# Reconocimiento automático de señales accesibles de semáforo en dispositivos móviles

Juan M. Fonseca-Solís
Centro de Invests. en Tecnologías de la Información y Comunicación, Universidad de Costa Rica
Email: juan.fonsecasolis@ucr.ac.cr

Sebastián Ruiz-Blais
Centro de Invests. en Tecnologías de la Información y Comunicación, Universidad de Costa Rica
Email: sebastian.ruizblais@ucr.ac.cr

Arturo Camacho-Lozano
Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica
Email: arturo.camacho@ecci.ucr.ac.cr

***Resumen*—En este artículo se presenta el diseño de una aplicación móvil que permite a las personas confirmar, mediante el sonido emitido, que un semáforo peatonal del tipo cucú les está cediendo el paso. Este es un problema interesante de resolver porque permite ayudar a quienes padecen discapacidades visuales y auditivas a transitar más seguros por las calles. Otros autores han propuesto soluciones basadas técnicas de visión por computadora, pero nosotros empleamos métodos de procesamiento de sonido. Los resultados obtenidos muestran que la aplicación implementada en Android tiene un buen rendimiento y que alcanza métricas de reconocimiento aceptables. Se incluyeron medidas para mejorar la accesibilidad. Como trabajo futuro se propone migrar la solución a *hardware* hecho a la medida.**

## I. Introducción

Según el informe del PROSIC del 2011, el 3.41 % de la población costarricense sufre de discapacidades auditivas y el 6.54 % de visuales; se desconoce cuántas personas sufren de ambas. En EE. UU. estos padecimientos también son serios: 10 millones de personas son parcial o totalmente ciegas y 28 millones tienen algún grado de sordera. Se sospecha que estas personas tienen dificultades para realizar tareas cotidianas e importantes como cruzar la calle de forma segura.[1]

Se conoce además que en en estos países existe una alta disponibilidad de teléfonos inteligentes, los cuales podrían ser empleados para ayudar a las personas a, entre otras cosas, cruzar la calle con mayor seguridad y confianza. Los teléfonos inteligentes son una plataforma factible para implementar sistemas portables, no solo por su bajo consumo de potencia y buen desempeño del procesador, sino también porque sus interfaces de programación de aplicaciones (API, por sus siglas en inglés) están optimizadas y bien documentadas [4]. De hecho, el sistema operativo Android incluye en sus API la *transformada rápida de Fourier* (FFT, por sus siglas en inglés), una herramienta ampliamente usada para procesar señales de audio y vídeo. Por medio de este tipo de procesamiento se ha logrado, por ejemplo, analizar imágenes capturadas por la cámara y reconocer los cruces peatonales pintados en la calle [1, 6], tecnología de potencial utilidad para personas con discapacidades visuales.

[1]Datos obtenidos de la National Blind Federation (https://nfb.org/fact-sheet-blindness-and-low-vision) y la National Court Reporters Association (http://www.ncra.org/Government/content.cfm?ItemNumber=9450), ambas de los EE. UU.

La mayoría de semáforos peatonales que hay en Costa Rica (sino todos) incluyen características de accesibilidad: indican al peatón que puede cruzar la calle por medio de señales visuales y auditivas. A estos dispositivos se les denomina *semáforos accesibles para peatones*. Los semáforos accesibles emiten sonidos factibles de reconocer, ya que tienen un contorno musical repetitivo y bien ubicado. En el país hay semáforos con de tres tipos de patrones sonoros: cucú, chirrido de alta frecuencia y chirrido de baja frecuencia.

Como parte de nuestro trabajo hemos creado un método para reconocer los sonidos de estos semáforos accesibles comparando su espectro con *kernels* diseñados especialmente para la tarea. La implementación realizada en Android reconoce los sonidos cucú, y busca ser robusta contra el ruido, sea armónico o inarmónico. En este artículo nos concentramos en evaluar esta implementación para verificar que cumple con requerimientos de acierto y rendimiento preestablecidos: respuesta a la frecuencia esperada, emisión oportuna de alertas, distancia máxima admitida en el rango tolerable y bajo consumo de memoria. Asimismo que fuera accesible y usable.

El artículo está organizado como sigue: la sección II explica el algoritmo (en su estado actual), la sección III explica la metodología de verificación propuesta, la sección IV presenta los resultados obtenidos, la sección V propone el trabajo futuro y la sección VI presenta las conclusiones.

## II. Reconocimiento de sonidos cucú

Los sonidos cucú emitidos por los semáforos accesibles se caracterizan por formar una secuencia de dos tonos: uno de 900 Hz y el otro de 1100 Hz. El algoritmo de reconocimiento funciona como sigue. Para cada ventana de tiempo, los dos tonos son identificados mediante el cálculo del producto normalizado de la magnitud espectral $A(t,f)$ y un *kernel* armónico $K_h$ de la siguiente forma:

$$H(f_0,t) = \frac{\int \sqrt{A(t,f)} K_h(f_0,f) df}{\sqrt{\int A(t,f) df} \sqrt{\int [K_h^+(f_0,f)]^2 df}}, \qquad (1)$$

donde:

$$K_h(f_0,f) = \alpha \sin(2\pi(f/f_0 - 0.75)),$$

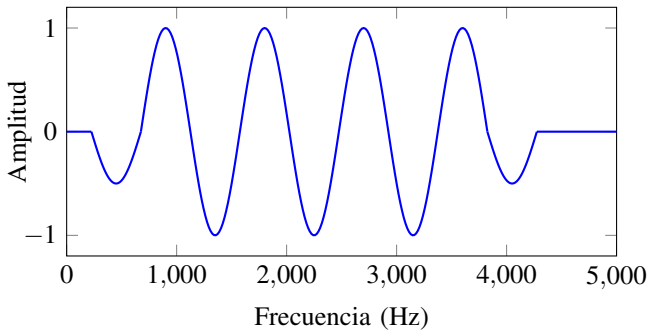



Figura 1: *Kernel* para un tono armónico con $f_0 = 900$ Hz y cuatro armónicas.

con:

$$\alpha = \begin{cases} 1, & 0.75 < \frac{f}{f_0} < 0.25 + N \\ 1/2, & 0.25 < \frac{f}{f_0} < 0.75 \vee 0.25 + N < \frac{f}{f_0} < 0.75 + N \\ 0, & \text{en otro caso} \end{cases}$$

y donde $K_h^+(f_0, f) = \text{máx}\{0, K_h(f_0, f)\}$ y $N$ es el número de armónicas ($N = 4$ para el tono de 900 Hz y $N = 3$ para el tono de 1100 Hz). Un ejemplo gráfico del *kernel* se muestra en la figura 1. Si la frecuencia fundamental de puntaje más alto alcanza un valor de 0.14, es guardada en la ventana de tiempo correspondiente. Cuando el puntaje no alcanza ese umbral, se le asigna una frecuencia de cero.

El patrón de dos tonos es reconocido cuando se alcanza la proporción de predicciones de frecuencia correspondientes a 70 ms del tono armónico en 1100 Hz, y 140 ms del tono armónico en 900 Hz, con una separación de 200 ms. El puntaje es calculado como $S_h(t) = T/w_t$, donde $T$ es el número de ventanas en las que los tonos fueron predichos correctamente y $w_t$ es el total de ventanas. Los momentos de silencio no son considerados como parte de las grabaciones, porque pueden contener ecos, los cuales producen puntajes bajos y erróneos. Un sonido del tipo cucú es reconocido cuando $S_h$ sobrepasa el umbral de 0.45 (0.25 para la implementación).

## III. Metodología

### III-A. *Respuesta a la frecuencia*

Para determinar la respuesta a la frecuencia se implementó una prueba instrumental y automatizada construida con *AndroidJUnit4* y ejecutada con un teléfono LG G2 Mini con sistema operativo Android Kitkat 4.4. La versión de la aplicación que se probó fue la 1.0.130. La prueba consistió en hacer que el módulo evaluador del sonido cucú procesara una grabación sintética de 3 s que contenía un barrido de frecuencias en el rango de 1 Hz a 22 050 Hz. La grabación tuvo una frecuencia de muestreo de 44 100 Hz y una cuantificación de 32 bits. Los puntajes que el módulo evaluador emitió para cada tono puro del barrido fueron capturados en consola y asociados a las frecuencias respectivas del barrido. Esta prueba fue diseñada para verificar que el kernel diseñado en Octave estuviera implementado correctamente en Java, y que la FFT fuera bien calculada.

### III-B. *Pico de emisión de la primer alerta*

Cada "pico de actividad" del cucú está formado por el par de tonos 1100 Hz y 900 Hz, separados por 1230 ms. Para determinar el pico promedio de emisión de alerta se construyó una prueba automatizada con Octave y *Android Debug Bridge* (ADB). La prueba reprodujo 10 veces una grabación de un cucú por cada valor de amplitud en el rango 0.02–1 unidades, registrando el *timestamp* de cada alertas emitida.[2]

Con el fin de mejorar la estimación del SNR (relación señal-ruido) se empleó ruido blanco de distribución uniforme para tener un piso estable de ruido de fondo de 51 dB [3]. Esta prueba fue diseñada para asegurar que la alerta de cucú se emitiera a lo sumo en el quinto pico de actividad. Esta cantidad fue calculada con base en el ancho de los carriles de tránsito (3.3 m) reglamentada en el *Reglamento de espacios públicos, vialidad y transporte*, el número de carriles de las vías donde se ubican los semáforos accesibles y la máxima velocidad a la que puede caminar un adulto mayor (1.4 m/s) [5, 7, 9].

### III-C. *Distancia máxima soportada*

Con el fin de determinar la cantidad de alertas emitidas por el sistema cada 8 picos de actividad, se reprodujo una grabación de un sonido cucú (también de 8 picos) a distintos niveles de poder sonoro y a distintas distancias respecto de la fuente. Para simular el semáforo accesible se elevó un equipo de sonido Sony CFD-V25 a 2 m de altura, en dirección al teléfono, y se reprodujo repetitivamente una grabación de cucú con ruido de fondo de 50–55 dB.[3]

### III-D. *Evaluación del consumo de memoria*

Para evaluar el consumo de memoria se hizo una inspección manual del código y se usaron las siguientes herramientas de *Android Studio*: *Memory Monitor*, *Allocation Tracker* y *HPROF Viewer and Analyzer*. El objetivo de la prueba fue asegurar que el consumo de memoria se mantuviera constante al usar la interfaz gráfica, ejecutar el servicio y emitir alertas. También se buscó que no hubieran fugas de memoria y que se siguieran buenas prácticas de programación como: uso limitado de las variables globales y objetos temporales; limpieza de las librerías sin uso y preferencia por aquellas diseñadas para dispositivos móviles; sustitución de los *setters/getters* por miembros públicos; uso de la sintaxis mejorada para bucles; uso de mecanismos de sincronización de programación paralela; silenciamiento de registros de depuración temporales; restricción al mínimo en el uso de clases abstractas y uso de imágenes livianas (estas últimas al descomprimirse durante tiempo de ejecución aumentaban el consumo de memoria hasta 40 MB).[4]

[2]La prueba se ejecutó en el laboratorio con el teléfono colocado a 1 m de distancia de los parlantes de la PC.

[3]La grabación reproducida fue tomada al costado noroeste del Parque de las Garantías Sociales a las 6:44 a.m. en condiciones atmosféricas despejadas y con poco ruido de fondo.

[4]Estas buenas prácticas se recopilaron a partir de las recomendaciones del fabricante para el uso de memoria. Más información en https://developer.android.com/topic/performance/memory.html.

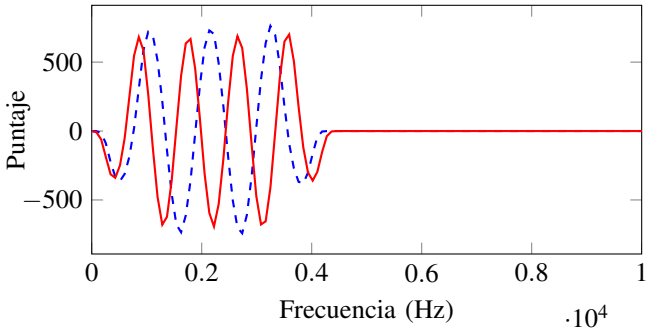



Figura 2: Respuesta a la frecuencia de los *kernels* de 1100 (- - -) y 900 Hz (——). El resto de entradas después de 1 kHz fue omitido por cuestiones de espacio, pero se comprobó que es cero de 4500 Hz en adelante.

Cuadro I: Amplitud, niveles de intensidad sonora y SNR

| Amplitud-señal | Señal (dB) | Ruido (dB) | SNR (dB) |
|---|---|---|---|
| 0.02 | 38 | 51 | -13 |
| 0.03 | 43 | 51 | -08 |
| 0.05 | 43 | 51 | -08 |
| 0.10 | 46 | 51 | -05 |
| 0.20 | 49 | 51 | -02 |
| 0.33 | 53 | 51 | 02 |
| 0.50 | 57 | 51 | 06 |
| 1.00 | 60 | 51 | 09 |

### *III-E. Accesibilidad y usabilidad*

Dado que el público meta consistió mayormente en personas con discapacidades visuales y auditivas, se optó por verificar que el sistema fuera accesible y usable. Se realizaron pruebas para comprobar que se cumpliera el principio POUR (*Perceivable, Operable, Understandable, y Robust*), es decir que el usuario pudiera saber fácilmente cuáles eran los controles, para qué servían, en qué estado se encontraban, cómo activarlos, y que estas facilidades fueran invariantes a las diferentes versiones del sistema operativo.[5]

## IV. Resultados

### *IV-A. Respuesta a la frecuencia*

Como lo muestra la figura 2, se obtuvo la respuesta deseada a la frecuencia, pues las formas obtenidas correspondieron con el *kernel* explicado en la sección II.

### *IV-B. Pico de emisión de la primer alerta*

El cuadro I muestra la relación entre los valores de amplitud de la señal y los niveles de intensidad sonora (SPL) medidos mediante un sonómetro Radioshack Plus 3300099 9V. Fue necesario calcular estas relaciones porque el SPL del valor de amplitud máximo de la PC era desconocido (además de que varía de equipo en equipo), y conociendo este valor junto con el SPL del ruido de fondo (51 dB) se podían simular las relaciones de SNR deseadas. La figura 3 muestra que para un

[5]Basado en las recomendaciones del sitio web: https://www.w3.org/TR/UNDERSTANDING-WCAG20/intro.html#introduction-fourprincs-head

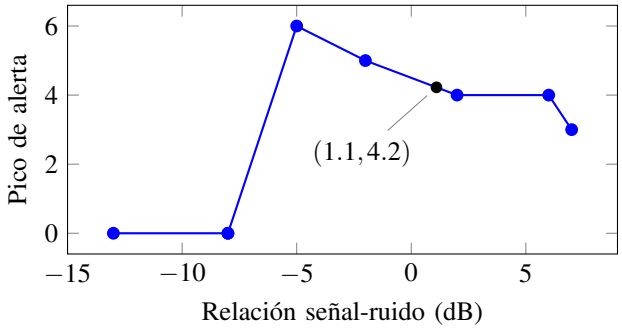



Figura 3: Pico de emisión de la primer alerta por nivel de SNR, cada punto es el promedio de haber realizado 10 iteraciones con el mismo valor. Un valor de cero significa que el sistema necesitó más de 8 picos de actividad para responder.

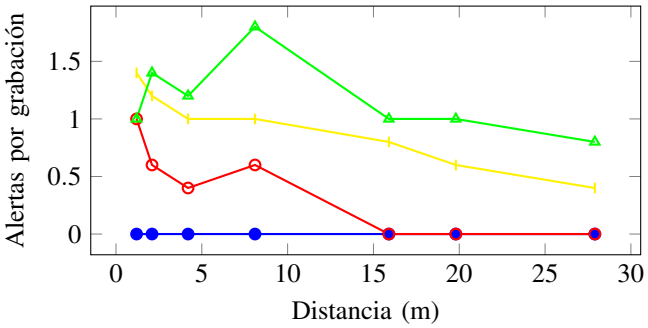



Figura 4: Promedio de la cantidad de alertas emitidas por la aplicación para distancias de 1 a 28 m. Se usaron grabaciones de cucús con 8 picos de actividad a varios niveles de SPL: 60 dB (—●—), 70 dB (—○—), 80 dB (—|—), 90 dB (—△—).

SNR de 1.1 dB se logró que la aplicación emitiera una alerta a más tardar en el cuarto pico de actividad (un pico menos del límite máximo), un resultado muy deseado.[6]

### *IV-C. Distancia máxima admitida*

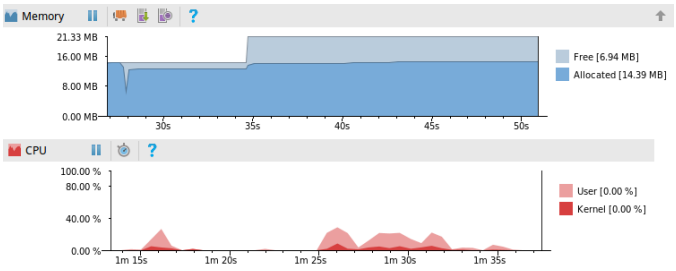
Los resultados obtenidos a varias distancias se muestran en la figura 4. Se observa que a ocho metros de alejamiento (un poco más de dos carriles) y con un nivel de 90 dB SPL (similar al nivel de intensidad sonora presente en las calles del centro de San José), la aplicación tiene la capacidad de emitir una alerta cada $8/1.8 \approx 4.4$ picos de actividad, lo cual es deseado por estar debajo de los 5. Sin embargo a un nivel de 80 dB y a la misma distancia, el sistema emitió una alerta cada 8 picos, lo cual debe mejorar. La misma métrica obtuvo la curva de 90 dB SPL cuando la distancia fue de 15.9 m (cinco carriles).

### *IV-D. Evaluación del consumo de memoria*

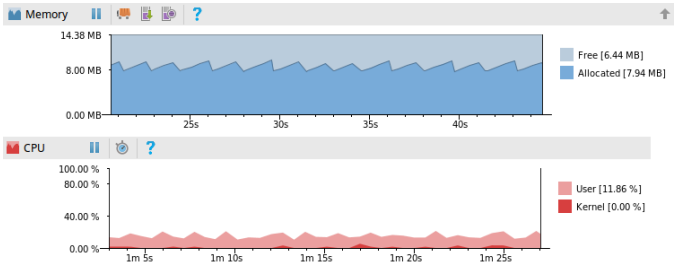
La figura 5a muestra que para la interfaz gráfica el consumo de memoria se mantuvo casi constante (15 MB) y el uso

[6]Esta es la relación que mantienen los dispositivos Novax DS-100 al autoajustarse para emitir tonos a +5 dB respecto del ruido ambiental (hasta un tope de 90 dB). El SNR se calculó como: $\text{SPL}_{\text{señal}} - \text{SPL}_{\text{ruido}}$ (pues ambas cantidades ya estaban en decibelios y por una propiedad de logaritmos).

del CPU tuvo picos de 35 %. La figura 5b muestra que para el servicio de emisión de alertas el consumo de memoria también se mantuvo casi constante (9 MB) y el uso del CPU se mantuvo acotado cerca de 22 %. La inspección manual del código permitió identificar las malas prácticas incurridas y corregir los componentes deficientes. Particularmente se implementó un mecanismo de sincronización "barrera cíclica" para lograr que el proveedor de la FFT marcara el paso de consumo del evaluador de sonidos cucú.




(a)




(b)

Figura 5: Consumo de memoria y CPU para la interfaz (5a) y el servicio emisión de alertas (5b).

### *IV-E. Accesibilidad y usabilidad*

La evaluación realizada facilitó la incorporación de descriptores de contenido en el *layout* de la interfaz gráfica para permitir a los usuarios no videntes activar el lector de pantalla *Talkback* (incluido en Android 4.4+). Se mejoró el contraste en la interfaz para asegurar un radio de 4.5:1 (aconsejado según el WCAG 2.0 nivel AA). También se agregaron patrones vibratorios en los siguientes escenarios: activar o desactivar el servicio (el botón vibra cuando se presiona), emisión de una alerta de cruce (se realizan dos vibraciones repetitivas, una larga seguida de una corta) y cuando el servicio se haya encendido (se da una vibración leve cada segundo).

## V. Investigación futura

La cantidad de dispositivos que admiten el sistema operativo Android es muy amplia y también lo son los recursos ofrecidos por cada fabricante. Esto impide fijar métricas de consumo de potencia, capacidad del CPU y memoria disponible. Sería deseable evitar este problema empleando *hardware* hecho a la medida, como un ASIC (*Application Specific Integrated Circuit*) que ofrezca tiempos de respuesta inmediatos (*hard-real-time*) y permita usar operaciones computacionalmente más complejas [8]. Este enfoque se usó en un proyecto de reconocimiento de patrones de disparos y motosierras en una red inalámbrica de sensores [2]. También a corto plazo sería conveniente usar los aceleradores de *hardware* provistos por *Renderscript* y compatibles entre dispositivos.

## VI. Conclusiones

Los resultados obtenidos mostraron que la implementación del reconocedor de sonidos cucú tiene buenas tasas de acierto y rendimiento, pero pueden mejorar. Se determinó que el sistema implementó adecuadamente el *kernel* diseñando y que las alertas emitidas están dentro del rango tolerable de picos de actividad para 1.1 dB SNR. Asimismo se determinó que a un nivel de intensidad sonora de 90 dB (usual en las calles) y a una distancia de 2 carriles, los semáforos accesibles de sonido cucú pueden ser detectados por debajo de los 5 picos, pero para calles de 5 carriles se requieren en promedio 8 picos. Se aseguró que el sistema fuera usable y accesible (satisfaciendo el principio POUR). Por último, se observó que, aunque el consumo de memoria fue constante en el dispositivo usado, sería deseable emplear *hardware* a la medida para asegurar tiempos de respuesta inmediatos y deterministas.

## Agradecimientos

Agradecemos a Mario Monge y Sharon Bejarano por proponer la idea de detectar los sonidos producidos por los semáforos accesibles y por aportar parte de las grabaciones usadas en esta investigación.

## Referencias

[1] Ahmetovic D., "Zebralocalizer: identification and localization of pedestrian crossings," Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services, ACM, pp. 275–284, 2018.

[2] Alvarado, P. y Chacón, A., "Sistema electrónico integrado en chip (SoC) para el reconocimiento de patrones de disparos y motosierras en una red inalámbrica de sensores para la protección ambiental," Instituto Tecnológico de Costa Rica, 2014, URL: http://hdl.handle.net/2238/3354 (consultado por última vez el 08/06/17).

[3] Asolkar P., Gajre S., Joshi Y. y Das A., "Simulation of colored and non-Gaussian wind noise for tropical shallow waters," Oceans 2016 MTS/IEEE Monterey, Monterey, CA, 2016, pp. 1-5, doi: 10.1109/OCEANS.2016.7761075.

[4] Briggs, M. y Zarkesh-Ha, P., "Evaluating mobile SOCs as an energy efficient DSP platform.", International System on Chip Conference, pp. 293–298, 2014, doi: http://doi.org/10.1109/SOCC.2014.6948943

[5] Brown K. C. et al., "Gait Speed and Variability for Usual Pace and Pedestrian Crossing Conditions in Older Adults Using the GAITRite Walkway", Gerontology and Geriatric Medicine, vol. 1, 2333721415618858, 2015.

[6] Ivanchenko V., "Crosswatch: a camera phone system for orienting visually impaired pedestrians at traffic intersections, "International Conference on Computers for Handicapped Persons, Springer, pp. 1122–1128, 2008.

[7] Monge T. y Solís Y., "El síndrome de caídas en personas adultos mayores y su relación con la velocidad de la marcha," Revista médica de Costa Rica y centroamérica, LXXIII (618) 91-95, 2016, URL: http://www.binasss.sa.cr/revistas/rmcc/618/art18.pdf (consultado por última vez el 21/06/17).

[8] Perneel L. , Fayyad-Kazan H. y Timmerman M.. "Can Android be used for real-time purposes?," 2012 International Conference on Computer Systems and Industrial Informatics, ICCSII 2012, doi: https://doi.org/10.1109/ICCSII.2012.6454350.

[9] Instituto Nacional de Vivienda y Urbanismo. "Reglamento de espacios públicos, vialidad y transporte," Sistema Costarricense de Información Jurídica, 2005, URL: http://www.pgrweb.go.cr/DOCS/NORMAS/1/VIGENTE/RM/2000-2009/2000-2004/2003/C6E1/67C8B.HTML (consultado por última vez el 22/06/17).