado a él empiecen simultáneamente. El procedimiento es el que aparece en el diagrama de la Figura 5. En primer lugar se asignan roles de maestro y esclavo y se establece la conexión. En el establecimiento de la conexión ambos dispositivos obtienen su propia referencia temporal y la almacenan en **tm** y **ts** respectivamente. El maestro ejecuta entonces la inicialización de sus variables y avisa al esclavo, después el maestro se queda esperando y el esclavo ejecuta la inicialización de sus variables y avisa al maestro. En ese momento el maestro coge una nueva referencia temporal **tm1** y le transmite al esclavo la diferencia **tm1-tm**. El esclavo toma entonces su propia referencia temporal **ts1** y calcula el tiempo de retardo o adelanto que lleva respecto al maestro como **(ts1-ts) – (tm1-tm)**. A partir de aquí ambos dispositivos esperan un tiempo fijo mediante el método **Thread.sleep(millisec)**, donde **millisec** son los milisegundos que el dispositivo espera, tal y como se muestra en la parte inferior de la Figura 5.

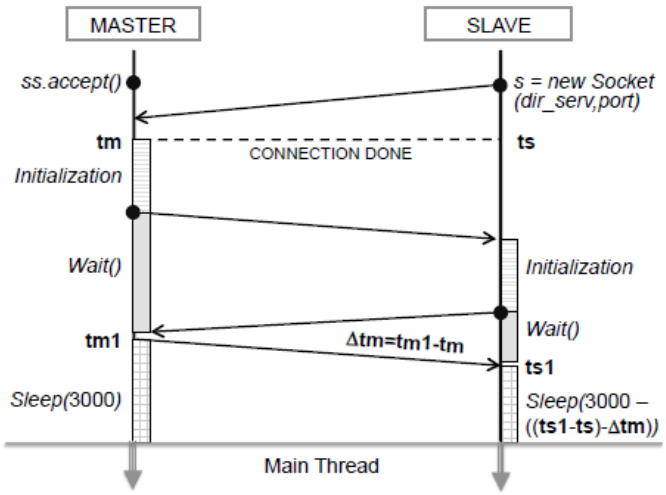

Fig. 5. Procedimiento de sincronización entre dispositivos.

Cabe destacar que esta sincronización debe hacerse con procedimientos ad-hoc como el mostrado en la Figura 5 porque Android no proporciona ningún método directo para poder sincronizar dos dispositivos entre sí, solo para sincronizarlos a una red a través de un servidor, pero cada uno de ellos de forma independiente.

Conclusiones

En este artículo se ha explicado el desarrollo de una aplicación programada en Android para establecer y gestionar las conexiones y procedimientos básicos necesarios para implementar una WASN que haga usos de dispositivos con dicho sistema operativo (tanto *smartphones* como *tablets*). A lo largo del artículo se han expuesto los pasos necesarios para gestionar y realizar las conexiones Wi-Fi y Bluetooth de los dispositivos, cómo sincronizarlas y como estimar los canales acústicos de la WASN teniendo en cuenta los retardos inherentes a las conexiones y al propio sistema operativo.

AGRADECIMIENTOS

Este trabajo ha sido financiado por la Unión Europea FEDER y el Gobierno español a través del proyecto TEC2012-38142-C04, y la Generalitat Valenciana a través del proyecto PROMETEOII/2014/003.

Referencias

[A2DP2012] **“**Advanced Audio Distribution Profile (A2DP) Specification, version 1.3”**,** Bluetooth Special Interest Group Std., July 2012. [Online]. http://www.bluetooth.org/en-us/specification/adopted-specifications.

[Akyildiz2002] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, “A survey on sensor networks”, *Communications Magazine, IEEE*, 40 (8) 102–114 (2002).

[Bertrand2011] A. Bertrand, "Applications and trends in wireless acoustic sensor networks: A signal processing perspective", *in Communications and Vehicular Technology in the Benelux (SCVT), 2011 18th IEEE Symposium on*, pp.1–6 (2011).

[Bertrand2015] A. Bertrand, S. Doclo, S. Gannot, N. Ono, and T. van Waterschoot, “Special issue on wireless acoustic sensor networks and ad hoc microphone arrays,” *Signal Processing*, 107, 1-3 (2015).

[Chen2004] W.-P. Chen, J. Hou, L. Sha, “Dynamic clustering for acoustic target tracking in wireless sensor networks”, *IEEE Trans. Mobile Comput.*, 3 (3) 258–271 (2004).

[eclipse] https://eclipse.org/

[Ferrer2015] M. Ferrer, M. de Diego, G. Piñero, A. Gonzalez, “Active noise control over adaptive distributed networks”, *Signal Processing*, 107, 82-95 (2015).

[Kardous2014] C.A. Kardous, P.B. Shaw, “Evaluation of smartphone sound measurement applications,” *The Journal of the Acoustical Society of America*, 135 (4) 186–192 (2014).

[Kim2014] H. Kim, S. Lee, J.-W. Choi, H. Bae, J. Lee, J. Song, and I. Shin, “Mobile maestro: Enabling immersive multi-speaker audio applications on commodity mobile devices”, *in Proc. 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp ’14)*, 277–288 (2014).

[Maroti2004] M. Maroti, G. Simon, A. Ledeczi, J. Sztipanovits, “Shooter localization in urban terrain”, *Computer*, 37 (8) 60–61 (2004).

[Martinez2004] K. Martinez, J.Hart, R.Ong, “Environmental sensor networks”, *Computer*, 37 (8) 50–56 (2004).

[Puccinelli2005] D. Puccinelli and M. Haenggi, "Wireless sensor networks: applications and challenges of ubiquitous sensing", *Circuits and Systems Magazine, IEEE*, vol. 5, no. 3, pp. 19–31, 2005.

[Rife1989] D. D. Rife and J. Vanderkooy, “Transfer-function measurement with maximum-length sequences,” *J. Audio Eng. Soc*, 37 (6) 419–444 (1989).