ADQUISIDOR INALÁMBRICO DE BIOPOTENCIALES CON INTERFAZ WEB

MADOU Rocío, GUERRERO Federico N., SPINELLI Enrique M.

Instituto de Investigaciones en Electrónica Control y Procesamiento de Señales LEICI, calle 48 y 116 S/N, La Plata (1900), Buenos Aires, Argentina.
rocio.madou@hotmail.com

INTRODUCCIÓN

La instrumentación electrónica moderna busca maximizar su potencial transmitiendo las señales medidas a computadoras para aprovechar su poder de procesamiento. Los métodos para lograrlo parten, históricamente, desde el uso de hardware específico como placas de expansión, pasando por buses estándar, buscando ganar simplicidad de uso, versatilidad y compatibilidad. Hoy, muchos dispositivos y servicios comerciales se despliegan sobre plataformas web, que aprovechan la ubicuidad de los navegadores web para funcionar sobre prácticamente cualquier plataforma sin necesidad de instalar ni adaptar componentes de hardware o software. La instrumentación biomédica posiblemente siga también este camino, en especial, para los usuarios de equipos orientados a la medición de señales electrofisiológicas; la simplicidad de uso puede habilitar incluso al público general a acceder a información médica valiosa, como se evidencia en el paradigma de la telemedicina.
En este trabajo se presenta el desarrollo de una plataforma de instrumentación inalámbrica que permite acceder a los datos a través de un navegador web. El trabajo se enfoca en el desarrollo de software que permite capturar en tiempo real señales obtenidas a partir de un módulo de adquisición de biopotenciales sobre el ordenador de placa reducida Raspberry Pi Zero W.
La plataforma es adaptable a distintos módulos de medición. Para lograr esta finalidad se presentan dos etapas en el software del equipo: la adquisición y almacenamiento de las señales, desacoplados de la recuperación, procesamiento e interfaz de usuario web.
Se muestra el funcionamiento del sistema completo a través de una señal simulada con el objetivo de corroborar la comunicación al servidor y pruebas experimentales verificando la integridad de transmisión de datos compatibles con señales de alta calidad de electrocardiografía.

PARTE EXPERIMENTAL

El adquisidor inalámbrico se basa en un dispositivo de medida previamente desarrollado[1], a su vez basado en un circuito integrado que incluye 8 canales de amplificación y conversión analógico/digital (ADS1299, Texas Instruments).
Asimismo, se utiliza la computadora de placa reducida Raspberry Pi Zero W que consiste en un procesador Broadcom BCM2835 que trabaja a 1GHz junto a una memoria RAM de 512 MB y una ranura de micro SD, donde se puede colocar una tarjeta que contenga un sistema operativo. Esta computadora cuenta con suficiente poder de cómputo para implementar un servidor web[2], y presenta una conexión inalámbrica WiFi, además de periféricos de entrada y salida digital de propósito general y dos buses SPI, lo cual permite interactuar en bajo nivel con hardware específico, como la placa de adquisición.
En la Figura 1 se detalla un diagrama en bloques del sistema implementado, y a continuación se describe la implementación de cada uno de los bloques.

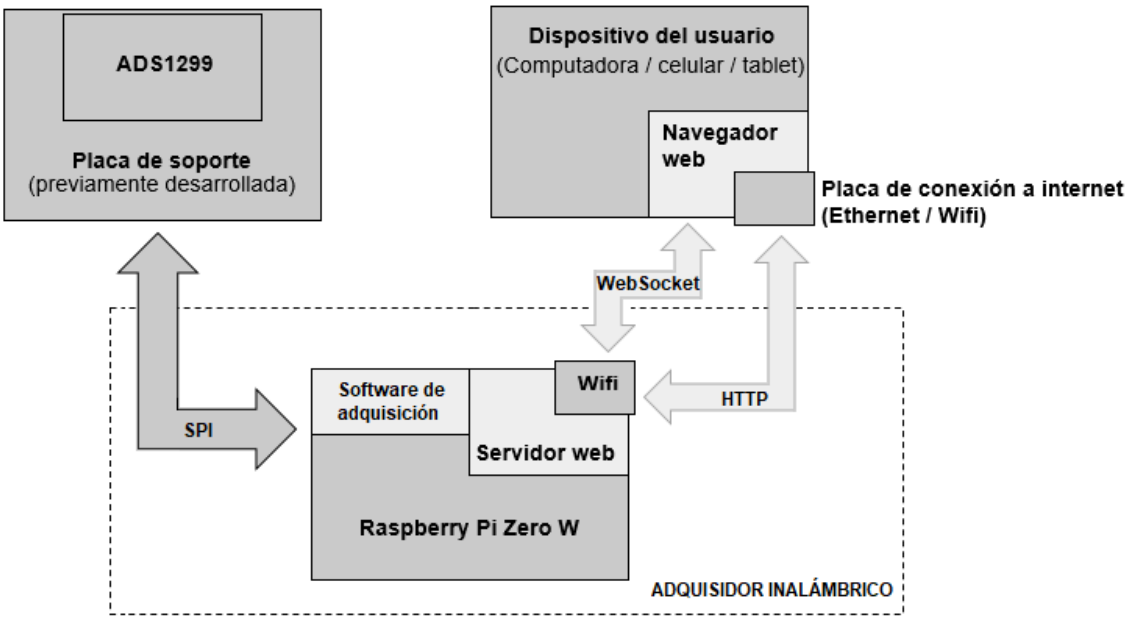

Figura 1. Diagrama de bloques del adquisidor inalámbrico.

La Raspberry corre el sistema operativo Raspbian, recomendado por el fabricante por estar optimizado para su hardware, el cual se basa en Debian, distribución de GNU/Linux.
El objetivo del servidor que correrá sobre este sistema operativo es otorgar al usuario la posibilidad de iniciar y finalizar la adquisición, la configuración de parámetros como la selección de canal, y la presentación de lo obtenido en un gráfico interactivo refrescado en tiempo real. En las siguientes subsecciones se describen los bloques implementados.

Etapa de adquisición

La placa que compone la etapa de adquisición es capaz de medir señales de biopotencial de distintos tipos como electrocardiograma (ECG), electromiograma (EMG), y electroencefalograma (EEG). Para ello cuenta con 8 canales de medida diferenciales de alta resolución (utiliza convertidores Sigma-Delta de 24 bits) con una tensión de ruido referida a la entrada menor a 1 µVrms, una tasa de muestreo de 2000 sps y un ancho de banda de 450 Hz.
Se realiza la conexión del mismo con la Raspberry a través de un puerto serie con el protocolo SPI. Para lograr una comunicación efectiva entre las partes, se desarrolla un programa en C encargado trasmitir por SPI la secuencia de configuración necesaria para el ADS1299 y la atención de las interrupciones generadas por el convertidor al haber muestras disponibles. Para ello se utilizan las librerías *wiringPI y wiringPiSPI*. Este programa corre en un proceso independiente y debe ser capaz de transmitir los datos sensados al servidor u otros procesos, para lo cual se implementaron dos *pipes*, tuberías de comunicación unidireccional de orden FIFO, uno de comandos y otro de datos, utilizando las librerías *unistd y sys/types.* Su utilización será mostrada en la sección “Resultados”.

Servidor Web

Django es un framework web de alto nivel que permite el desarrollo de sitios web rápidos, seguros y mantenibles, desarrollado en Python. Basados en el hecho que la Raspberry posee gran soporte en el lenguaje Python, sabiendo que el mismo posee variedad de librerías desarrolladas por lo que es adaptable a proyectos de alto y bajo nivel, y la tecnología a utilizar es de fácil modificación para proyectos escalables, como es este caso, se estableció como mejor opción para el nuevo servidor.

El modelo de funcionamiento *Model-View-Controller (MVC)* se puede definir como la columna vertebral de Django; por lo que será explicado a continuación en la Figura 2.Cuando el usuario interactúa con la interfaz, el controlador recibe la notificación de la acción solicitada por la persona y gestiona el evento a través de un manejador (o *handler*), para luego acceder al modelo y actualizarlo. A continuación, una vez obtenidos los datos del modelo, delega a los objetos de la vista la tarea de desplegar la nueva interfaz, reflejando los cambios de la información solicitados previamente por el usuario. Finalmente, la pantalla espera nuevas interacciones, comenzando el ciclo nuevamente.

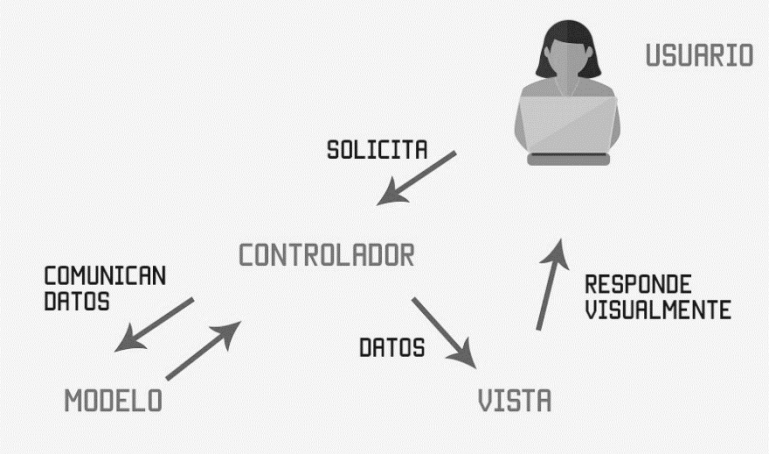

Figura 2. Modelo de funcionamiento MVC

Controlador

El controlador es el encargado de la lógica del servidor, por lo que previo al desarrollo de la comunicación entre las unidades del sistema es necesaria la creación y configuración del servidor como unidad independiente.

Si se desea acceder al servidor desde un dispositivo, se debe agregar la IP de la Raspberry como dirección propia del mismo, por lo que, al realizar solicitudes desde un browser, éste será capaz de reconocerlos y procesarlos. Esto se logra modificando el registro "ALLOWED_HOSTS" dentro del archivo *settings.py*.

Un servidor Django puede realizar la administración de una cantidad limitada de aplicaciones, una característica que permite la escalabilidad del sistema. Así, al comenzar el desarrollo del mismo es necesaria la creación de la aplicación particular del adquisidor. Se procede a incluirla dentro de las aplicaciones conocidas por el servidor modificando la lista llamada "INSTALLED_APPS" también en *settings.py*, además de incorporar dentro del archivo *urls.py* una línea particular que referencia a la aplicación misma.

A partir de este momento, toda configuración se realiza dentro de la carpeta de la aplicación. Se definen las rutas a las que responde el servidor en *urls.py* y su comportamiento asociado, detallado en *views.py*, archivo que, en caso de ser necesario, también las relaciona con una vista.

Modelo

En este caso, como se explicará en la sección de transmisión de datos, los resultados sensados no son almacenados en una base de datos, sino que son reflejados a medida que se adquieren; es por ello que no es necesaria la creación de modelos para la administración de la aplicación.

Vistas

Finalmente, estableciendo como necesidad el diseño de una interfaz amigable al usuario, para que sea intuitiva y fácil de utilizar, se desarrollan vistas basadas en botones y

formularios. En el caso del *template* principal, éstos se utilizan para la adaptación de los valores de configuración, asistiendo al elemento principal de la página: un monitor, es decir, un gráfico en tiempo real que refleja los datos sensados por el sistema adquisidor.
El desarrollo de las vistas es producto de la utilización de Bootstrap, un framework basado en la disponibilidad de diseños de componentes con posibilidad de personalizar el diseño de la página a gusto, incluyendo tipografías, formularios, botones, menús de navegación, entre otros elementos basados en HTML y CSS; y JavaScript (JS), específicamente la librería Canvas, la cual permite adaptar dinámicamente sus elementos de referencia en base a los valores recibidos, como por ejemplo el dibujo continuo de la señal dentro de los límites del monitor.

Transmisión y presentación de datos al cliente

Como fue mencionado anteriormente, los datos no se almacenan en una base de datos o archivo, por lo que es necesaria una comunicación continua e ininterrumpida para evitar la pérdida de los mismos. Por lo tanto, se debe proveer un canal de datos de tiempo real para atender la solicitud de datos del usuario y la representación de los mismos en pantalla.
Buscando una comunicación continua con los componentes de la vista, sin necesidad de recargar la página o bloquear al servidor por la transmisión de datos, es necesaria una vía de transferencia paralela al servidor Django. La opción más utilizada para situaciones con objetivos similares es *WebSockets(WS)*, la cual posee gran desarrollo y soporte en JS. La problemática surge al momento de combinarlo con Django, ya que este no posee soporte para dicha herramienta. La alternativa presente para el servidor desarrollado es *Channels*, cuyo modo de funcionamiento, luego de la configuración de una serie de parámetros, permite la interacción con la librería WS.

RESULTADOS Y DISCUSIÓN

A partir del funcionamiento de los componentes descriptos, se logró el objetivo propio del trabajo: la adquisición de datos del ADS1299, el envío de los mismos hasta la vista y su graficación.
En la Figura 3 se explicita el camino de datos logrado para la entrega de datos al cliente.

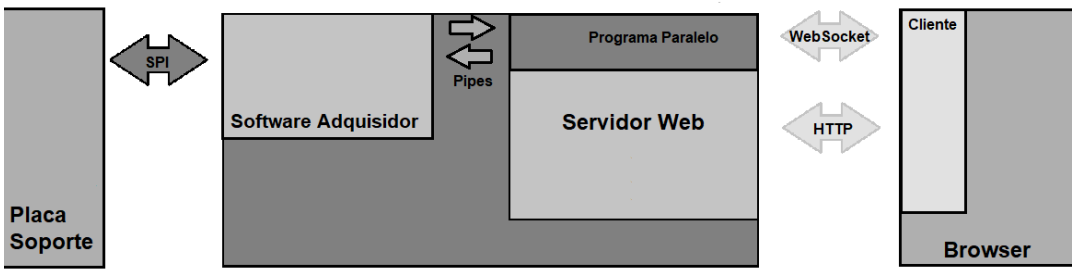

Figura 3. Camino de datos desde el adquisidor hasta el cliente

Al ingresar a la dirección URL correspondiente al monitor de datos, se crea la instancia del servidor desarrollado con Channels, el cual queda pendiente a la petición de conexión por parte de cliente. Como respuesta del servidor Django, se renderiza la vista que se presenta en la Figura 4, que al mostrarse en el navegador ejecuta un script realizando la conexión al servidor Websocket. De esta forma, el canal de comunicación estará a la espera del envío de un mensaje.

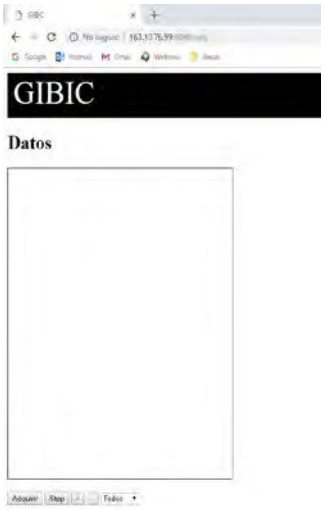

Figura 4. Monitor de datos de la interfaz web.

Como se observa en la imagen, debajo de los límites del gráfico, ahora vacío, se encuentran los botones encargados del inicio y fin de la adquisición. En el momento en que el usuario presiona el primero de ellos, se envía un comando de inicio de adquisición a través de WS; esto crea un hilo paralelo ejecutando el script encargado de la solicitud de datos al adquisidor, solicitando la conexión a los pipes de comandos y datos para el inicio de la recepción de muestras. Cada dato recibido por el adquisidor, en este momento trabajando a una tasa de 250muestras por segundo, y 25 paquetes por segundo, es reenviado al contenido del HTML para su guardado, filtración e impresión.
Esto continúa hasta que el usuario elige detener la adquisición a través del botón o cierra la pantalla del monitor. En el primer caso, se envía un comando de finalización a través de WS, la cual es retransmitida por el pipe, deteniendo la adquisición, para luego cerrar las conexiones con el programa en C y, finalmente, terminar el hilo encargado de la retransmisión de datos. Sucedido esto, se eliminan de la vista los datos sensados, dándole la posibilidad al usuario de descargar el archivo encargado de almacenarlos desde el inicio del proceso.
La tasa de paquetes lograda de 25 paquetes por segundo fue suficiente para mostrar una señal responsiva al usuario, suficiente para evaluar el funcionamiento del equipo en tiempo real. Si bien hasta el momento se logró adquirir una tasa de sólo 250 muestras por segundo. Para transmitir las señales de más ancho de banda de biopotencial como el EMG sería deseable extender el muestreo a 2000 sps, lo que significa enviar más muestras dentro de cada paquete. La limitación actual para lograrlo se basa en la performance de los algoritmos implementados en Python.

CONCLUSIONES

Se desarrolló un servidor basado en Django y su extensión Channels que corrió exitosamente sobre la computadora de placa reducida Raspberry Pi Zero W, logrando recuperar muestras a través de pipes de otro proceso en el que un programa en C se encarga de la comunicación de bajo nivel con un convertidor analógico digital para medidas de biopotenciales ADS1299.
El servidor permitió a un usuario utilizar el adquisidor a través de una interfaz web. La misma corre sobre cualquier navegador y permite comenzar y detener la adquisición, visualizar las señales y descargar un archivo de datos con las mismas.
La comunicación de datos en tiempo real se logró utilizando la tecnología WebSocket. Como trabajo futuro resta optimizar la performance de las distintas etapas para lograr aumentar la tasa de muestreo y realizar procesamientos especiales pedidos por el cliente, como detección de envolvente en EMG.

BIBLIOGRAFÍA

[1] Guerrero, F. N., & Spinelli, E. (2015). Surface EMG multichannel measurements using active, dry branched electrodes. In VI Latin American Congress on Biomedical Engineering CLAIB 2014, Paraná, Argentina 29, 30 & 31 October 2014 (pp. 1-4). Springer, Cham.

[2] Vujović, V., & Maksimović, M. (2015). Raspberry Pi as a Sensor Web node for home automation. Computers & Electrical Engineering, 44, 153-171.

[3] Kelly, G. (2014). Development of a compact, low-cost wireless device for biopotential acquisition.