UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Escuela técnica superior ingenieros industriales Valencia

Universitat Politècnica de València

Desarrollo de un sistema de monitorización de parámetros biomédicos basado en ROS sobre una plataforma empotrada

TRABAJO FIN DE GRADO

Grado en Ingeniería en Tecnologías Industriales

Autor: Ernesto Oliver Chiva

Tutor: Marina Vallés Miquel

Cotutor: Ángel Valera Fernández

6 de julio de 2015

Resumen A lo largo de este proyecto se ha desarrollado un sistema modular de sensores biomédicos para dotar a cualquier robot con ellos. Para llevar a cabo el desarrollo se ha tenido que profundizar en conceptos de informática al utilizar la tarjeta embebida *RaspberryPi* con el entorno de trabajo ROS y el conversor ADC LTC2309. Por otro lado ha sido necesario conocer como funcionan los sensores biomédicos utilizados, así como la señal de salida esperada a la que se le han tenido que aplicar diferentes tratamientos como filtros digitales, media móvil, derivada y detección de patrones.

El sistema desarrollado se ha conectado a un robot paralelo diseñado para la rehabilitación de tobillos, en donde los sensores envían información a dos módulos, uno de control y otro de alarma. El módulo de control gradúa la intensidad del robot dependiendo de parámetros fisiológicos del paciente mientras que el módulo de alarma detecta situaciones de riesgo que detienen al robot inmediatamente.

Palabras clave: Sistemas empotrados, adquisición de datos, sensores biomédicos, ROS, robots, RaspberryPi.

Resum Al llarg d'aquest projecte s'ha desenvolupat un sistema modular de sensors biomèdics per dotar a qualsevol robot amb ells. Per dur a terme el desenvolupament ha sigut necessari profunditzar en conceptes d'informàtica al utilitzar la tarjeta embeguda *RaspberryPi* amb l'entorn de treball ROS y el convertidor ADC LTC2309. Per una altra banda ha calgut conéixer com funcionen els sensor biomèdics utilitzats, axina com la senyal d'eixida esperada a la qual ha fet falta aplicar-li diferents tractaments com filtres digitals, mitja mòbil, derivada i detecció de patrons.

El sistema desenvolupat s'ha conectat a un robot paral·lel dissenyat per a la rehabilitació de turmells,on els sensors envien la informació a dos mòduls, un de control i un altre d'alarma. El mòdul de control gradua la intensitat del robot depenent de paràmetres fisiològics del pacient mentre que el mòdul de l'alarma detecta situacions de risc que detenen el robot immediatament.

Paraules clau: Sitema encastat, adquisició de dades, sensors biomèdics, robots, RaspberryPi.

Abstract Along this project a modular system with biomedical sensors was developed in order to provide any robot with them. Computer knowledge was needed due to the use of the embedded card *RaspberryPi* with the framework ROS and the AD convertes LTC2309. On the other hand it has been needed to learn about the biomedical sensor used, as well as the expected output signal which has been processed with different treatments like digital filters, moving average, differentiating and pattern detection.

The develped system has been connected to a parallel robot designed with the purpose of ankle rehabilitation,in which the sensors send the information to two modules, one of them for controll and the other for alarm. The controll module sets the intensity of the robot depending on the patient's physiological parameters whereas the alarm module detects risk situations that make the robot stop immediately.

Keywords: Embedded system, data acquisition, biomedical sensors, robots, RaspberryPi.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción, mototivación y objetivos.

1.1 Introducción

La robótica es una rama de la tecnología que abarca muchas, y muy diversas, disciplinas. Lo más sencillo es pensar que la robótica es una mezcla de electrónica e informática combinadas para crear algún tipo de mecanismo dotado de movimiento gracias a sensores y actuadores guiados por programas que le proporcionan la funcionalidad para una tarea dada. Pero a medida que ha ido desarrollándose este campo también ha ido evolucionando la conciencia acerca del alcance de la robótica, un enfoque interesante a esta idea lo podemos encontrar haciendo un recorrido a la historia de la ciencia ficción.

Aunque no se trata rigurosamente de un robot, la obra publicada por *Mary Shelley* en 1818 ya refleja la idea de crear un ser vivo a partir de *piezas* cuando el Dr. Frankenstein logra dar vida a su criatura. En 1921 el escritor checo Karel Čapek acuña el término *Robot* a partir de la palabra checa *Robota* que se traduce como *servidumbre* [2] y en 1927 se estrena la película *Metrópolis*, en la que se juega con la idea de un robot capaz de imitar a la perfección a cualquier humano. Si avanzamos hasta mitad del siglo XX donde aparecen las obras de *Isaac Asimov*, vemos que este relaciona la psicología y la robótica. Los robots imaginados por *Asimov* poseen un cerebro positrónico que los dota de una inteligencia artificial tan avanzada que se les tiene que imponer las conocidas leyes de la robótica:

- Primera ley: Un robot no hará daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.

- Segunda ley: Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la 1ª Ley.
- Tercera ley: Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la 1ª o la 2ª Ley.

En los años ochenta se popularizó en el cine obras como *terminator* o *Blade Runner*, donde las máquinas dominan el mundo destacando el poder bélico que podían suponer los robots.

En cambio, la tendencia hoy en día es la idea es pensar en robots que más que sirvan, ayuden a la personas. Este es el caso del robot *Baymax* de la película de animación *Big Hero 6*, un robot enfermero capaz de conocer el estado médico de una persona usando diferentes sensores. En este aspecto, se está popularizando la combinación del campo de la medicina y la robótica.

Por lo tanto, es seguro que el futuro de la robótica pasa por adaptar cada vez ciencias más complejas que puedan resultar beneficiosas.

1.2 Motivación

Con este proyecto surgió de la necesidad de dotar de parámetros biomédicos a un robot paralelo diseñado con el objetivo de realizar ejercicios físicos a un usuario. El robot estaba diseñado para trabajar en modo *maestro-esclavo*, almacenando un primer ejercicio guiado por un especialista para poder repetirlo automáticamente. El problema era que este robot no recibía ningún tipo de retroalimentación más que la fuerza que estuviera ejerciendo el usuario por lo que se vio la necesidad de conectarle algún sensor capaz de registrar parámetros fisiológicos.

Desde un primer momento se descartaron los equipos médicos profesionales por su elevado precio, pero se vio que el kit *eHealth sensor platform* de *Cooking Hacks* podría servir. Este kit se adaptaba tanto a un *Arduino UNO* como a una *RaspberryPi Model B*, por lo que usando esta última se podía instalar el framework usado para el control del robot paralelo gracias a que este es compatible con el sistema operativo *Raspbian*.

Otra razón para empezar este proyecto fue el potencial que tienen los sensores biomédicos adaptados a la robótica, pudiendo reutilizar este trabajo en otros robots. Como por ejemplo enviar información del usuario a un coche autónomo y que este sea capaz de mejorar su conducción a partir de dicha información, o en un ambiente más hospitalario se podría reprogramar como un módulo que monitorice el estado del paciente y se encargue de avisar a los enfermeros en caso de que fuera necesario.

1.3 Objetivos

El objetivo principal de este trabajo ha sido crear un sistema modular de sensores biomédicos fácilmente adaptables a cualquier robot que use el framework ROS.

Esto se traduce en entender como trabajar con cada uno de los sensores utilizados para poder realizar la medición correctamente, frecuencias de muestreo necesarias, tipos de filtro y otros tratamientos, e interpretación de los resultados. Por otra parte es necesario conocer el funcionamiento de ROS que hará la función de entorno de trabajo, así como las diferencias de que esté instalado en un ordenador convencional con el sistema operativo Ubuntu e instalado en un ordenador embebido como es la RaspberryPi que trabaja con una arquitectura diferente.

CAPÍTULO 2

Desarrollo teórico

2.1 Procesado de señales

Toda la información que obtenemos por medio de los sentidos no es más que señales que recibimos del entorno; que nuestro cerebro procesa y transforma en mensajes. Cuando reconocemos la voz de un amigo o identificamos una canción, es nuestro cerebro el que de entre todos los sonidos que recibe sabe extraer la información que podría ser útil. Lo mismo ocurre con la vista, los ojos se limitan a transformar la luz en señales nerviosas que son procesadas por el cerebro que distingue las formas, los colores, en incluso distancias. Eso mismo es lo que se ha perseguido en el estudio del procesado de señales. Aunque podemos encontrar principios del procesado de señales en el análisis numérico del siglo 17, no fue hasta la década de los sesenta cuando empezaron a estar disponibles los primeros ordenadores digitales y empezaron a desarrollarse las primera aplicaciones que requerían el procesado de señales. Al principio y debido al elevado precio de los ordenadores el procesado de señales se limitó a cuatro áreas: En seguridad militar con *radares y sónars*, en la *exploración de hidrocarburos* por los beneficios económicos que podía suponer, en la *exploración espacial* donde las mediciones son limitadas y en *medicina* donde afecta directamente a la vida y salud de las personas [1].

Regresión

Normalmente cuando se toman datos de una señal se hace a una frecuencia constante, puede darse el caso de que el tiempo entre muestras no sea siempre el mismo. Normalmente esto sucede cuando el dispositivo encargado de leer las señales no está especialmente diseñado para eso y el proceso sufre interrupciones debido al sistema operativo, retardos de los conversores, tiempo de acceso a memoria, etc. Situaciones que se evitan con dispositivos que cuentan con microcontroladores específicamente diseñados para lecturas a tiempo real.

En el caso de tener una señal en la que la diferencia de tiempos entre cada muestra no sea constante, podemos recurrir a la *regresión* para, con una nube de k puntos, encontrar la función polinómica que siga la señal en un determinado intervalo centrada en un punto. En otras palabras, buscamos la función de la forma $f(x) = a_0 + a_1 x + a a_2 x^2 + ... + \alpha_n x^n$ que se adapte a un punto, y esto para cada uno de los puntos equidistantes que necesitemos. El grado de la regresión viene definido por n: una regresión lineal tendrá un $n = 1$, una regresión cuadrática tendrá $n = 2$,... En general es suficiente con una regresión lineal.

$$A = \begin{pmatrix} a_0 & a_1 & \cdots & a_n \end{pmatrix}; \qquad X = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & \cdots & x_k^n \end{pmatrix}; \qquad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix};$$

Este problema se aborda mediante álgebra de matrices, buscando una solución(2.2) para A de la ecuación 2.1 para cada punto equidistante con la condición que se encuentren suficientes datos que cumplan $x_1 < t < x_k$ y $k > n$.

$$A \times X = Y \tag{2.1}$$

$$A = YX^T(XX^T)^{-1} \tag{2.2}$$

Filtrado

El procesado de señales es importante porque, aunque normalmente la información que esperamos obtener se encuentra en el dominio del tiempo, esta es complicada de leer debido al ruido o información no deseada que se superpone. Este problema se aborda con la *transformada de Fourier* 2.3a, que permite obtener una señal en el dominio de la frecuencia. Conocidas las frecuencias y las fases de estas es fácil eliminar o atenuar aquellas que no sean de interés y/o amplificar la banda de frecuencias que contengan la información, en otras palabras, filtrar. Una vez tratadas las frecuencias podemos volver al dominio del tiempo usando la *transformada inversa de Fourier* 2.3b.

$$F(\omega) = \int_{-\infty}^{\infty} f(x) \mathrm{e}^{-2\pi i \omega x} dx \tag{2.3a}$$

$$f(x) = \int_{-\infty}^{\infty} F(\omega) \mathrm{e}^{2\pi i \omega x} d\omega \tag{2.3b}$$

En la práctica podemos abordar el problema tanto con los *filtros analógicos*, que se componen principalmente de amplificadores operacionales, resistencias y condensadores; como con *filtros digitales* que, mediante circuitos digitales o directamente software, imitan los filtros analógicos.

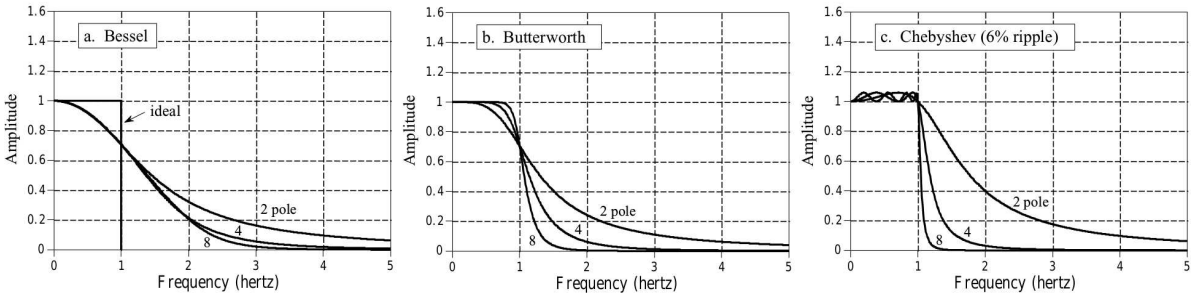


Figura 2.1: Respuesta en frecuencia de los tres filtros. Extraído de [1].

Normalmente nos vamos a encontrar con tres tipos de filtros: Bessel, Chebyshev y Butterworth. Cada uno presenta una particularidad y pueden ser más o menos restrictivos dependiendo del número de polos y ceros que tengan, ver figura 2.1.

Aunque existe infinidad de circuitos con los que crear filtros, un ejemplo sencillo es el circuito *Sallen-Key* 2.2. Con el circuito *Sallen-Key* es posible tener los tres tipos de filtros anteriores modificando únicamente los valores de los componentes, ver tabla 2.1. En la misma tabla aparecen también los parámetros para diferentes etapas, esto es porque aunque un único circuito *Sallen-Key* es un filtro con dos polos, estos se pueden diseñar en cascada añadiendo dos polos adicionales por cada circuito extra.

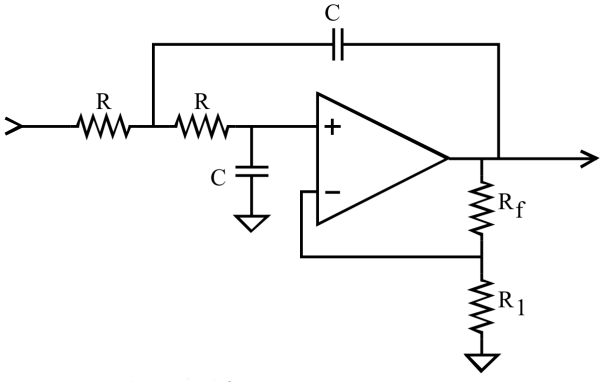


Figura 2.2: Ejemplo de circuito Sallen-Key.

Tabla 2.1: Parámetros para el diseño de filtros. Extraído de [1]

		Bessel		Butterworth		Chebyshev	
#polos		**k1**	**k2**	**k1**	**k2**	**k1**	**k2**
2	etapa1	0.1251	0.268	0.1592	0.586	0.1293	0.842
4	etapa1	0.1111	0.084	0.1592	0.152	0.2666	0.582
	etapa2	0.0991	0.759	0.1592	1.235	0.1544	1.660
6	etapa1	0.0990	0.040	0.1592	0.068	0.4019	0.537
	etapa2	0.0941	0.364	0.1592	0.586	0.2072	1.448
	etapa3	0.0834	1.023	0.1592	1.483	0.1574	1.846

Son estos mismos circuitos analógicos los que se busca imitar con los filtros digitales mediante el uso de circuitos digitales o directamente por software. Los filtros digitales trabajan con un conjunto de datos discretos por lo que la ecuación 2.3a para funciones continuas deja de ser válida. Para tratar señales discretas en el tiempo y en un dominio finito recurrimos a la *transformada discreta de Fourier* 2.4a, una adaptación de la *transformada de Fourier*.

$$X_k = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-\frac{2\pi i}{N}kn} \qquad k=0,\cdots,N-1 \tag{2.4a}$$

$$x_n = \sum_{n=0}^{N-1} X_k \mathrm{e}^{\frac{2\pi i}{N}kn} \qquad n=0,\cdots,N-1 \tag{2.4b}$$

Aunque es posible una aplicación directa de 2.4a, esto supondría el cálculo de N^2 operaciones, siendo N el número de datos discretos. Para optimizar este cálculo normalmente se recurre a la *transformada rápida de Fourier*. El algoritmo de la *FFT* se basa en el concepto de *divide y vencerás* (ver figura 2.3), se trata de ir dividiendo las muestras en posiciones pares e impares hasta llegar al punto de tener conjuntos de dos datos a los que se les realiza la *DFT* (2.4a).

Este procedimiento se usa principalmente por su simplicidad y porque reduce la complejidad numérica de N^2 a $N\log_2(N)$ [1], por otra parte hay que tener en cuenta es que N debe ser una potencia entera de dos para poder optimizar el algoritmo.

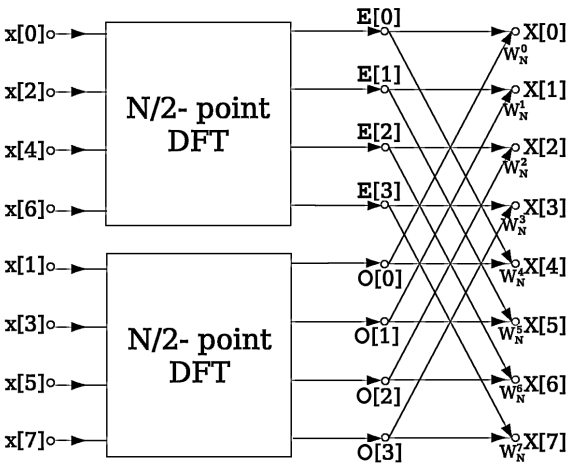


Figura 2.3: Ejemplo del flujo de datos en una transformada rápida de Fourier, extraído de [1]

El lector podría pensar que si la *transformada discreta de Fourier* devuelve el dominio de la frecuencia se podría hacer un filtro ideal que eliminara por completo las frecuencias no deseadas e hiciera la *IDFT* únicamente de las frecuencias que necesitamos. Esto no es conveniente debido a que las frecuencias que tenemos, a igual que los tiempos, son discretas y por lo tanto la frecuencia de corte no suele ser exacta. Otro motivo es que un corte limpio de frecuencias genera ruido al pasarlo al dominio del tiempo. La *transformada de Fourier* genera una serie de senos y cosenos que se suman y restan entre sí para formar la señal, si cortamos a partir de un punto en vez de ir atenuándolos no se compensan y por lo tanto es un filtro menos eficaz que los filtros de Butterworth, Chebyshev o Bessel.

Media móvil

Existen otros filtros digitales que no están basado en la *DFT*, ese es caso de la *media móvil*. Los filtros basados en la media sirven para suavizar la señal, siendo especialmente útiles para reducir el ruido aleatorio que pudiera existir [1].

La versión más simple de este filtro consiste en dividir la señal de n en conjuntos de k valores para luego hacer la media aritmética de cada conjunto. El principal problema que presenta este método es que el número total de datos después del suavizado es igual a $\frac{n}{k}$, lo mismo ocurre con la frecuencia de muestreo.

$$X_i = \frac{1}{n} \sum_{j=1}^{n} x \tag{2.5}$$

La versión más simple de la media móvil que podemos usar y que no necesita eliminar tantos valores de la señal es parecida a la anterior. Se trata de hacer la media con los datos de una banda centrada en cada valor de la señal original, esto es, tenemos que quitar un valor y añadir el siguiente para cada punto.

$$X_i = \frac{1}{k} \sum_{j=i-\frac{k}{2}}^{j=i+\frac{k}{2}} x_j \tag{2.6}$$

Estos tipos de filtros no solo usan la media aritmética, es común encontrar casos en los que merece la pena el uso de medias ponderadas para darle diferente peso a los valores que se encuentran por delante del centro de la banda de los que se encuentran por detrás.

$$X_i = \frac{\alpha_1 x_{i-\frac{k}{2}} + \alpha_2 x_{i-\frac{k}{2}+1} + ... + \alpha_{\frac{k}{2}} x_{\frac{k}{2}+i} + ... + \alpha_{k-1} x_{i+\frac{k}{2}-1} + \alpha_k x_{i+\frac{k}{2}}}{k} \tag{2.7}$$

Otras operaciones

Hay ocasiones en la que nos puede interesar destacar comportamientos concretos de las señal para facilitar la detección de situaciones concretas. Por ejemplo podemos estar más interesados en conocer el grado de variación y en ese caso podemos destacar las fuertes subidas y bajadas derivando la señal. O si tenemos valores negativos y positivos pero la información que necesitamos es el módulo podemos elevarla al cuadrado.

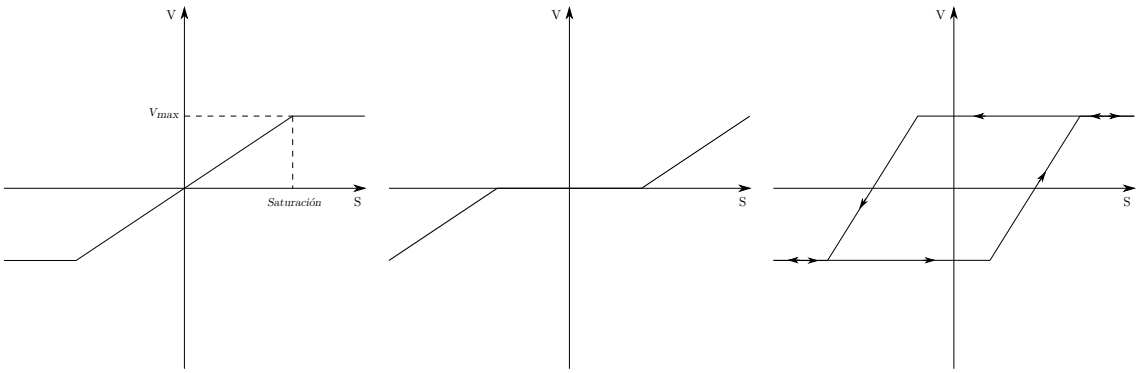
2.2 Sensores

Como se ha mencionado en el punto anterior, los seres humanos disponen de un sistema sensorial enormemente desarrollado. Las tareas cotidianas suponen el uso de muchísima información obtenido por diferentes vías que es captada por los sentidos sin apenas percatarnos. Los robots también necesitan recibir información tanto del entorno como interna, pueden necesitar saber que obstáculos se encuentran a su alrededor o en que posición están sus ejes, por ejemplo. Si bien los sensores artificiales no pueden competir con los sentidos de los seres vivos en complejidad, existen en el mercado sensores capaces de tomar lecturas prácticamente de cualquier magnitud física haciendo de los robots herramientas increíblemente versátiles.

2.2.1. Definición de sensor.

Siguiendo la definición de [2]: Un sensor es un dispositivo eléctrico y/o mecánico que transforma magnitudes físicas a valores medibles de dicha magnitud. Cuando hablamos de robótica o electrónica digital entendemos que un sensor va a transformar una magnitud física en un nivel de tensión proporcional a dicha magnitud para que pueda ser transformado a un valor digital usando un conversor A/D, y así ser tratable por un procesador.
Para caracterizar un sensor se usan lo que se llaman *descriptores*, estos pueden ser estáticos si definen el comportamiento en regimen permanente o descriptores dinámicos si hacen referencia a la evolución temporal de la señal de salida en referencia a excitaciones en la señal de entrada [2].

- Descriptores estáticos
  - **Rango:** Indíca los valores máximos y mínimos en los que puede trabajar el sensor. Normalmente indica el rango de valores de la magnitud física que puede transformar en señal eléctrica.
  - **Exactitud:** La lectura de un sensor se distribuye estadísticamente alrededor del valor real; así, se define la exactitud como la desviación de la lectura de un sistema de medida respecto a una entrada conocida. Se define como el mayor error esperado entre las señales medida e ideal. Suele expresarse como un valor porcentual respecto al valor máximo del sistema de medida o como un rango máximo de error. Por ejemplo un sistema que pretenda medir el desplazamiento en un rango de 0 a $50cm$ podría presentar, por ejemplo, un error de $\pm 5\,\%$ o lo que es lo mismo, un error de $\pm 2,5cm$.



(a) Saturación. (b) Zona muerta. (c) Histéresis.

Figura 2.4: Distintas no linealidades en la respuesta de un sensor.

- **Repetitividad:** Relacionado con la exactitud, la repetitividad se define como la capacidad de reproducir una misma lectura con una precisión dada.
- **Reproducibilidad:** Tiene el mismo significado que la repetitividad excepto que las medidas se realizan bajo condiciones diferentes.
- **Resolución:** La resolución se define como el valor de incremente más pequeño que se puede detectar. La resolución se calcula dividiendo el rango entre el número de valores que puede tomar. Por ejemplo, un sensor que trabaje a $5V$ y $10bits$ tiene una resolución de $\frac{5}{(2^{10}-1)} = 0{,}00489V$.
- **Error:** Se trata de la diferencia entre el valor medido por el sensor y el valor real.
- **No linealidades**: Se considera que un sensor es lineal si su respuesta sigue la ecuación de la recta $y = mx + n$. Aunque muchas vece se supone la linealidad, esto no siempre es cierto y se producen errores llamados de linealidad.
  Dentro de los errores de lienalidad cabe destacar los siguientes (fig:2.4):
  1. Saturación:Efecto producido en determinados dispositivos en los que, una vez pasado un determinado valor de entrada, la salida deja de aumentar linealmente y se mantiene constante en el valor de saturación (2.4(a)).
  2. Zona muerta o dead zone: Rango de entradas en las que el sensor deja de tener funcionalidad, en esta zona la entrada no es posible determinarla con la salida del sistema de medida (2.4(b)).
  3. Histéresis: Esta no linealidad se caracteriza por devolver medidas diferentes dependiendo del sentido de la señal, si el valor de esta aumenta o disminuye seguirá una función u otra (2.4(c)).

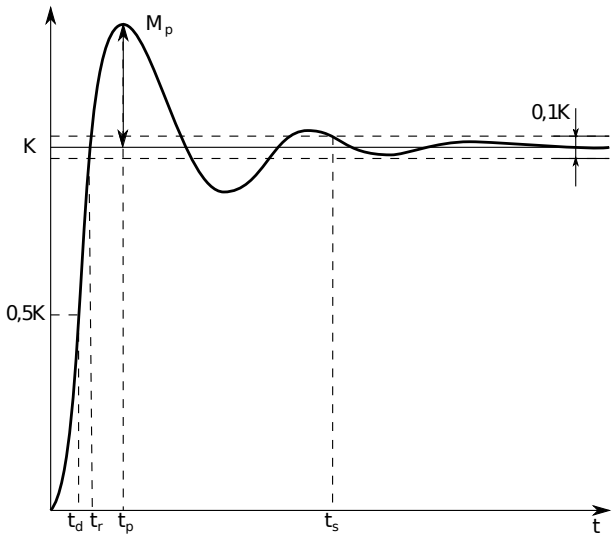


Figura 2.5: Descriptores dinámicos en la respuesta de un sensor.

- **Sensibilidad:** Es la razón del cambio de la salida frente a cambios en la entrada. Suele medir las variaciones de voltaje frente a variaciones de la entrada $S = \frac{\partial V}{\partial x}$, en otras palabra, la sensibilidad de un sensor es la medida de hasta qué grado la señal eléctrica de salida cambia a la vez que las cantidades de las magnitudes de entrada.
- **Excitación:** Cantidad de corriente o tensión necesarias para el funcionamiento des sensor.
- **Estabilidad:**Es una medida de la posibilidad de im determinado sensor de mostrar una misma salida en un rango en el que la entrada permanece constante.
- **Ruido:** La señal leída puede estar modificada por señales externas con información no deseada que se superpone a la información de interés. Esto hace que en muchas ocasiones la señal esperada difiera de la real y ocasione problemas en los procesadores que la interpretan.

- Descriptores dinámicos: Como se ha dicho anteriormente, éstos caracterizan la evolución temporal de la señal. A partir de la respuesta de un sensor frente a una entrada tipo escalón (fig: 2.5) se pueden definir los siguientes parámetros que la caracterizan:

  - El tiempo de retardo, t_d, indica el tiempo que le cuesta a la señal de salida alcanzar el 50 por ciento de su valor estacionario.
  - El tiempo de subida, t_r, se define como el tiempo que tarda la señal en estabilizarse en su valor final. Este parámetro proporciona la información acerca de lo rápido que responde ante una entrada, es decir, de la velocidad de este.
  - El tiempo de pico, t_p, es el tiempo que tarda la señal de salida en alcanzar el máximo ante un cambio.

- El pico de sobreoscilación, M_p, se expresa como la diferencia entre el valor máximo de la respuesta y el valor final. Dicho de otro modo, es el valor de la amplitud del primer pico y se suele expresar como un porcentaje del valor final.
- El tiempo de establecimiento, t_s, indica el tiempo en el que el valor de la señal de salida alcanza, sin volver a sobrepasar, una banda de $\pm 5\,\%$ el valor final o estacionario.

2.2.2. Factores de aplicación

Junto a los descriptores anteriores, en el *datasheet* generalmente también se incluyen otros factores ambientales que pueden alterar el funcionamiento del sensor como puede ser la humedad, temperatura, vibraciones, etcétera, así como el rango que el fabricante certifica que estos factores no estropearán el sensor.Debido a, entre otras cosas, estos factores es necesario realizar una calibración del sensor usando otro sensor con una respuesta estándar conocida con el fin de asegurar la medición, de esta manera se puede establecer la relación entre la variable medida por el sensor y su señal de salida.

Una vez está el sensor calibrado, y previamente a su utilización, se deben realizar los tests oportunos para asegurar el buen funcionamiento tanto del este como del resto de la instalación. Por lo tanto, es necesario comprobar que la respuesta ante determinados estímulos es la esperada y no está falseada debido a la circuitería, sistema mecánico, interferencias electromagnéticas o cualquier otro elemento que pudiera afectarle.

2.3 Robótica

Aunque asociamos la robótica con el futuro, la idea de crear mecanismos que imiten a las personas u otros seres vivos no es precisamente nueva. Los primeros mecanismos no tenían un fin productivo, su objetivo era más bien el de entretener, el de mostrar al público de que se era capaz gracias al ingenio.
Uno de los primeros ejemplos documentados lo encontramos en el siglo $IVa.C.$,se trataba de un modelo de paloma construido por *Archytas de Tarentum* capaz de moverse mediante un chorro de vapor. Más adelante, si avanzamos hasta el siglo $XIII$ encontramos la cabeza parlante de *Bacon* o el hombre de metal construido por *Albertus Magnus* [2]. No fue hasta la llegada de la revolución industrial que toda esta experiencia y habilidades adquiridas a lo largo de los años fueron aplicadas con un fin meramente productivo.

La revolución industrial supuso un gran salto para la robótica, sobre todo en el sector textil empezaron a desarrollarse mecanismos capaces de realizar las tareas repetitivas reduciendo el coste y aumentando la producción de manera significativa. Pero el antes y el después lo encontramos con el final de la segunda guerra mundial, con los avances tecnológicos en materia de electrónica y computación además de los sistemas de engranajes y la ciencia de materiales.
El primer robot como tal llega a partir del trabajo de *George Devol (1912-2011)*, quien fusiona los manipuladores mecánicos con técnicas de programación usadas para poder realizar tareas autómaticas sin necesidad de un control *maestro-esclavo* [2].

2.3.1. Clasificación

La gran variedad de robots que existen complica excesivamente la tarea de hacer una clasificación universal. Aquí se ha seguido la clasificación de [2], bastante aceptada entre la comunidad científica.

- Humanoide: Cualquier robot que busque imitar la apariencia y comportamiento humano. Es el más recurrido en las obras de ciencia ficción.
- Robot móvil: Robot instalado sobre una plataforma móvil.
  - Rodantes: Aquellos que usan ruedas para su desplazamiento. La disposición de estas es muy variada, la más común es la de 2 + 2 ruedas similar a los vehículos tradicionales donde dos ruedas tienen la tracción y las otras dos la dirección, otro ejemplo son los robots con un sistema oruga o incluso los que usan dos ruedas y un sistema de péndulo invertido. Estos son los más rápidos sobre superficies lisas.
  - Andante: Buscan imitar las piernas humanas o las patas de los animales. Aunque el control del movimiento implica complicados algoritmos estos tienen más posibilidades de movimiento, por ejemplo, un robot hexápodo es capaz de moverse por terrenos irregulares con facilidad que otro rodante.
  - Reptadores: Imitando el movimiento de los reptiles, estos robots compuestos deun elevado número de secciones son capaces de acceder a zonas que otros no podrían, son especialmente útiles en labores de rescate.
  - Nadadores: Diseñados para trabajar en un medio acuático estos se utilizan para acceder a zonas que la presión hace inaccesible de otro modo.
  - Voladores: También conocidos como *drones*, estos robots están en auge en el entorno civil debido a que se ha reducido su precio considerablemente y las posibilidades de captar imágenes aereas fácilmente. El desarrollo en el entorno militar se debe a su alta maniobrabilidad y que debido a su tamaño son complicados de detectar.

- Robot industrial: Robots manipuladores programados para realizar tareas repetitivas pudiendo usar diferentes herramientas. Los robots industriales se pueden clasificar siguiendo tres criterios diferentes: numero de ejes, tipo de control y estructura mecánica.

  1. Clasificación según el número de ejes o grados de libertad:
     - Robots con 3 ejes.
     - Robots con 4 ejes.
     - Robots con 5 ejes o más.
  2. Clasificación según el tipo de control:
     - Secuencia-controlada: Los movimientos que realiza siguen un orden determinado.
     - Trayectoria-operada/continua: Robot en el que tres ejes o más operan de acuerdo a las especificaciones de trayectoria requerida para alcanzar la posición deseada, normalmente por interpolación.
     - Adaptativos: Robot con control sensorial, adaptativo o funciones para control mediante aprendizaje. Se dice que es adaptativo si el robot se ajusta durante el proceso dependiendo de las condiciones detectadas. Se dice que es sensorial si se ajusta de acuerdo a la información detectada por sensores externos. Y se dice que es mediante aprendizaje si utiliza los datos de ciclos anteriores para reajustar los parámetros de control y/o algoritmos de forma automática.
     - Teleoperados: Se puede operar de modo remoto, *maestro-esclavo*, por un operador humano.
  3. Clasificación según el tipo de estructura mecánica:
     - Cartesianos.
     - SCARA.
     - Antropomórficos.
     - Paralelos.
     - Esféricos y cilíndricos.

- Robot inteligente: Robots capaces de desenvoluparse en entornos no estructurados y con eventos impredecibles. Se realimenta con información recibida por sensores y es capaz de aprender.

- Robot de servicios: Robots con autonomía total y parcial capaces de de desarrollar servicios útiles, excluyendo las tareas de fabricación.

2.4 Sistemas embebidos

El rápido avance de la tecnología ha dado pie a sistemas cada vez más compactos, económicos y fáciles de utilizar, haciendo llegar el mercado de la informática y la electrónica a un público menos especializado que ha visto en este campo un entretenimiento. A pesar de las limitaciones que presentan estas tarjetas microcontroladoras de bajo coste en comparación con los sistemas industriales tradicionales, son un excelente punto de partida para estudiantes, investigadores y, en resumen, para cualquier persona que se quiera embarcar en un proyecto que no tenga la certeza suficiente que justifique el desembolso que suponen los equipos industriales.

Por poner algunos ejemplos de estos sistemas embebidos(tabla: 2.2): Son muy conocidas las tarjetas *RaspberryPi*, los microcontroladores *Arduino* y la *BeagleBoard*, todas ellas bajo las licencias *open-source* y *open-hardware*.

	Arduino UNO R3	**RaspberryPi 2 Model B**	**BeagleBone Black**
Precio	25€	40€	55€
Procesador	ATmega328	ARMv7	AM335x
Velocidad del procesador	16Mhz	900Mhz	1GHz
Pins digitales	14 (6 PWM)	40 GPIO	65 GPIO
Pins analógicos	6	-	7
Memoria	Flash memory 32Kb, SRAM 2Kb, EEPROM 1Kb	1GB RAM	512MB DDR3RAM , 8GB eMMC
Lenguaje programación	ArduinoIDE, C Variant, Matlab	Cualquiera	Cualquiera

Tabla 2.2: Comparación entre tarjetas embebidas, información de [4],[5] y [6].

2.4.1. Arduino

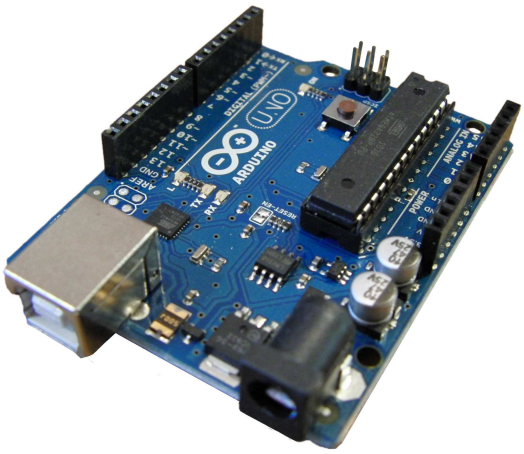


Figura 2.6: Arduino UNO.

Arduino se inicio en 2005 en el instituto IVREA de Italia como un proyecto para acercar a los estudiantes a la electrónica. Los diferentes microcontrolador que ofrece hoy en día se basan en los chips de *ATmega*, pudiendo elegir entre varios modelos dependiendo de su frecuencia de procesamiento, opciones de comunicación de que disponen, capacidad de memoria y el número de entradas y salidas totales. El modelo que se use dependerá de la finalidad del proyecto; el *Arduino UNO* (fig: 2.6) es el más simple y de uso general, utiliza el chip *ATmega328* y dispone de 16 pi-

nes digitales de entrada/salida más 6 de entrada analógica; otro ejemplo es el *Arduino LilyPad* usado para proyectos textiles e incluso se puede lavar, usa e chip *ATmega168V* o *ATmega328V* y dispone de 14 pines digitales y 6 analógicos; o el *Arduino Robot* que viene ensamblado en una estructura con ruedas, utiliza el chip *ATmega32u4* y dispone de 6 pines digitales y 6 analógicos además de botones, pantalla LCD, lector de tarjetas SD entre otros sensores y actuadores.

En los proyectos con *Arduino* el punto principal suele ser el uso de las entradas y salidas para controlar diferentes sensores y/o actuadores. Normalmente estos pines trabajan entre 0 y $5V$. Algunos de los pines digitales se pueden programar para que saquen una señal *PWM*, esto es, una señal cuadrada a una frecuencia de aproximadamente 490Hz que dependiendo del tiempo que estén a 1 lógico y el tiempo a 0 equivale a una salida analógica. Las entradas analógicas disponen de un conversor A/D de 10*bits*, lo que supone una resolución de $4{,}9mV$.

Otro aspecto de *Arduino* y el que le *ha dado alas*, son las tarjetas de expansión. Tanto *Arduino* como otros fabricantes han desarrollado módulos que se adaptan al *GPIO* de las placas *Arduino* dotándolas de funcionalidades adicionales como canales de comunicación *Ethernet*, *wifi* o *bluetooth*; circuitos controladores de motores, etc.

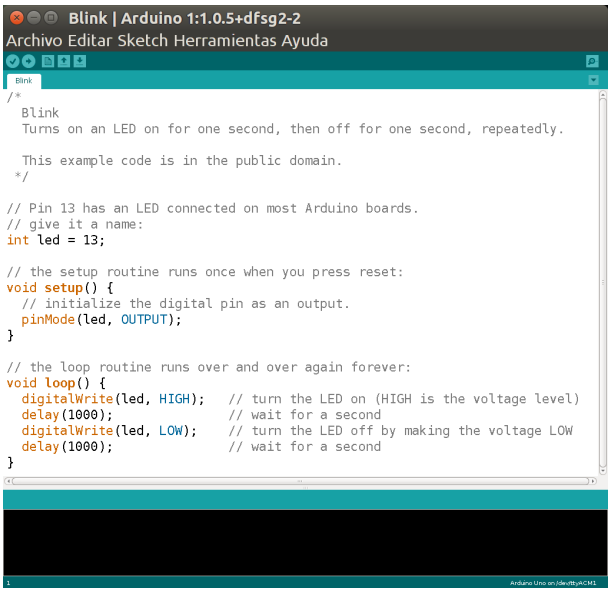


Figura 2.7: Ejemplo de un programa que hace parpadear un LED escrito en el IDE de Arduino.

La web de *Arduino* tiene disponible un IDE para cualquier plataforma, ya sea *Microsoft*, *Mac OS* o *Linux*. La sencillez de este IDE, basado en *Processing*, es la razón de que sea ideal para cualquier persona que quiera aprender electrónica y no tenga ninguna base previa. Extraído de [4]

Aunque el IDE está escrito en *Java*, el lenguaje que se usa para programar las placas es una mezcla simplificada de *C++* y *Java*. La comunicación de la placa con el ordenador, a diferencia de los microcontroladores tradicionales que necesitaban de un programador, se hace directamente por puerto *FTDI* por lo que el único requisito es tener instalados los drivers correspondientes. También es posible comunicarse con el *Arduino* por puerto serie una vez está corriendo. También es posible programar *Arduino* usando otros medios, por ejemplo, el paquete *Simulink*

de *MathWorks* dispone de bloques pensados especialmente para *Arduino* de manera que la aplicación es directa sin necesidad de hacer la conversión para adaptarlo al lenguaje de *Arduino* [3].

2.4.2. RaspberryPi

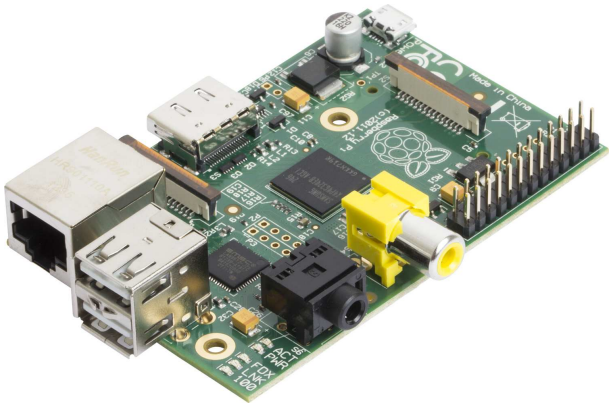


Figura 2.8: RaspberryPi Model B

La *RaspberryPi* también es un sistema embebido de bajo coste y comparte gran parte de la filosofía *open-source* y *open-hardware* con *Arduino*. A diferencia de este último, una *RaspberryPi* no es un microcontrolador monoproceso que se centra en el control de entradas y salidas, una *RaspberryPi* es capaz de ejecutar un sistema operativo completo gracias a un *system-on-a-chip Broadcom* el cual incluye un procesador *ARM11* con una unidad en coma flotante y un procesador gráfico *Videocore IV* capaz de soportar *OpenGL* y *OpenVG*. Una tarjeta SD funciona como almacenamiento y es donde se instala el sistema operativo, los periféricos se conectan a través de diferentes conectores como la salida de vídeo digital *HDMI*, puertos *USB*, entrada de *Ethernet* y salida de vídeo analógico *S-Video*. A fecha de hoy existen varias versiones de *RaspberryPi*, pero las diferencias significativas no aparecen hasta la última versión, la tabla 2.3 muestra las principales diferencias entre las versiones *RaspberryPi Model B+* y la *RaspberryPi 2 Model B*.

	RaspberryPi Model B+	**RaspberryPi 2 Model B**
SoC	Broadcom BCM2835	Broadcom BCM2836
CPU	ARM11 ARMv6 700 MHz	ARM11 ARMv7 ARM Cortex-A7 4 núcleos @ 900 MHz
Overclocking	Sí, hasta velocidad Turbo; 1000 MHz ARM, 500 MHz core, 600 MHz SDRAM, 6 overvolt. de forma segura	Sí, hasta arm_freq=1000 sdram_freq=500 core_freq=500 over_voltage=2 de forma segura
RAM	512 MB LPDDR SDRAM 400 MHz	1 GB LPDDR2 SDRAM 450 MHz
Consumo	5v, 600mA	5v, 900mA, aunque depende de la carga de trabajo de los 4 cores

Tabla 2.3: Comparación de dos modelos de *RaspberryPi* extraído de *www.xataca.com*

Una característica que acerca la *RaspberryPi* a *Arduino* son sus entradas y salida de propósito general o *GPIO* (General Purpose Input/Output), aunque por defecto hay algunas entradas configuradas para conexiones de interfaces $UART$, I_2C, y SPI, todas son configurables a excepción de los pines de alimentación y las masas. Es precisamente gracias a esta característica que la *RaspberryPi* ha cosechado tanto éxito dentro del campo de la electrónica y la robótica ya que

permite conectarle facilmente tarjetas de expansión (o *shields*) para un propósito dado como puede ser el control de motores, cámaras o cualquier dispositivo dentro de la gran variedad que existe en el mercado.

Desde la página web oficial de *RaspberryPi* se pueden descargar varios sistemas operativos dependiendo del uso que se le quiera dar. El sistema operativo de uso general más utilizado y del que se puede encontrar más documentación en la red es *Raspbian*, una versión del conocido *SO Debian* adaptado a las características de la *Rasberry*. También existen otros sistemas operativos *open-source* de uso general y como los basados en Ubuntu o Fedora, algunos especializados como son los casos de *Openelec* y *OSMC* pensados para transformar la *RaspberryPi* en un media center o alguno privativo como es *Windows10*, aunque este solo estará disponible para la *RaspberryPi 2 Model B*. Extraído de [5]

2.4.3. BeagleBoard

Figura 2.9: BeagleBoard-xM

La *BeagleBoard* surge a mediados de 2008 como el primer micro ordenador de bajo coste del mercado de la mano de *Texas Instruments* en asociación con *Digi-Key* y *Newark Element14*. Es una de las tarjetas más competitivas que hay hoy en día en el mercado, aunque su precio es algo superior al de la *RaspberryPi* (la *Beagle Board Black* se vende por unos 50$ mientras que la *RaspberryPi 2* ronda los 40$) las prestaciones de la *Beagle Board* son bastante superiores.
El procesador que usa la *BeagleBoard* es un $ARMCortex - A8$ a una frecuencia de $1GHz$ y una *GPU PowerVR SGX530* a $200MHz$. La capacidad de la RAM es la misma que la *Raspberry*, $512Mb$, la salida de audio y vídeo es mediante micro-HDMI, solo tiene un puerto USB y una tarjeta microSD de almacenamiento.

Para la conexión de tarjetas de expansión, que llaman *capes*, dispone de 92 pines reconfigurables que permiten la conexión de hasta cuatro puertos serie, un bus *CAN*, cuatro temporizadores, ocho pines permiten la señal *PWM* y dispone de siete entradas analógicas en base a $1{,}8V$ con una resolución de AD de $12bits$, además de dos conexiones *SPI* y dos puertos I_2C que permite la com unicación con cualquier hardware compatible. Extraído de [6]

Al igual que con la *RaspberryPi*, la *BeagleBoard* soporta varios sistemas operativos como los basados en *Linux*: *Fedora*, *Ubuntu* o *openSUSE*; aunque también existe soporte para *Android* y *Windows Embedded*. Cabe destacar que cuando no detecta ningún sistema operativo arranca con un *Linux* que tiene guardado en la memoria interna. Se puede usar casi cualquier lenguaje de programación como *C, C++, Java, Python, JavaScript, Android* e incluso Matlab-Simulink, aunque este último solo para el caso de la placa *BeagleBoard xM* [3].

CAPÍTULO 3

Desarrollo práctico

3.1 Sistema a desarrollar

En el primer punto del desarrollo práctico se ha querido presentar el proyecto de forma esquemática con el fin de proporcionar al lector una idea general que le facilite el seguimiento de los puntos siguientes.

3.1.1. Hardware

Empezando por el usuario, lo primero que nos encontramos representado en la 3.1 son los diferentes ***sensores biomédicos***. Los sensores utilizados en este proyecto han sido: Electrocardiograma, que mide la tensión generada por la contracción del corazon; respuesta galvánica de la piel, este mide la resistencia eléctrica entre dos puntos del cuerpo; flujo de aire al respirar, medido gracias a la variación de temperatura detectada por un termopar; electromiograma, el funcionamiento es similar al electrocardiograma; termómetro, es otro termopar y un acelerómetro para detectar la posición en la que se encuentra el usuario.

Los sensores se conectan a la placa ***eHealth sensor platform***, la cual está preparada expresamente para estos. La placa se compone de los amplificadores operacionales, filtros y otros dispositivos como los que permiten la comunicación I_2C, podemos encontrar el esquema de esta en el anexo B. La *eHealth sensor platform* está diseñada para *Arduino*, por lo que esta trabaja a $5V$ y por lo tanto se necesita añadir un dispositivo que permita conectarla a la *RaspberryPi* que trabaja a $3{,}3V$.

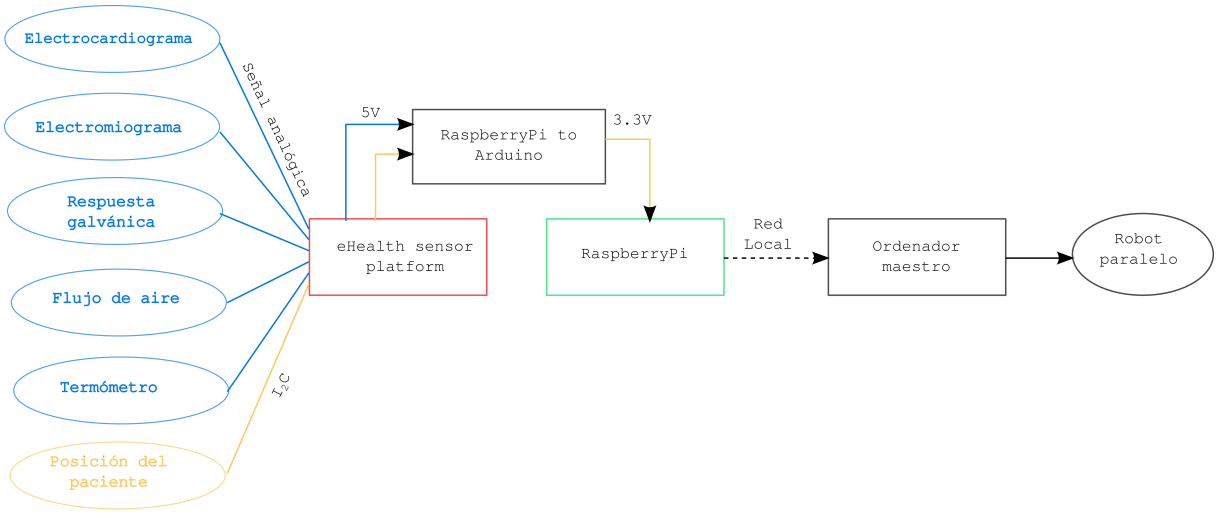


Figura 3.1: Representación gráfica del hardware del sistema desarrollado.

La placa *eHealth* además de trabajar a $5V$ también envía señal analógica mientras que la *RaspberryPi* solo dispone de entradas digitales, es por ello que se necesita la placa ***RaspberryPi to Arduino shields connection bridge***. Esta se basa en un conversor analógico−digital que lee las lecturas analógicas a cinco voltios y las convierte a digital mediante un mensaje en I_2C pudiendo así ser leídas por la *RaspberryPi*.

La ***RaspberryPi Model B*** es el ordenador que recibe la información de los diferentes sensores y la procesa para luego enviarla al ordenador maestro del robot por red local, ya sea mediante wifi o ethernet.

El ***ordenador maestro*** es el ordenador que tiene el control del robot paralelo y por lo tanto el *cerebro* del sistema, recibe la información de la *Raspberry* que determina las ordenes que enviará al robot paralelo.

Por último el ***robot paralelo*** es el agente a controlar y el que al final le da utilidad a toda la información procesada.

3.1.2. Software

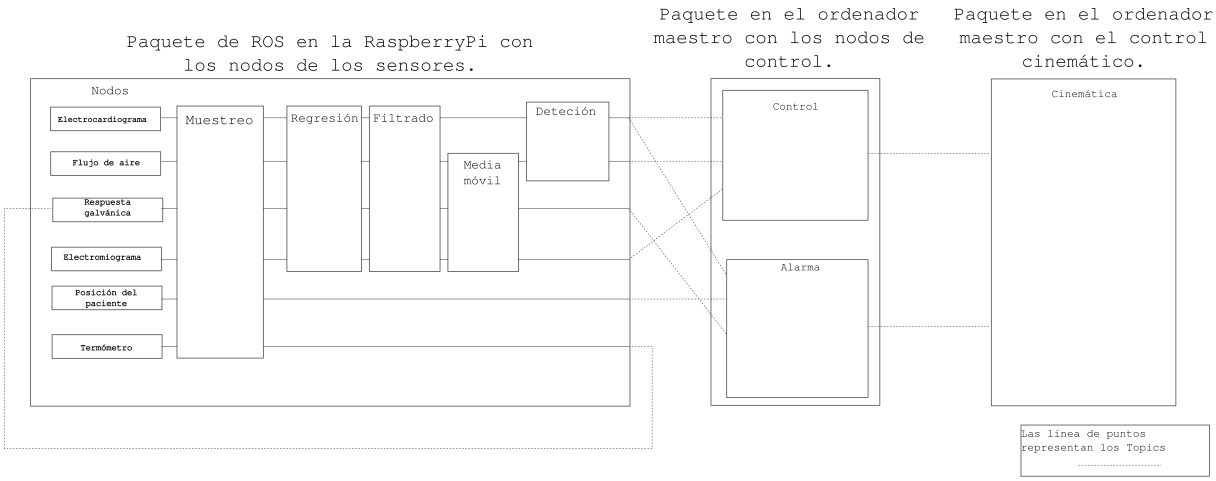


Figura 3.2: Representación gráfica del software del sistema desarrollado.

Una vez explicada la parte física del sistema el siguiente paso es intentar esquematizar la parte lógica (figura: 3.2). Lo primero que se hace es el ***muestreo de señal*** que se recibe de los diferentes sensores, esta lectura puede ser simple o continua dependiendo del sensor. Los sensores de *respuesta galvánica de la piel*, *electrocardiograma*, *electromiograma* y *flujo de aire* precisan de una lectura analógica continua a lo largo de un periodo dado, mientras que el *termómetro* basta con una lectura analógica por ciclo. El acelerómetro que detecta la *posición del paciente* también es suficiente una lectura por ciclo, pero esta se envía a través de un mensaje I_2C.

El ***procesado de señales*** es necesario para aquellas obtenidas por medio de una lectura analógica continua. Los diferentes procesos por los que pasan las señales son la *regresión*, debido a que la lectura no es totalmente periódica; el filtrado, ya que suelen presentar ruido; la media móvil, para aquellas señales que necesitan ser suavizadas; y por último la detección para obtener la información deseada, también puede usarse la derivación en algún sensor para facilitar este último paso.

La información útil de los sensores se envía a dos ***módulos de control*** situados en el *ordenador maestro* en forma de *mensajes de ROS*, en estos módulos se combina el resultado de los sensores que se traducen en el comportamiento del *robot paralelo*. Un módulo ajusta el movimiento del robot según el estado físico del usuario mientras que el otro se encarga de parar el sistema si detecta algún peligro para la persona.

El último software que encontramos es el programa que controla propiamente el robot. Mediante los canales de comunicación creados en ROS, este programa envía la cinemática con los movimientos que debe hacer el robot.

3.2 Sensores biomédicos

3.2.1. Shield eHealth

Figura 3.3: Vista cenital del shield eHealth de Cooking Hacks.

La medición de parámetros biomédicos suele precisar de equipos complejos y caros que aseguren una mínima precisión ya que su aplicación directa es la salud de las personas, generalmente tan compleja como importante. Pero con el avance de la electrónica se han conseguido abaratar suficiente los precios para desarrollar pequeños dispositivos que, aunque no tienen la precisión ni fiabilidad para obtener una certificación médica, son de gran utilidad para de investigadores, desarrolladores o incluso artistas que tiene una oportunidad menos prohibitiva para experimentar y poder aprender como funcionan estos instrumentos a más bajo nivel.

Este es el caso de la placa *eHealth* diseñada por *Cooking Hacks*, un shield capaz de dotar un *Arduino* con hasta nueve sensores biomédicos:

- Sensor de temperatura.
- Pulsioxímetro.
- Esfigmomanómetro.
- Posición del paciente.
- Flujo de aire.
- Electrocardiograma.
- Electromiograma.
- Respuesta galvánica de la piel.
- Glucómetro.

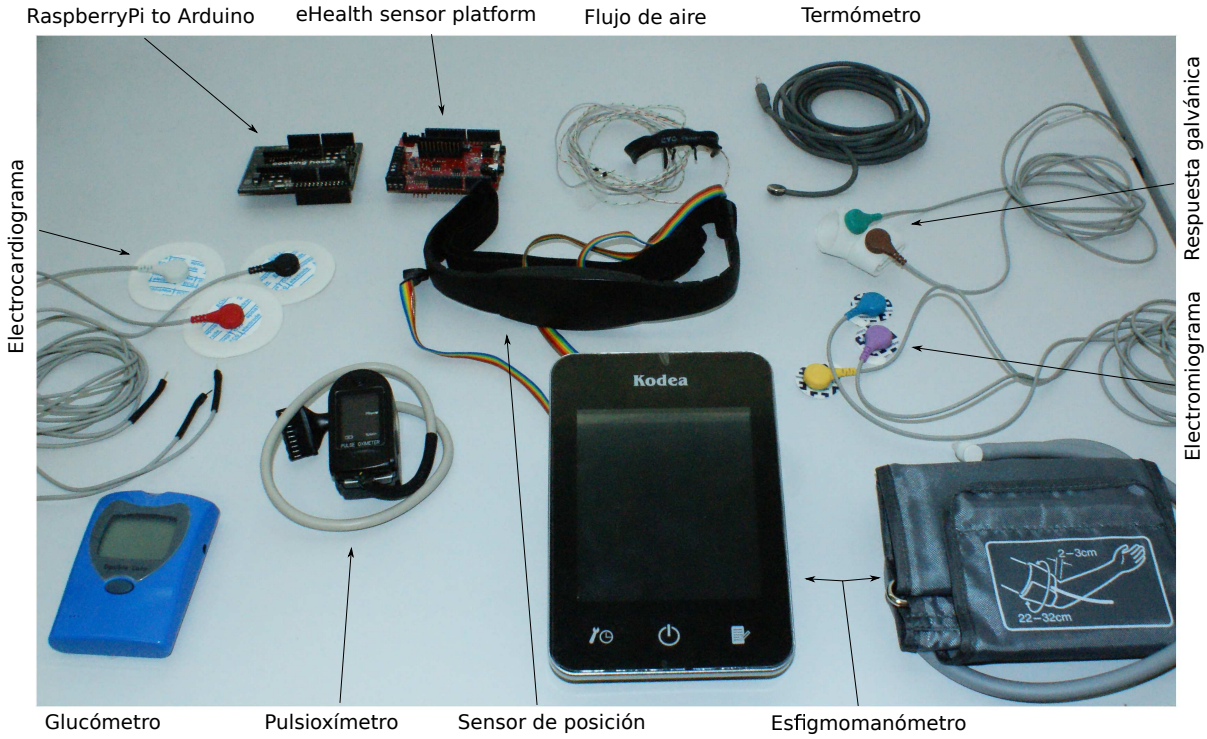


Figura 3.4: eHealth sensor platform V2.0 kit

Debido a que para este proyecto se ha usado una RaspberryPi, ha sido preciso el uso de una placa de conexión *RaspberryPi−Arduino* (figura: 3.5) que actúa como *conversor A/D de 12bits* y por una cara tiene la misma distribución de pines que un *Arduino ONE* mientras que la otra se adapta al *GPIO* de la *Raspberry Pi B*[9].

Aunque tanto el *Arduino* como el *ADC* trabajan entre 0 y $5V$ existen algunas diferencias entre usar uno y otro. Con el *ADC* se han mejorado la precisión pasando de los $10bits$ de *Arduino* a $12bits$ que presenta el conversor. La otra ventaja y por la cual se ha optado por la *Raspberry Pi* es que esta corre un sistema operativo completo, dando mucha más versatilidad y alcance al conjunto.

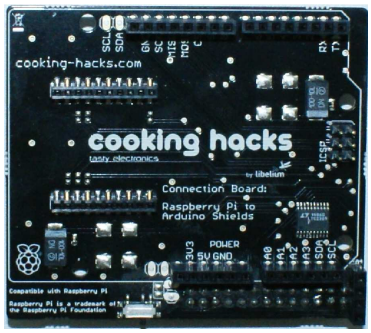


Figura 3.5: RaspberryPi to Arduino shields connection bridge.

Por otro lado cabe mencionar que el uso de un sistema operativo resulta perjudicial para la frecuencia de adquisición de datos, ya que este tiene interrupciones que pueden provocar retrasos entre las instrucciones de nuestros programas. Este problema, además, se agrava con el

tiempo intrínseco a la conversión del *ADC* que según el *datasheet* podría ser de hasta $2\mu s$.

En los siguientes apartados se explicará con más detalle cada uno de los sensores usados en este proyecto.

3.2.2. EMG

Cuando el objetivo es saber si una persona se está moviendo y como, podemos enfocarlo desde diferentes ángulos. Si es suficiente el enfoque cinético podemos optar por sensores de movimiento o de posición como pueden ser acelerómetros, cámaras de infrarrojos o incluso una webcam común más un software que reconozca movimiento, cuando la dinámica empieza a cobrar importancia ya hay que recurrir a sensores que midan fuerzas como los piezoeléctricos que devuelven una tensión proporcional a la presión a la que están siendo sometidos o recurrir a las propiedades físicas del cuerpo, cosa que se complica enormemente cuando este no es homogéneo, uniforme e isótropo como es el caso del cuerpo humano. Pero cuando el objetivo es saber como es mueve una persona, más concretamente, como se mueve un músculo de una persona podemos recurrir a un electromiograma.
Un electromiograma consiste en detectar la diferencia de tensión que surge con la contracción de un músculo. Pero antes de entrar en cómo detectar esa diferencia de tensión es interesante conocer cómo y porqué se produce.

La contración de un músculo es una reacción en cadena iniciada por la llegada de un impulso nervioso. Cuando el músculo está relajado existe una diferencia de voltaje entre la carga intracelular y la superficie externa que puede estar entre -80 y $-90mV$, esto es debido a la proteína *bomba sodio−potasio* encargada del transporte de iones. Entendiendo el músculo como una membrana semi−permeable, el impulso nervioso facilita el flujo de iones Na^+ aumentando la tensión hasta los $30mV$ [8]. Este fenómeno, llamado *despolarización* y que recorre el músculo a lo largo, es precisamente la señal que esperamos detectar con el electromiograma. Este *potencial de acción* es lo que va provocar la salida de calcio y este, a la vez, provoca que los filamentos de actina se desplacen sobre filamentos de miosina produciendo un acortamiento del tejido muscular. Aunque esto queda fuera del alcance de este trabajo.

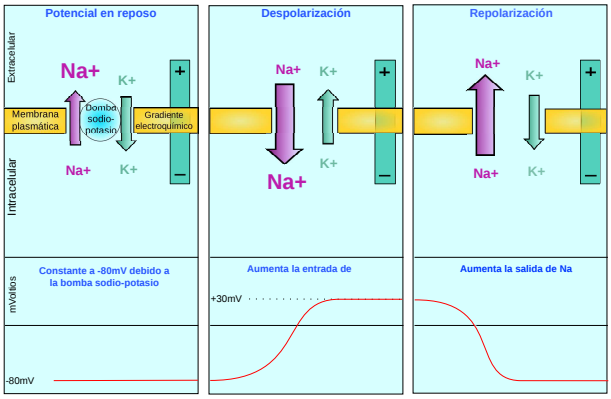


Figura 3.6: Esquema de un ciclo despolarización/repolarización dentro de membranas excitables. Adaptado desde [7].

Inmediatamente después de la *despolarización* la *bomba sodio−potasio* da paso a la *repolarización* revirtiendo el intercambio de iones para volver al estado inicial de reposo.

Este proceso no es único para cada contracción de un músculo, si no que para mantener un músculo contraído se produce un *bombardeo* desde varias neuronas, esto supondrá que la señal que va a leer el sensor va a ser muy sucia, aunque esto se explicará más adelante con detalle.

Para medir esta señal se ha optado por utilizar el amplificador instrumental *AD8221*, un tipo de amplificador diferencial. Como su propio nombre indica este dispositivo es el encargado de medir los cambios de voltaje en el músculo y preamplificarlos.

La señal del electrocardiograma que va a recibir el amplificador es muy pequeña, estará entre unos pocos microvóltios y $3mV$. La ganancia que amplifica la señal se ajusta con la resistencia R_G, y siguen la siguiente fórmula: $G = 1 + \frac{49,4k\Omega}{R_G}$.
En este punto solo hace falta una preamplificación por lo que se ha usado una resistencia $R_G = 240\Omega$ que proporciona una ganancia de 206,83 aumentando el máximo voltaje de salida hasta los $0,62V$. Este máximo aún dista mucho de los $5V$ que la placa es capaz de devolver, esto es porque este sensor cuenta además con un *rectificador de señal*, un *filtro de paso bajo* y finalmente un *amplificador de ganancia variable* que hacen que la salida se acerque a los $5V$. Estos dispositivos se han reservado para la sección de *Procesamiento de señales*.

Como se ha mencionado antes, se produce un frente de excitación que se propaga a lo largo del músculo a una velocidad aproximada de $2 - 6m/s$ [7]. Conectando las dos entradas del amplificador a una distancia suficiente para medir la tensión entre el potencial de acción y una zona en la que todavía no haya llegado dicha tensión podemos medir la señal que está haciendo que el músculo se contraiga, para más adelante tratarla e intentar identificar frecuencias y amplitudes de la misma.
La forma de conectar las entradas del amplificador a los músculo es mediante electrodos. Aunque existe una gran variedad de electrodos, para este proyecto se han elegido *electrodos pediátricos para ECG*, con gel sólido y un tamaño de entre 20 y $30mm$, más pequeños que la opción más económica de $50mm$ pero más selectivos para evitar las interferencias de los músculos que no necesitamos medir. Desde un primer momento se descartó cualquier opción de usar métodos invasivos como agujas electrodo. Aunque son más precisas que los electrodos superficiales y soportan mejor los movimientos que pueda hacer el usuario, se necesita cierta formación médica para poder aplicarlos sin riesgo.

Para colocar los electrodos de gel sólido hay que identificar el músculo que nos interesa medir y pegar dos de los electrodos en la zona del músculo que más varía su tamaño con la

contracción, estos dos electrodos serán las entradas $+IN$ y $-IN$ del amplificador *AD8221*. El tercer electrodo, correspondiente a la entrada *REF* del amplificador, será el voltaje de referencia. En el cuerpo humano existen varios puntos que apenas varían su tensión y por lo tanto ideales para tomarlos como referencia. Ver la figura A.1 del anexo.

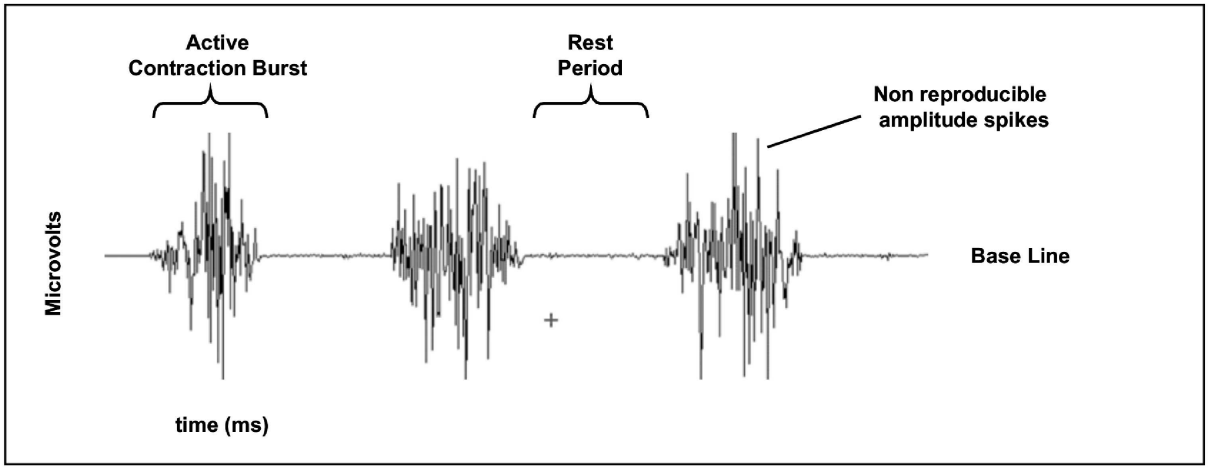


Figura 3.8: EMG sin tratar de tres contracciones de bíceps. Adoptada desde [7]

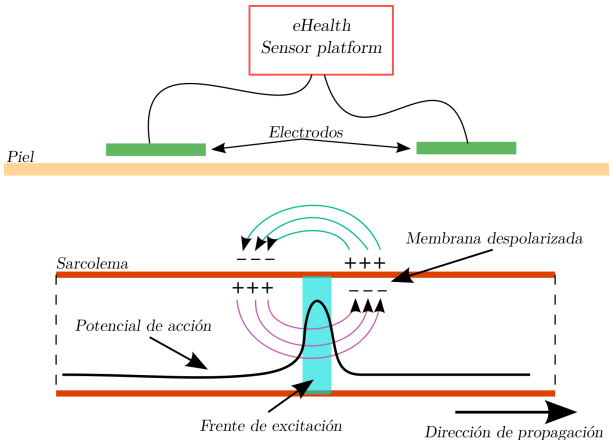


Figura 3.7: Esquema del frente de excitación, adaptado desde [7]

El uso que se le ha dado a este sensor es poder ajustar la resistencia que ejerce el *robot paralelo* al esfuerzo que está realizando el usuario.

Antes de medir debemos saber que parámetros son los más apropiados. Ya se ha comentado el rango de tensiones en el que se mueve el electrocardiograma y como se ha amplificado, pero también se ha de tener en cuenta que se trata de una señal que contiene información en el espectro de frecuencias. Esta información se encuentra principalmente en el rango que va desde los $10Hz$ hasta los $500Hz$ [7]. Debido a la limitación de los equipos usados para este proyecto se ha decidió usar como frecuencia de muestreo F_s la mínima que no viola el *teorema de Nyquist* ($F_s > 2F_{max}$), por lo tanto se ha optado por $F_s = 1KHz$.

Aunque no se ha podido tomar una lectura directamente de la salida del amplificador debido a que a la salida del circuito la señal ya ha sido pretratada, la señal sin tratar es similar a la que se observa en la figura 3.8, una señal con valores muy aleatorios y difícilmente reproducible.

3.2.3. ECG

Aunque también es un músculo, el corazón presenta suficientes particularidades para estudiarlo por separado. La contracción del corazón, a diferencia de la mayoría de los músculos, no está destinada al movimiento de las articulaciones si no que se trata de un sistema hidráulico capaz de bombear la sangre por todo el cuerpo. Está función, además, se realiza de manera involuntaria y periódica.

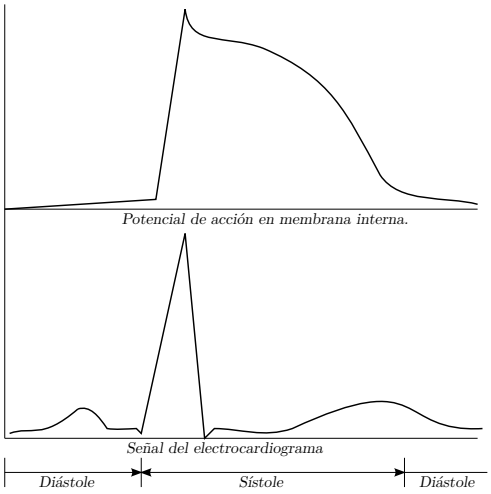


Figura 3.9: Potencial de acción de las membranas frente a la señal del electrocardiograma. Basado en [8]

El modelo de las membranas semi-permeables descrito en el punto anterior es también válido para el *electrocardiograma* salvando alguna diferencia. La primera diferencia la encontramos en el potencial de un grupo de células en el momento que el corazón está en reposo (*diástole*), a diferencia de las otras células estas incrementan su potencial lentamente (probablemente debido a la permeabilidad característica de la célula) hasta el punto de llegar al umbral que provoca la *despolarización*, y con ello la contracción (*sístole*)sin necesidad de un estímulo nervioso. Precisamente este mecanismo el que controla la frecuencia de los latidos [8]. La otra diferencia a destacar reside en el conjunto de fibras especializadas en propagar el *potencial de acción*, sincronizando las contracciones de cada sección del corazón para un bombeo óptimo de la sangre [8] y generando una señal más clara que la del *EMG*. Esta misma secuencia de contracciones producidas por el *potencial de acción* es lo que más tarde se dibujará con el característico seno cardinal.

La lectura de la señal se puede realizar con electrodos comunes de $50mm$ de diámetro ya que a diferencia del *electromiograma* la señal producida por el corazón es bastante fácil de medir y que además estos son especialmente económicos debido a su alta producción.
La colocación de los electrodos se realiza siguiendo el *triángulo de Einthoven* 3.10. Cada una de

los lados del triángulo representa una derivación periférica [8] o diferencia de potencial entre dos de los vértices. Para este proyecto se ha usado la *derivación II* ya que es la más clara de las tres, otras opciones podrían haber pasado por añadir o cambiar la distribución de los electrodos para obtener mejores lecturas como las *derivaciones periféricas aumentadas*, las *derivaciones precordiales* [8], aunque para el alcance de este trabajo cualquiera de las derivaciones básicas entra dentro de los requisitos.

La electrónica para este sensor es muy similar a la utilizada para el *electromiograma*. El shield *eHealth* utiliza el amplificador diferencial *INA321* para amplificar la tensión entre los puntos *RA* y *LL* del *triángulo de Einthoven*, poniendo la referencia en el punto *LA*.

A diferencia de la señal que se obtiene con una medida del *EMG*, la señal de *ECG* (fig:3.11) es bastante clara y a simple vista se podrían obtener datos como el número de latidos por minuto o detectar síntomas de una posible arritmia.

El problema de la señal como está es que no podemos trabajar con ella. El tiempo entre las medidas no es constante y la cantidad de ruido complica los algoritmos usados para detectar patrones por lo que ha hecho falta procesar la señal.
Esto se ha hecho para conocer el estado físico del usuario, conociendo sus pulsaciones por minuto y la variación de estas a lo largo del tiempo se puede detectar situaciones de cansancio, estrés, etc.

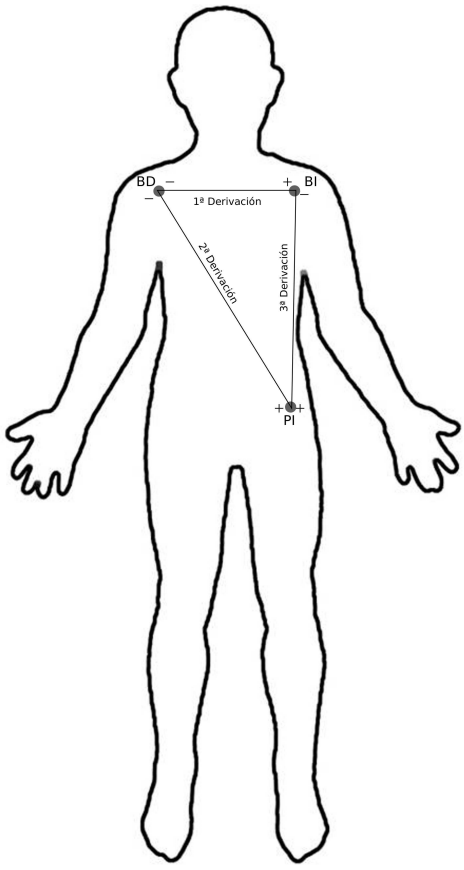


Figura 3.10: Triángulo de Einthoven. Adaptado de [8]

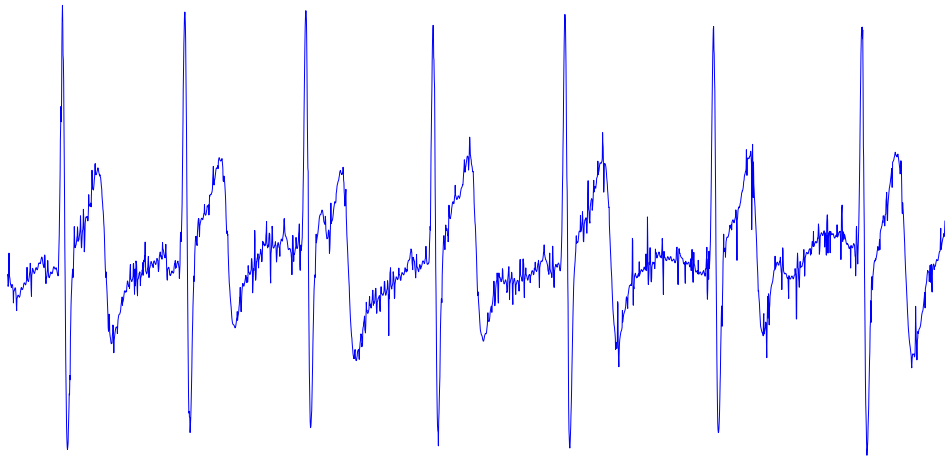


Figura 3.11: Señal del electrocardiograma sin tratar.

3.2.4. Flujo de aire y sensor de temperatura

Posiblemente los sensores más básicos que presenta *eHealth* sean *sensor de temperatura* y el detector de *flujo de aire*. Ambos usan un *termopar* conectado a un amplificador que saca una señal proporcional a la temperatura que detecta. La temperatura detectada por el *sensor de temperatura* es proporcional a la tensión, únicamente se ha tenido que corregir de la biblioteca los valores de la tensión que alimenta el amplificador y el valor de algunas resistencias.

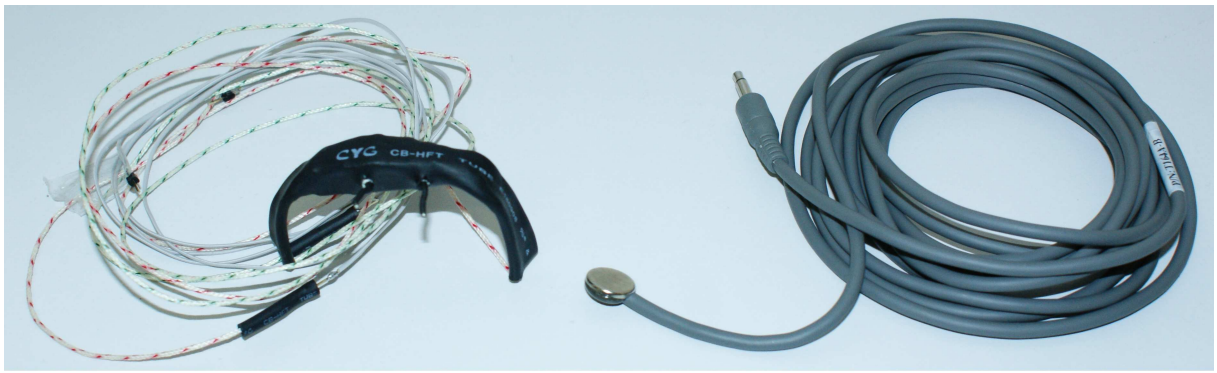


Figura 3.12: Sensor de flujo de aire a la izquierda y termómetro a la derecha.

Para detectar el flujo de aire no ha sido necesario saber la temperatura, ya que la información que necesitamos es la variación de esta. Asegurándonos que el extremo del termopar recibe el flujo de aire al respirar podemos saber cuando el usuario esta inspirando o espirando, viendo si se enfría o calienta respectivamente.

Mientras que la lectura del *sensor de temperatura* es directa y no ha hecho falta pasarle ningún tratamiento, no ocurre lo mismo con el *flujo de aire*. El *flujo de aire* detectado 3.13 tenía mucho ruido para detectar picos con claridad y el proceso de inspiración es poco suave comparado con la espiración. Todos estos procesos de tratamiento de señales se abordarán en el punto siguiente de *Procesamiento de señales*.

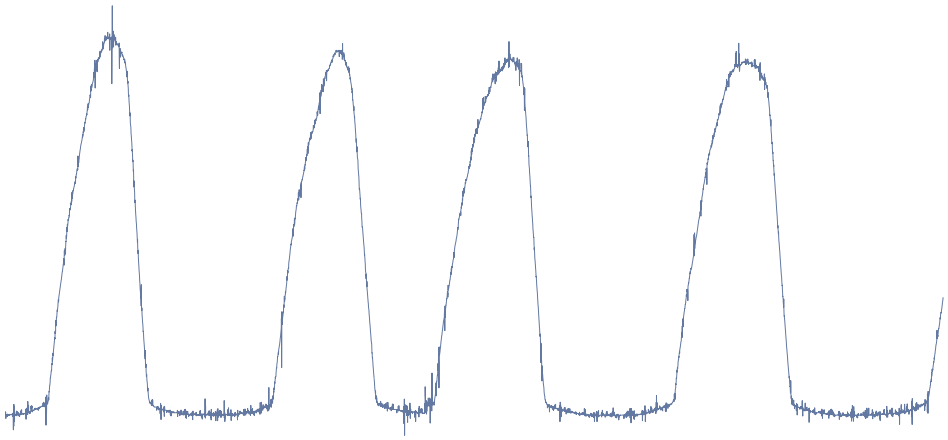


Figura 3.13: Señal sin tratar detectada por el termopar.

3.2.5. Posición del paciente

Para detectar en que posición se encuentra el usuario se ha usado un acelerómetro *MMA8452Q*, este devuelve la orientación del plano en el que se encuentra. Esto se ha usado simplemente como control para poder para el proceso si se detecta que el usuario se ha caído o ha cambiado su posición de manera que pudiera lesionarse por una mala postura.

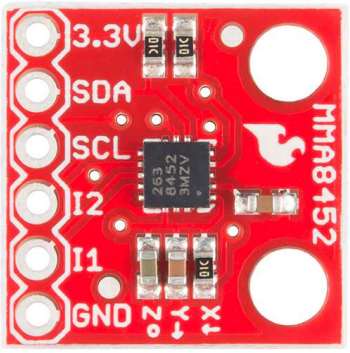


Figura 3.14: Vista cenital del acelerómetro MMA8452Q

3.2.6. Respuesta galvánica de la piel

Figura 3.15: Aplicación del sensor de respuesta galvánica de la piel.

La *respuesta galvánica de la piel* es quizás el sensor más útil de los que disponemos para conocer el estado psicológico del usuario. Su funcionamiento es también muy simple, únicamente se trata de medir la resistencia eléctrica que ejerce el cuerpo humano entre dos dedos índice y corazón. Esta resistencia se mantiene constante a lo largo del tiempo si el usuario está relajado, pero es en el momento que este se excita por algún motivo, como miedo o estrés, que sus manos empiezan a sudar y por lo tanto la resistencia cae. Este sensor al igual que conocer la posición del usuario se ha usado como alarma para detener cualquier ejercicio que haya podido causar el sobresalto.

3.2.7. Otros sensores biomédicos

El pack del shield *eHealth* traía otros sensores que no se han utilizado para este proyecto por ser poco "adaptables" a un proyecto como es este de monitorizar el estado de una persona de la manera menos invasiva posible. Tanto el *pulsioxímetro*, como el *glucómetro* y la *esfignomanómetro* (presión sanguínea) necesitan de hardware adicional y salvo el *pulsioxímetro*, que devuelve información similar al *ECG*, no vale la pena realizar medidas constantemente.

Figura 3.16: Fotografía de glucómetro, esfignomanómetro y pulsioxímetro.

3.3 Sistema de desarrollo

El objetivo de este proyecto es proveer a cualquier robot con los sensores biomédicos del capítulo anterior, aunque quizás hubiese sido más fácil escribir programas *ex profeso* de cada sensor para un robot en concreto, eso supondría un esfuerzo extra cada vez que se quisieran implementar en otro robot que trabaje de manera diferente. Una manera de abordar esto es trabajando con nodos, donde los procesos pueden ser independientes entre si o abrir los canales de comunicación que necesiten. En este proyecto se ha usado *ROS (Robotic Operating System)* instalado en una *RaspberryPi* para crear y coordinar cada uno de los nodos.

3.3.1. ROS

ROS es un proyecto que nació en la universidad de Stanford a mediados de 2007 como un proyecto para crear un framework con el que desarrollar sistemas robóticos. *ROS* sigue una filosofía colaborativa muy ligada al *software-libre*, se basa en crear una red de desarrolladores alrededor de todo el mundo en la que cualquier persona puede aportar algo. Gracias a esto varias instituciones se interesaron en colaborar y hoy en día se ha convertido en una herramienta muy útil tanto para alguien que simplemente tenga la robótica como *hobby* como una gran empresa que necesite un sistema de automatización industrial. Un ejemplo reciente de esto lo encontramos en la empresa *BOSCH*, la cual adquirió un *Tesla Model S* al que dotó con *Ubuntu 14.04* y *ROS* para su proyecto de investigación (Fuente original: *www.stuff.tv*).

ROS trabaja con nodos, esto es, existe un ordenador principal que tendrá el rol de *host* y será el encargado de ejecutar *roscore* para que cada nodo que se esté ejecutando (no necesariamente dentro del mismo ordenador) se conecte al núcleo de *ROS* y sea este el que coordine la comunicación entre nodos, así como otras herramientas. En este proyecto se ha creado un nodo para cada sensor con el fin de poder usarlos individualmente, en la figura 3.17 se muestra una representación gráfica de los nodos.

Antes de hablar de los nodos habría que describir un poco la jerarquía con la que *ROS* organiza los directorios. Aparte de los propios archivos que *ROS* necesita para ejecutarse, se crea un espacio e trabajo en el que se guardarán tanto nuestros paquetes como todos los archivos auxiliares para poder trabajar con ellos. Los paquetes no son más que directorios donde se alojan los ejecutables, el único requisito es que tengan un archivo *CMakeLists.txt* con las opciones de compilación y un archivo *package.xml* con información varia como el propietario, licencia o dependencias. Dentro de estos paquetes es donde se guardan los ejecutables y demás archivos que podamos necesitar para compilarlos y que son precisamente estos ejecutables los que *ROS*

entenderá como nodos cada vez que estén corriendo.
Uno de los problemas con el que se ha tenido que lidiar es que en la documentación oficial de *ROS* no se explica como añadir bibliotecas externas dentro de los paquetes creados por el usuario. Esta información se encuentra en la documentación de *CMake*[11].

```
# include_directories(include)

include_directories(
  ${PROJECT_SOURCE_DIR}/include/
)

## Declare a cpp library

 add_library(arduPi
   ${PROJECT_SOURCE_DIR}/include/eHealth/arduPi.cpp
 )

add_library(eHealth
${PROJECT_SOURCE_DIR}/include/eHealth/eHealth.cpp
)
```

La explicación del código es sencilla, la sección *include directories(include)* se le indica a *CMake* la dirección donde están guardados los archivos de cabecera. La sección *Declare a cpp library* manda la instrucción de crear el archivo objeto del ejecutable de una biblioteca para luego poder añadirlo al compilar. Con este método se han podido incluir librerías externas dentro de los nodos de *ROS*.
Como se ve en la figura 3.17, los nodos de *ROS* pueden abrir canales de comunicación o conectarse a ellos. Estos canales, que *ROS* conoce como *topics*, se activan cuando un nodo crea un *publisher* y a partir de ahí cualquier otro nodo puede publicar, para enviar información, o suscribirse si lo que necesita es recibirla. Hay que tener en cuenta que para enviar y recibir información esta tiene que usar un formato conocido por *ROS*, esto es, usando *mensajes*. Un *mensaje* es simplemente un tipo de variable definida en un archivo auxiliar.
Esto se muestra en el siguiente trozo de código extraído de un programa usando el *ECG* publica los latidos por minuto:

Se crea el publisher EMG_pub que abrirá el topic *EMG* y será de tipo *Float*64.

```
ros::Publisher EMG_pub = n.advertise<std_msgs::Float64>("EMG", 1);
```

Se declara una variable con el formato del mensage que queremos publicar.

```
std_msgs::Float64 beats;
```

Y por último se llama a la función publish() de *ROS* para enviar el mensaje al *topic*.

```
EMG_pub.publish(beats);
```

El suscriptor tiene una particularidad, y es que uno de sus parámetros es una función *callback* que se ejecuta cuando se publica un mensaje y usa este como parámetro de entrada.

```
ros::Subscriber EMG_sub = n.subscribe("EMG",1,EMG_callback);
```

ROS además dispone de herramientas para ver el estado de los nodos, publicar dentro de ellos, ver que mensajes envían, etc.

Otra herramienta auxiliar que se ha usado ha sido el simulador de robótica *V-Rep*, que cuenta con el plugins para conectarse automáticamente a *ROS* desde que arranca. Además, dentro de su APIs, tiene funciones para interactuar con los nodos de *ROS* con apenas una línea de instrucción. En la parte final de este proyecto se ha añadido la instrucción de elevar la plataforma del robot paralelo al que va enfocado el proyecto proporcionalmente al esfuerzo muscular del usuario.

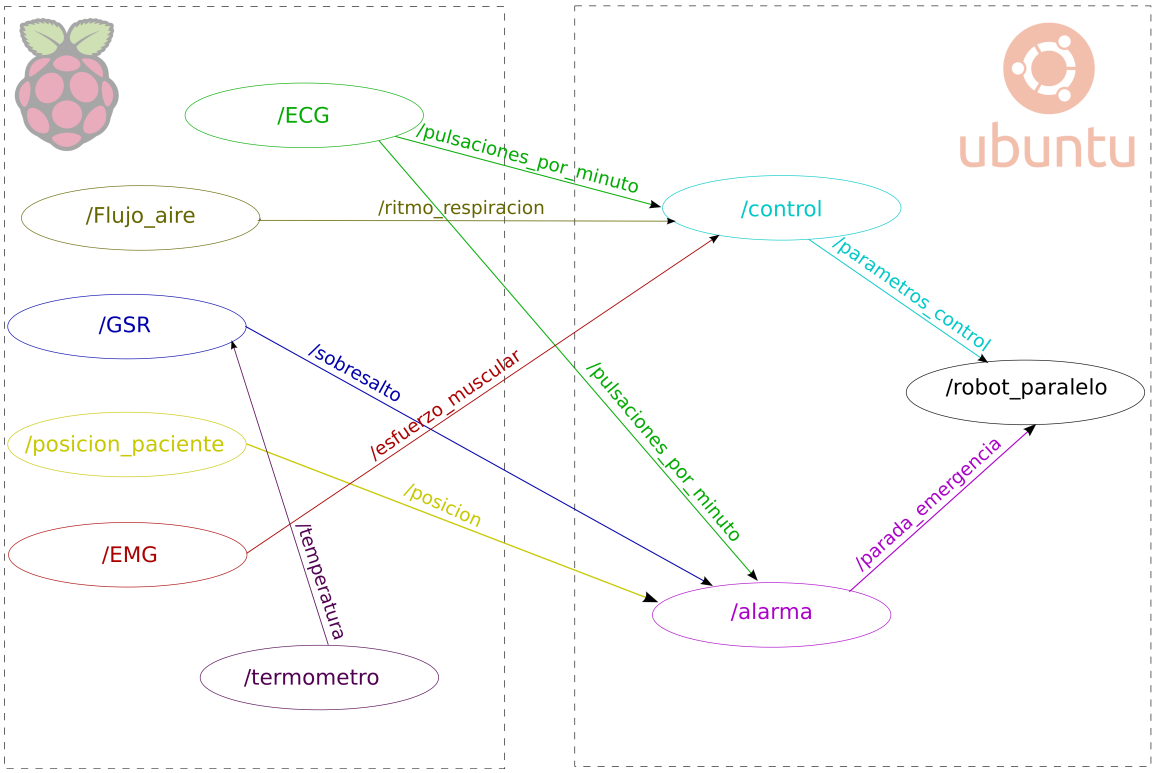


Figura 3.17: Representación gráfica de los *nodos* y *topics* en ROS.

3.3.2. RaspberryPi Model B

Aunque es posible instalar *ROS* en una *RaspberryPi* con *Raspbian*, esto no es un proceso directo como el que se necesita para instalar *ROS* en un ordenador con el sistema operativo *Ubuntu*, al cual basta con añadir los repositorios al sistema de gestión de paquetes *APT* y llamar a la instrucción de instalación. Toda esta información se encuentra en [10].

Raspbian utiliza una arquitectura *ARM* diseñada especialmente para dispositivos móviles, por lo que el sistema de gestión de paquetes no dispone de todas las dependencias necesarias para *ROS*. Esto se soluciona con el sistema de gestión de paquetes *Pip*, que trabaja con software escrito en *Python* y con *Rosdep*, que es una herramienta interna de *ROS* para instalar dependencias de este. Los pasos a seguir se pueden encontrar en *Installing ROS Hydro on Raspberry Pi* de [10], aunque este tutorial tiene un *bug* en una instrucción del cual se envió un informe al encargado del texto.
Una vez instalado *ROS* en la *RaspberryPi* se debe crear un espacio de trabajo donde se almacenarán los programas que escribamos en forma de paquetes, este paso es el mismo tanto para*Ubuntu* como para*Raspbian*.

Otro aspecto que cabe mencionar sobre *ROS* en la *RaspberryPi* es el hecho de que esta va a trabajar comunicándose con el ordenador del robot que hará de *master*. Al ejecutar el *master*, este crea un servidor al que conectar todos los dispositivos que se necesiten únicamente indicanco la IP local del *master* en cada ordenador que queramos conectar. Una vez conectados, La *RaspberryPi* es capaz de crear *nodos* y de abrir *topic* por los que enviar *mensajes* a cualquier *nodo* que se subscriba a dichos *topics*.

3.4 Procesamiento de señales

Como se ha ido comentando en secciones anteriores, las señales han sido procesadas antes de poder extraer información útil.
Principalmente se han usado filtros digitales implementados mediante software pero cabe destacar que el shield *eHealth* ya dispone de filtros que realizan un pretratamiento de la señal, sobre todo se tratan de filtros de paso bajo para reducir el ruido que pueda llevar la señal. El circuito del *EMG* además lleva un rectificador de señal que invierte el signo de las señales negativas.

Uno de los puntos más importantes de este proyecto ha sido la implementación mediante software de los diferentes tratamientos de datos. Empezando por preparar la señal para poder

ser tratada con diferentes filtros hasta la detección de patrones como picos o fuertes pendientes.

Partimos de dos vectores, uno con los valores de la señal y otro con el tiempo en el que se ha medido.

3.4.1. Regresión

Lo primero que se ha hecho para toda las medidas tomadas ha sido usar la **regresión lineal** para asegurar que nuestros puntos son equidistantes en el tiempo para así estar seguros de que la frecuencia de muestreo es correcta para cada punto.
Para ellos se ha programado un algoritmo para definir los valores de las variables de la función 2.2.

```
1  pos=pos1;
2  while((round(1000000/Fs)*i-regression_band )>time[pos])pos++;
3  pos1=pos;
4  while((round(1000000/Fs)*i+regression_band)>=time[pos])pos++;
5  pos2=pos;
6  ne=pos2-pos1;
7
8  x=new double [ne*m];
9  y=new double [ne];
10
11 for (j=pos1;j<pos2;j++)
12   {
13     y[j-pos1]=data[j];
14     x[j-pos1]=1.;
15
16     for (k=1;k<m;k++) x[j-pos1+k*ne]=x[j-pos1+(k-1)*ne]*time[j]/1000.0;
17     }
18 regresion(x,y,b,m,ne);
```

Desde la línea 1 hasta la 6 se recorre el vector de tiempos buscando que posiciones de los vectores iniciales están dentro de una banda de tamaño definido. Una vez conocida esta banda creamos las matrices X e Y, ne indica el número de datos usados para la regresión y m el grado de esta. El bucle $for()$ de la línea 11 asigna valores a las posiciones de las matrices. La función $regresion()$ resuelve la ecuación 2.2 y guarda los valores en el vector b.
A continuación se guardan en otro vector de datos los resultados de la ecuación $y = a + bx + cx^2 + ...$ siendo x cada uno de los puntos para los que se ha hecho la regresión. Obteniendo un vector con, ahora si, los datos equidistantes a los que poder aplicar filtros.

3.4.2. Filtrado

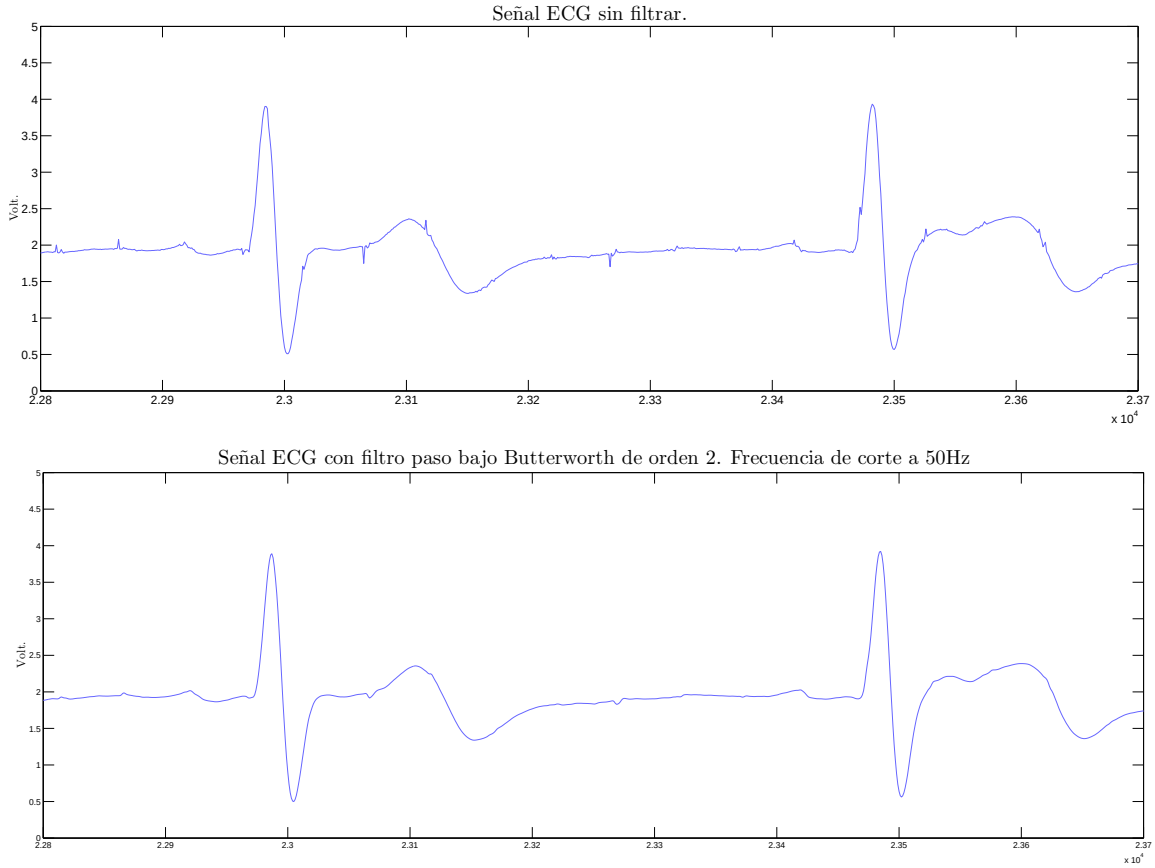


Figura 3.18: Ejemplo de un electrocardiograma filtrado.

Para aplicar los filtros se ha usado la biblioteca *DSPFilters* sujeta a la licencia *open source* del *MIT*. En la tabla 3.1 se puede ver que tipo de filtro se ha usado para cada sensor.

Tabla 3.1: Parámetros usados para filtrar las diferentes señales.

Sensor	**Tipo de filtro**	**Tipo de respuesta**	**Orden**	**Frecuencia de corte**
ECG	Paso bajo	Butterworth	2	50Hz
EMG	Paso banda	Chebyshev	2	500hz y 10Hz
Flujo de aire	Paso bajo	Butterworth	2	30Hz
GSR	Paso bajo	Chebyshev	2	100Hz

3.4.3. Averaging

La siguiente operación que se ha aplicado a todas la señales ha sido la **media** o ***averaging***, se ha programado un algoritmo de la función 2.5 para todas salvo para el *EMG* que se ha usado la media cuadrática debido a que necesita más suavizado [7].

$$a = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}} \tag{3.1}$$

```
A=new double [(pot2-25)];
    for(i=0;i<pot2-Fs/20;i++)
    {
        A[i]=0;
        for(j=0;j<Fs/20;j++)
        {
            A[i]=A[i]+pow(F[0][i+j],2);
        }
        A[i]=A[i]/(Fs/20);
        A[i]=sqrt(A[i]);
    }
```

Hay que tener en cuenta que con la media simple se reduce el número de datos, por lo que la frecuencia de muestreo también cambia. La frecuencia de muestreo original se divide por el número de datos usados para hacer la media.

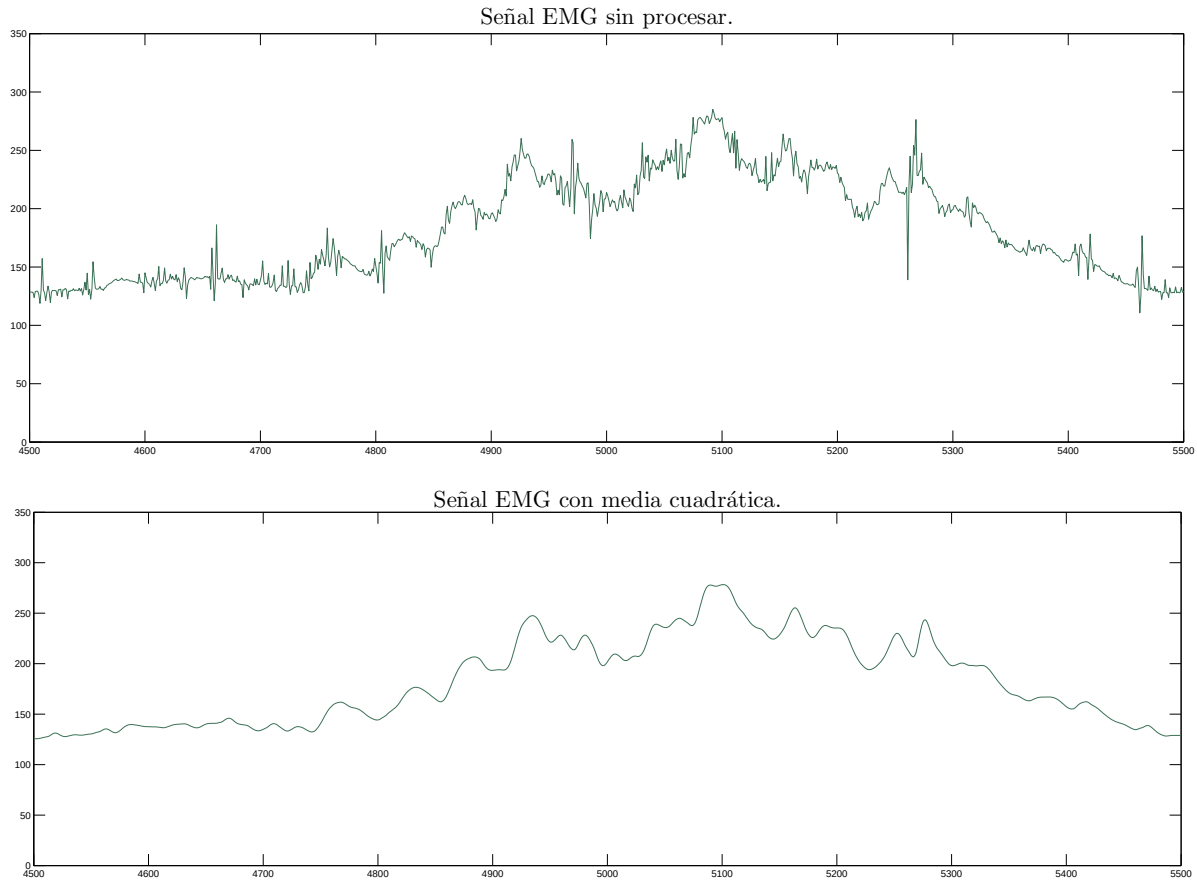


Figura 3.19: Suavizado de una señal EMG usando la media cuadrática.

3.4.4. Detección

Una vez la señal se ha acondicionado se ha procedido a extraer la información. Tanto el *EMG* como la *respuesta galvánica de la piel (GSR)* se basan en detectar cuando la señal supera un cierto umbral en muy poco tiempo por lo que la programación de esto no ha supuesto mayor problema. Existe una diferencia entre la forma de definir el umbral que activará la alarma. Mientras que con el *EMG* este umbral es constante debido a que la tensión en reposo no varía, la conductividad de la piel medida con el *GSR* puede cambiar significativamente por las variaciones en la segregación de sudor sin que deba detectarse nada. Esto se ha resuelto con un umbral flotante que se va actualizando con el tiempo y el programa únicamente detectará las repentinas variaciones de sudor.

En cambio para los sensores de *flujo de aire* y el *ECG* si que es necesario detectar patrones a lo largo de la señal, en concreto hay que detectar los picos de señal, los cuales son representación

de cada ciclo cardiaco o de la respiración. A la señal del flujo de aire se le ha hecho, además, la derivada con la idea de tener picos mucho más definidos en cada proceso de *inspiración*.
El siguiente paso ha sido establecer el umbral detectando el valor máximo dentro del vector de datos para el *ECG* o el mínimo en el caso de la respiración. Estos valores se multiplican por 0,8 y 0,45 respectivamente establecer el umbral. Una vez se tiene el umbral se recorre el vector hasta que un valor supera el umbral, entones se compara el valor leído con el anterior y posterior; si estos valor son los dos menores se trata de un pico y si son mayores es un valle, entonces se almacena la posición y salta $0{,}1 \times F_{muestreo}$ ya que es seguro que no hay un pico o valle a esa distancia. Llegados a este punto solo resta dividir el número de posiciones entre la $F_{muestreo}$ para saber los segundos que hay entre picos o valles.

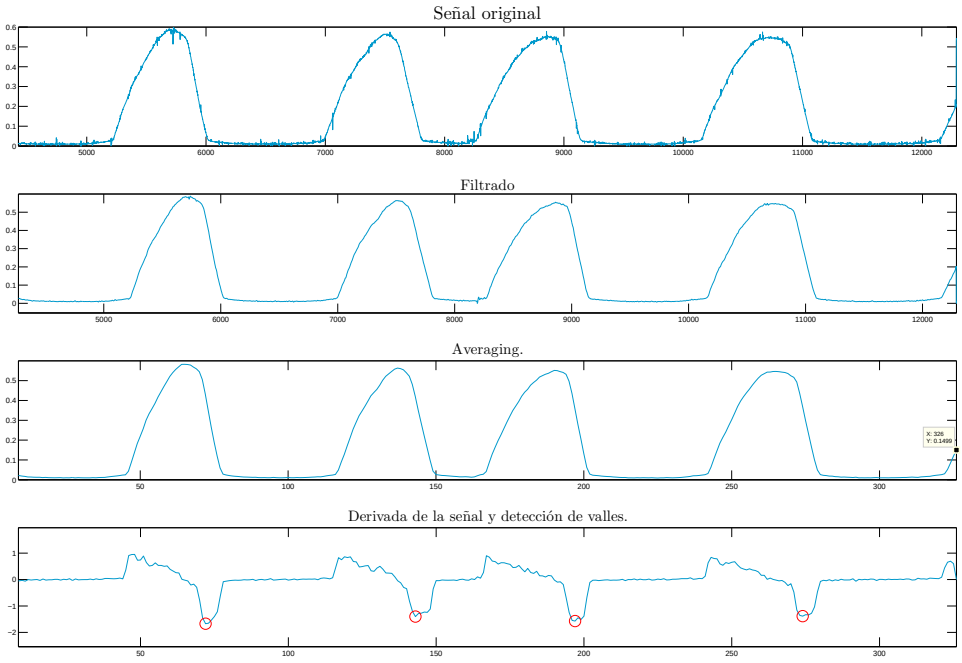


Figura 3.20: Ejemplo de un proceso completo del flujo de aire. En orden descendente: Original, filtrado, media móvil y derivada más detección.

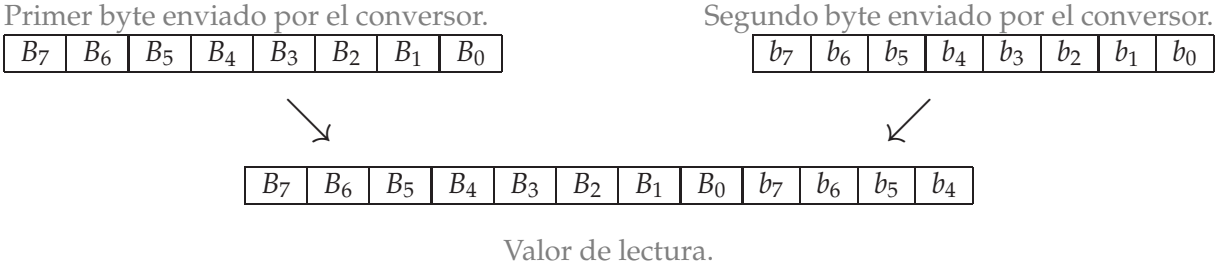
3.5 Muestreo de señales

La *Raspberry Pi* dispone de un *GPIO* por el que recibir datos usando diferentes protocolos, pero los pines de este solo admiten entradas y salidas digitales haciendo necesario el uso de un *ADC*. *Cooking Hacks* dispone de una placa con el *ADC LTC2309* especialmente diseñada para conectar el shield *eHealth* a una *RaspberryPi* y gracias a la biblioteca *ArduPi* disponible en su web [9] se puede programar la toma de datos de manera muy similar a como se haría usando el microcontrolador *Arduino*. El mismo fabricante también proporciona la biblioteca *eHealth* donde se encuentran las clases y funciones que permiten leer los sensores directamente.

3.5.1. Errores de bibliotecas detectados y solucionados

Las primera medidas que se tomaron tenían una precisión de aproximadamente $20mV$. Una precisión que no coincidía con la calculada, ni para *RaspberryPi* ni para *Arduino*.En la figura 3.21 se aprecia la diferencia entre la señal con baja resolución y otra con la resolución adecuada.
Las entradas analógicas de *Arduino* trabajan entre 0 y 5 voltios con una precisión de $10bits$, o lo que es lo mismo, deberíamos poder ver valores en rangos de $4{,}88V$. En cambio la precisión del *ADC* es de $12bits$ que equivaldrían a $1{,}22mV$, mayor precisión incluso que el *Arduino*.

Cuando se probó con un *Arduino* este si que respondía como se esperaba por lo que se decidió abordar el conversor. Cuando el conversor *LTC2309* lee el valor de un pin analógico tiene que convertirlo en digital, esto lo hace mediante el protocolo I_2C. El conversor le envía a la *Raspberry* dos secuencias de un byte cada una con la información. El primer byte representa los $8bits$ más significativos de los 12 que representarán el valor de la lectura y los cuatro bits restantes están almacenados en los cuatro bits más significativos del segundo byte enviado por el *ADC*.



Conociendo esto se descubrió que la precisión de $20mV$ podía ser debido a que los cuatro bits menos significativos en realidad no contenían información. Analizando la biblioteca *ArduPi* se encontró el error. La biblioteca guardaba el mismo valor tanto en el primer como en el segundo byte, teniendo información en los $8bits$ más significativos pero ninguna información en los cuatro bits menos significativos ($\frac{5}{2^{12}} \times 2^4 = 0{,}0195V$). Conocido esto se buscaron las líneas donde se el programa leía la dirección de memoria y se añadió la lectura del segundo byte solucionando el problema.

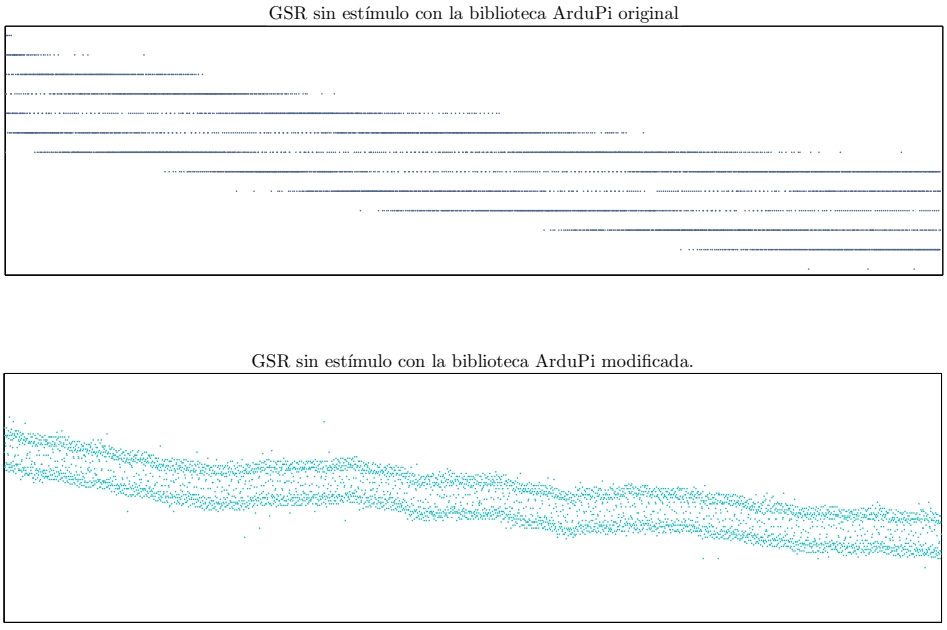


Figura 3.21: Señal antes y después de reparar la biblioteca ArduPi

El otro problema que se detectó fue la conversión de bits a voltaje que realizaba las funciones de la biblioteca *eHealth*. Esta, pensada para *Arduino*, hacia la conversión para una resolución de $10bits$ por lo que solo hizo falta cambiar la resolución a $12bits$.

3.5.2. Lectura de datos

La placa *eHealth* envía la señal del sensor a una salida que es recibida por la *RaspberryPi*, este paso es directo con la función *AnalogRead()* de la biblioteca *ArduPi*.

El primer paso ha sido preparar un bucle que tome cien lecturas y el momento en el que se han tomado, con ellas se calcula el tiempo medio mínimo que tarda tanto el conversor como el sistema operativo en realizar dicha lectura. Sabiendo el tiempo medio mínimo que va a tardar se procede a la toma de datos en sí misma.
La toma de datos sigue la lógica del paso anterior. Se almacena el valor que devuelve el sensor dentro de un vector, la instrucción inmediata guarda la hora exacta con precisión de nanosegundos en otro vector y por último se ejecuta un *sleep()* con el tiempo ajustado al retardo mínimo calculado con el paso previo, aun así, el tiempo entre datos no va a ser constante. Estas operaciones se repiten n veces, siendo n un valor suficientemente grande para que el vector resultante de la *regresión* pueda tener una longitud igual a la potencia de dos usada en la *FFT* (2.3). Los parámetros para cada sensor son diferentes, estos se resumen en la tabla 3.2.

Sensor	Frecuencia de muestreo	Filtro	Media	Derivada	Detección
EMG	500Hz	Paso bajo a 50Hz	Simple a n valores	No	Picos
Flujo de Aire	100Hz	Paso bajo a 30Hz	Simple a n valores	Si	Valles
GSR	300Hz	Paso bajo a 100Hz	Simple a n valores	No	Fuerte pendiente de subida
ECG	1000Hz	Paso banda entre 10Hz y 250Hz	Cuadrática	No	Valor medio de muestra
Posición	1Hz	Ninguno	No	No	Valor directo
Temperatura	1Hz	Ninguno	No	No	Valor directo

Tabla 3.2: Parámetros utilizados para el tratamiento de las distintas señales.

El acelerómetro del *sensor de posición*, en cambio, guarda la información en seis registros de memoria de un byte. Cada eje tiene una resolución de 12*bits*, repartidos en dos registros: El primer registro contiene los 8 bits mas significativos y el segundo registro, empezando por el bit de mayor peso, almacena los cuatro restantes.

Tabla 3.3: Registro de memoria del acelerómetro MMA8452Q.

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x01: X_MSB	XD11	XD10	XD09	XD08	XD07	XD06	XD05	XD04
0x02: X_LSB	XD03	XD02	XD01	XD00	0	0	0	0
0x03: Y_MSB	YD11	YD10	YD09	YD08	YD07	YD06	YD05	YD04
0x04: Y_LSB	YD03	YD02	YD01	YD00	0	0	0	0
0x05: Z_MSB	ZD11	ZD10	ZD09	ZD08	ZD07	ZD06	ZD05	ZD04
0x06: Z_LSB	ZD03	ZD02	ZD01	ZD00	0	0	0	0

Estas operaciones se realizan con funciones definidas en la biblioteca *eHealth*, reduciéndolo a la llamada de una función.

3.6 Aplicaciones desarrolladas

El objeto final de este proyecto ha sido el desarrollo de aplicaciones que hagan saber a un robot el estado fisiológico de un usuario. Pero esto no ha sido un camino directo, ha sido necesario un trabajo previo al desarrollo de las aplicaciones finales con el fin de conocer las limitaciones de las *RaspberryPi* haciendo la función de registrador de datos, la precisión del pack de sensores biomédicos *eHealth* y la rigidez de *ROS* trabajando con la arquitectura del ordenador embebido.
Es por esto que el desarrollo de aplicaciones se ha dividido en dos bloques diferentes, el primero se corresponde con el periodo de toma de contacto con las herramientas a utilizar y el segundo con el desarrollo de las aplicaciones finales en si mismas.

3.6.1. Proceso de pruebas

El primer paso ha sido escribir un programa simple que mediante un bucle *for()* almacene los valores de que devuelve una función de un sensor cualquiera en un vector y se guarda en otro el instante en el que se ha leído. El sensor utilizado para este test ha sido el *GSR* por ser el más fácil de aplicar y para la lectura del tiempo se ha empleado la función $clock\_gettime(REAL\_TIME, structtimespec * tp)$, una función almacena el tiempo real en dos variables de 32$bits$, una para indicar los segundos y otra que representa los nanosegundos. La función $usleep()$ se ha usado para controlar la frecuencia de muestreo, la diferencia entre un $sleep()$ y un $delay()$ reside en la forma de parar la rutina. Mientras que las funciones $sleep()$ pausan el sistema en una rutina hasta que pasa el tiempo indicado, las funciones $delay()$ no pausan todo el sistema si no que ponen la rutina pausada en cola para que se ejecute a partir del tiempo indicado, por lo que las funciones $sleep()$ son útiles para pausas cortas y precisas mientras que las funciones $delay()$ se usan para pausas más largas que requieren menos precisión. Por último ambos vectores se guardan en un archivo de texto para poder analizar los datos facilmente con $MATLAB$®.
Ha sido precisamente en este punto donde se vio el error de resolución que presentaban las librerías *ArduPi* y *eHealth*, ver figura 3.21.

Una vez se tuvo el programa básico para un sensor, este se modificó para poder elegir que sensor se iba a proceder a usar mediante un puntero a funciones, el tiempo durante el que estaría tomando datos, la frecuencia a la que los tomara y el archivo donde los guardaría al terminar. Esto programa se enfocó en que cualquier usuario ajeno al desarrollo del proyecto, por su simplicidad, pudiera obtener datos *en crudo* de cualquier sensor, por eso mismo ha sido

este software el punto de partida para conocer los sensores.

Para ajustar los datos de la señal se usó la regresión mencionada en secciones anteriores que no presentó mayor complicación. Para el filtrado de la señal, en cambio, se descartó la primera opción al no funcionar debidamente. Esta consistía en realizar la *transformada discreta de Fourier* (2.4a) y realizar filtros ideales igualando a cero las componentes correspondientes a las frecuencias a eliminar. A continuación se realizaba la función inversa para volver a tener la señal filtrada en el dominio del tiempo. Este método se acabó descartando, en su lugar se ha usado la biblioteca *DSPFilters* escrita por *Vinnie Falco*. Esta biblioteca permite el uso del varios filtros a diferentes canales de señal al mismo tiempo de forma muy sencilla como se ve en el ejemplo:

```
// Create a Chebyshev type I Band Stop filter of order 3
// with state for processing 2 channels of audio.
Dsp::SimpleFilter <Dsp::ChebyshevI::BandStop <3>, 2> f;
f.setup (3,    // order
         44100,// sample rate
         4000, // center frequency
         880,  // band width
         1);   // ripple dB
f.process (numSamples, arrayOfChannels);
```

La primera opción presentaba la ventaja de que era simple y por lo tanto muy fácil de entender y modificar. En cambio la biblioteca *DSPFilters* es muy completa, mucho más de lo que este proyecto necesitaba, cosa que por otro lado ha aumentado el poder de cálculo necesario.

El último tratamiento que se realiza a la señal es la detección de patrones, concretamente se buscan los picos y valles. Mientras que detectar los picos del *ECG* ha resultado ser sencillo por estar estos muy bien definidos y poder usar el *pulsioxímetro* para tener información redundante, en el caso del *flujo de aire* se detectaron errores debidos al alto rizado de la señal devolviendo falsos picos. Esto se abordó derivando la señal para destacar las pendientes más abruptas, en este caso se vio que lo más fácil de detectar era la pendiente de la espiración, y además, leyendo los datos empezando por el final hacia el principio de la lectura.

En este punto se disponía de los programas para cada uno de los sensores por separado e incluidos en *ROS*. Pero a la hora de ejecutar varios simultáneamente estos se interrumpían al intentar acceder a un bloque de memoria que ya estaba siendo usado por otro, esto se vio que era debido a que la longitud del mensaje reservada por cada *publisher* era de 1000 provocando que este ocupara prácticamente todo el espacio. Al enviar únicamente un dato, el número de pulsaciones por minuto, la proporción de fuerza que realiza, el músculo, si hay sobresalto o

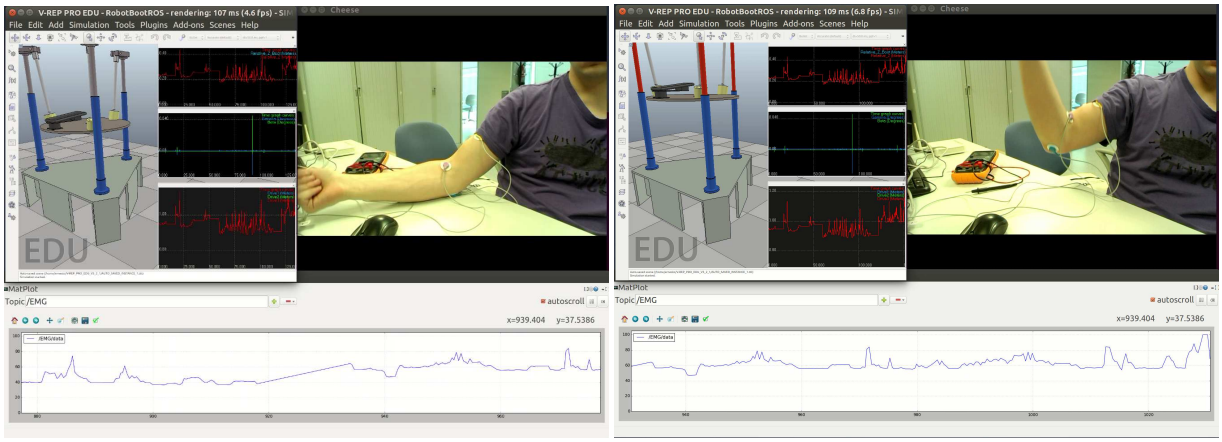


Figura 3.22: Capturas de pantalla de la prueba con el simulador V-Rep.

no, etc. la longitud del mensaje puede ser perfectamente de 1 dejando espacio suficiente para poder ejecutar varios hilos en *multithreading*.

El último test se realizó usando el simulador *V-Rep* y la señal del *EMG*. Esta prueba consistía en ser capaces de controlar la elevación del robot paralelo mediante la contracción de un músculo. *V-Rep* trabaja mediante *scripts* y como se ha mencionado anteriormente tiene una APIs para conectarse con los topics de *ROS*, enviando y recibiendo información al robot simulado.

El nodo de *ROS* encargado del *EMG* se encuentra corriendo en la *RaspberryPi*. Este empieza calibrándose mediante un test de *Máxima contracción muscular (MCV)*, que consiste en saber cual es el valor máximo que el usuario es capaz de generar con la contracción del músculo. Ese valor se usará como referencia para el resto de medidas. La medida que el nodo de *ROS* enviará por el *topic* es el resultado de una lectura durante 0,5*s* que seguidamente es filtrada, por último se hace la media de la señal y se devuelve en valor porcentual del *MCV*. Ese valor medio es recibido por otro nodo alojado en el ordenador principal que transforma el valor porcentual del músculo en una altura a la que el robot pueda posicionarse. Inmediatamente publica en otro *topic* dicho valor gracias a la función *callback* del *subscriber* que recibe el valor del *EMG*. *V-Rep* está subscrito al *topic* de control e inmediatamente este recibe el mensaje, posiciona el robot a una altura proporcional a la contracción muscular realizada. La figura 3.23 representa este mapa de conexiones.

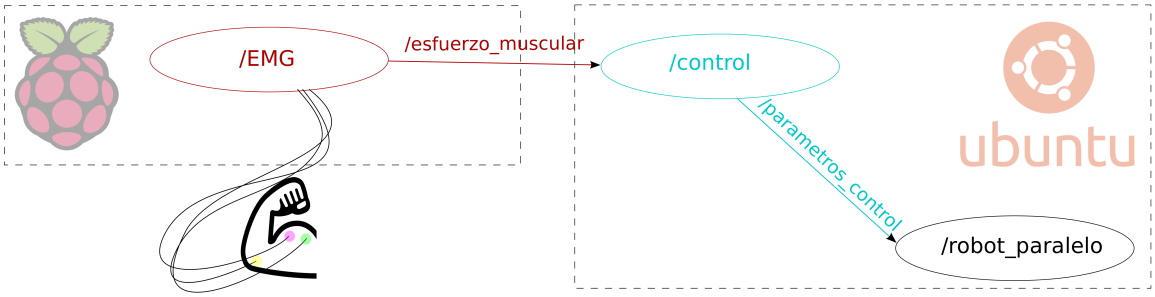


Figura 3.23: Esquema de la simulación del control del robot paralelo usando EMG.

3.6.2. Aplicaciones finales

Aunque el objetivo de esta aplicación es tomar datos y enviarlos a través de *ROS* a cualquier receptor que pueda darles una utilidad, este proyecto ha sido enfocado a enviar dichos datos a un robot concreto. Se trata de un robot paralelo que basará su comportamiento en el estado del usuario obtenido a través de los sensores.

Por el momento el sistema de control del robot es muy simple ya que haría falta un estudio médico detallado para definir perfectamente el comportamiento de este. El sistema se resume en las siguientes ordenes:

- **Pulsaciones por minuto:** Un rápido aumento significa que el usuario está realizando un esfuerzo mayor del previsto. El robot deberá reducir el ritmo de la actividad para que el usuario reduzca las pulsaciones por minuto.
- **Flujo de aire:** Está relacionado con las pulsaciones por minuto. Se trata de información redundante para contrastar con el ECG.
- **Respuesta galvánica de la piel:** Una fuerte pendiente en la conductividad de la piel del usuario significa que el usuario ha sufrido algún tipo de sobresalto. El robot deberá detener inmediatamente la actividad volviendo suavemente a la posición de reposo, se comprobará también la temperatura para intentar conocer el motivo del sobresalto.
- **Posición del paciente:** El paciente debe permanecer siempre en la misma posición. Una posición diferente puede significar que el paciente se ha movido y corre el riesgo de que el robot le lesione por lo que el ordenador lanzará un aviso, o que el usuario ha perdido el conocimiento y por lo tanto el robot se detendrá inmediatamente.
- **Esfuerzo muscular:** El robot dispondrá de diferentes ejercicios. Para que el usuario pueda seguir dichos ejercicios el robot deberá oponer más o menos oposición dependiendo del esfuerzo muscular que este esté ejerciendo.

CAPÍTULO 4

Conclusión

El desarrollo de este trabajo ha supuesto un paso para la introducción de sensores biomédicos de bajo coste en proyectos de robótica. A diferencia de los sensores tradicionales como los sensores de posición, cámaras, y demás, los sensores biomédicos apenas han sido estudiados en aplicaciones con poca o ninguna relación con la medicina ya que hasta hace poco y gracias al *software-libre* y *hardware-libre* no existían alternativas económicas como la tarjeta de expansión *eHealth sensor platform.*

Aun así, el estudio de los parámetros fisiológicos sigue siendo más complejo que las magnitudes físicas ya que estas son suficientemente constantes mientras que los resultados obtenidos con un electrocardiograma, electromiograma o la respuesta galvánica de la piel varían enormemente dependiendo de la persona e incluso pueden variar de un día para otro en un mismo usuario.

Por lo tanto, este proyecto ha supuesto un esfuerzo a la hora de normalizar los resultados obtenido por los sensores sobretodo teniendo en cuenta que se trata de un equipo experimental que no dispone de certificación médica por lo que la salida de los sensores ha de ser tratada prácticamente desde cero para obtener unos resultados que puedan ser útiles, es por esto que el desarrollo de este trabajo se ha centrado mucho en los filtros digitales.

Por otro lado, el uso de tarjetas embebidas como es la *RaspberryPi* y su sistema operativo basado en *Linux* permite tener el control absoluto sobre el hardware pudiendo optimizar al máximo los recursos aunque esto requiere utilizar lenguajes de muy bajo nivel que suponen un esfuerzo extra.

La conclusión final que podemos extraer del proyecto es que el uso de parámetros biomédicos en la robótica es un paso necesario para el desarrollo de esta al permitir un tipo diferente de interacción con las personas. Además el uso de *hardware-libre* facilita el acceso a esta tecnología a cualquier estudiante, científico, desarrollador o incluso artista que decida experimentar con ella. No obstante, llegados a este punto de tener una idea general del conjunto de sensores, el siguiente paso debería ser el estudio individualizado de cada uno de los sensores con el fin de sacarle el máximo partido, ya que estos se desarrollan en contextos diferentes que bien darían para completar un trabajo de investigación por cada uno de ellos.

CAPÍTULO 5

Presupuesto

CÓDIGO	Unidades	Descripción	Rendimiento	Precio	Importe
MO01	h.	Ingeniero de desarrollo	35	28.00 €	980.00 €

Presupuesto de mano de obra ..... 980.00 €

CÓDIGO	Unidades	Descripción	Rendimiento	Precio	Importe
RASPI	uni.	RaspberryPi Model B	1	35.00 €	35.00 €
COMP01	uni.	Ordenador con SO Ubuntu			
EHEAL	uni.	eHealth Sensor Platform complete kit V2.0	1	450.00 €	450.00 €
ASDPI	uni.	RaspberryPi to Arduino shields conecction bridge.	1	40.00 €	40.00 €

Presupuesto de material ..... 525.00 €

NºActividad	CÓDIGO	Unidades	Descripción de unidades de obra	Rendimiento	Precio	Importe
1	INST01		**Instalación del framework.** Instalación de ROS en un ordenador y en la RaspberryPi.	1	116.52 €	116.52 €
2	INST02		**Software de procesado de señal.** Programa para tratar las señales.	1	233.03 €	233.03 €
3	INST03		**Conjunto de módulos de sensores.** Programas para cada uno de los sensores.	1	1043.45 €	1043.45 €
4	INST04		**Módulo de control** Programas que funcionan de controlador y alarma.	1	112.00 €	112.00 €

Presupuesto de material ..... 1505.00 €

NºActividad	CÓDIGO	Unidades	Descripción de unidades de obra	Rendimiento	Precio	Importe
1	INST01		**Instalación del framework.** Instalación de ROS en un ordenador y en la RaspberryPi.			
	MO01	h	Ingeniero de desarrollo	4	28 €	112.00 €
	RASPI	uni.	RaspberryPi Model B	0.13	35 €	4.52 €
	COMP01	uni.	Ordenador con SO Ubuntu			
2	INST02		**Software de procesado de señal.** Programa para tratar las señales.			
	MO01	h	Ingeniero de desarrollo	8	28 €	224.00 €
	RASPI	uni.	RaspberryPi Model B	0.26	35 €	9.03 €
	COMP01	uni.	Ordenador con SO Ubuntu			
3	INST03		**Conjunto de módulos de sensores.** Programas para cada uno de los sensores.			
	MO01	h	Ingeniero de desarrollo	19	28 €	532.00 €
	RASPI	uni.	RaspberryPi Model B	0.61	35 €	21.45 €
	COMP01	uni.	Ordenador con SO Ubuntu			
	EHEAL	uni.	eHealth Sensor Platform complete kit V2.0	1	450 €	450 €
	ASDPI	uni.	RaspberryPi to Arduino shields conecction bridge.	1	40 €	40 €
4	INST04		**Módulo de control.** Programas que funcionan de controlador y alarma.			
	MO01	uni.	Ingeniero de desarrollo	4	28 €	112.00 €

Presupuesto de ejecución material . . . . . 1505.00 €

Gastos generales 13 % . . . . . 195.65 €

Beneficio industrial 6 % . . . . . 90.30 €

Presupuesto por ejecución de contrata . . . . . 1790.95 €

I.V.A. al 21 % . . . . . 370.10 €

Presupuesto de licitación . . . . . 2167.05 €

Asciende el presente presupuesto a la expresada cantidad de:
DOS MIL CIENTO SESENTA Y SIETE EUROS CON CINCO CÉNTIMOS.

Bibliografía

[1] Smith, Steven W.. *The Scientist and Engineer's Guide to Digital Signal Processing* Segunda edición. California Technical Publiching, San Diego, U.S.A. 1999.

[2] Torres, Fernando et al. *Robots y sistemas sensoriales.* PEARSON EDUCACIÓN, S.A. Madrid, España, 2002.

[3] Valera, Ángel et al. *Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica.* ScienceDirect, Revista Iberoamericana de Automática e Informática industrial 00 (2014) 1–14.

[4] Arduino. [Web en línea]. [Última consulta 26-06-2015]. http://www.arduino.cc/

[5] RaspberryPi. [Web en línea]. [Última consulta 26-06-2015]. http://www.raspberrypi.org

[6] BeagleBoard. [Web en línea]. [Última consulta 26-06-2015]. http://www.beagleboard.org/

[7] Konrad, Peter. *The ABC of EMG: A practical introduction to kinesiological electromyography*. Noraxon INC, Arizona, U.S.A., 2005.

[8] Ritter, Arthur B. et al. *Biomedical Engineering Principles*. 2ª ed. Taylor and Francis Group, LLC.

[9] Cooking Hacks. *e-Health Sensor Platform V2.0 for Arduino and Raspberry Pi [Biometric / Medical Applications]* [Web en línea]. [Consulta: 1-6-2015]. https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical.

[10] ROS. [Web en línea]. [Última edición 09-05-2015]. http://wiki.ros.org/

[11] CMake. [Web en línea]. [Consulta: 20-04-2015]. http://www.cmake.org.

APÉNDICE A

Figuras EMG

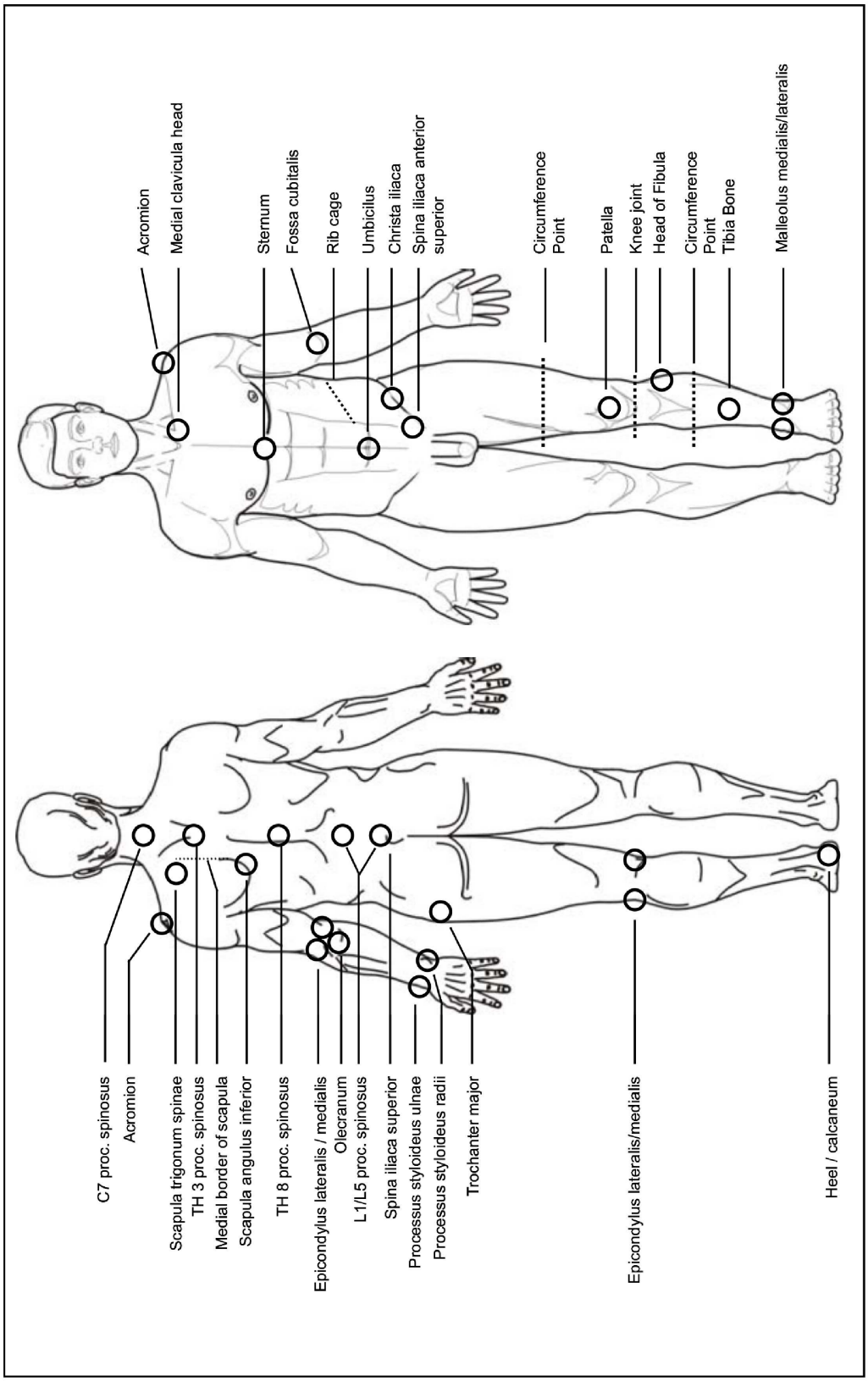


Figura A.1: Puntos de referencia eléctrica en el cuerpo humano.

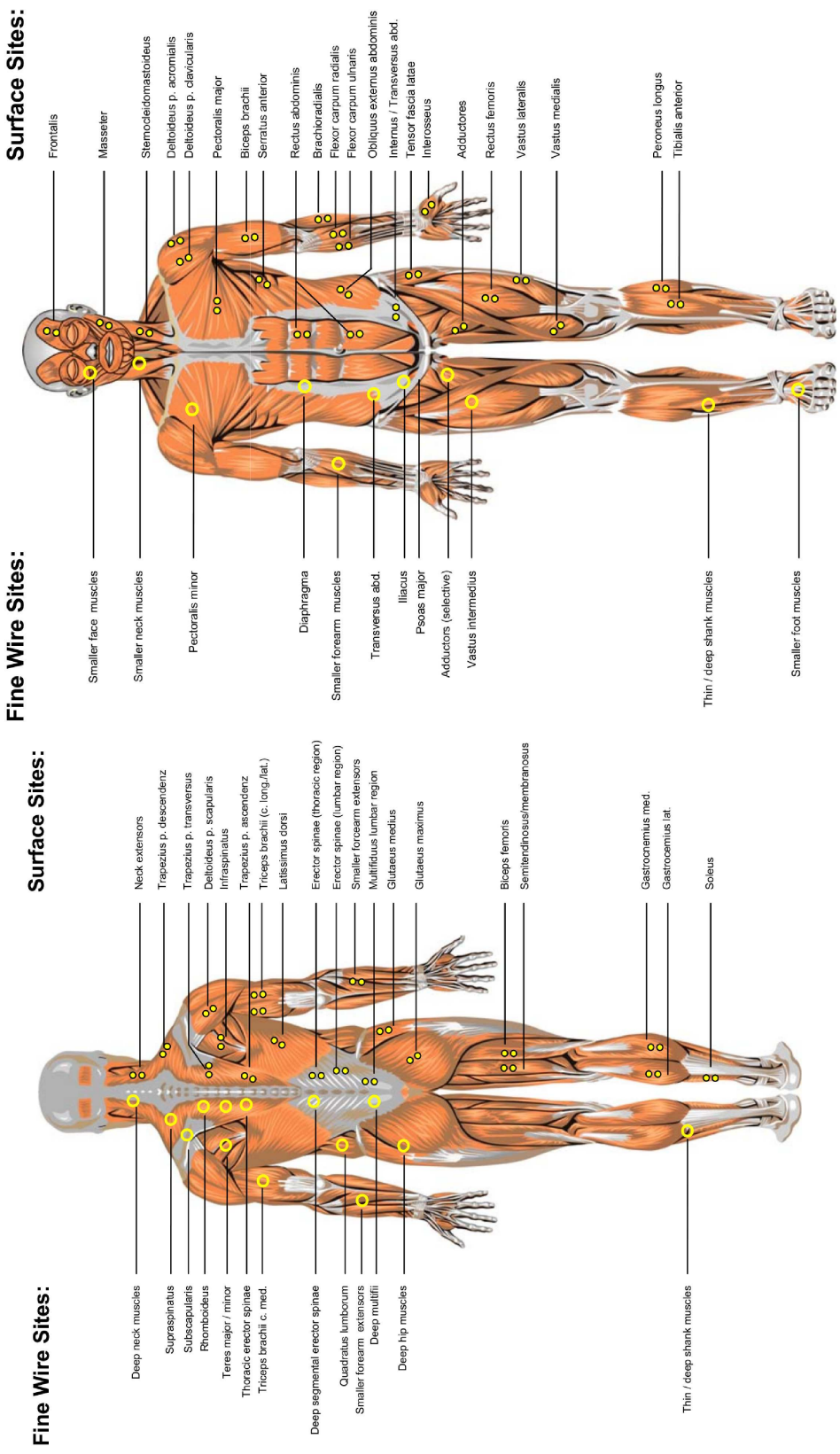


Figura A.2: Posiciones comunes de electrodos. Electrodos superficiales a la derecha y electrodos de aguja a la izquierda. Extraído de [7]

APÉNDICE B

Esquema eHealth sensor platform

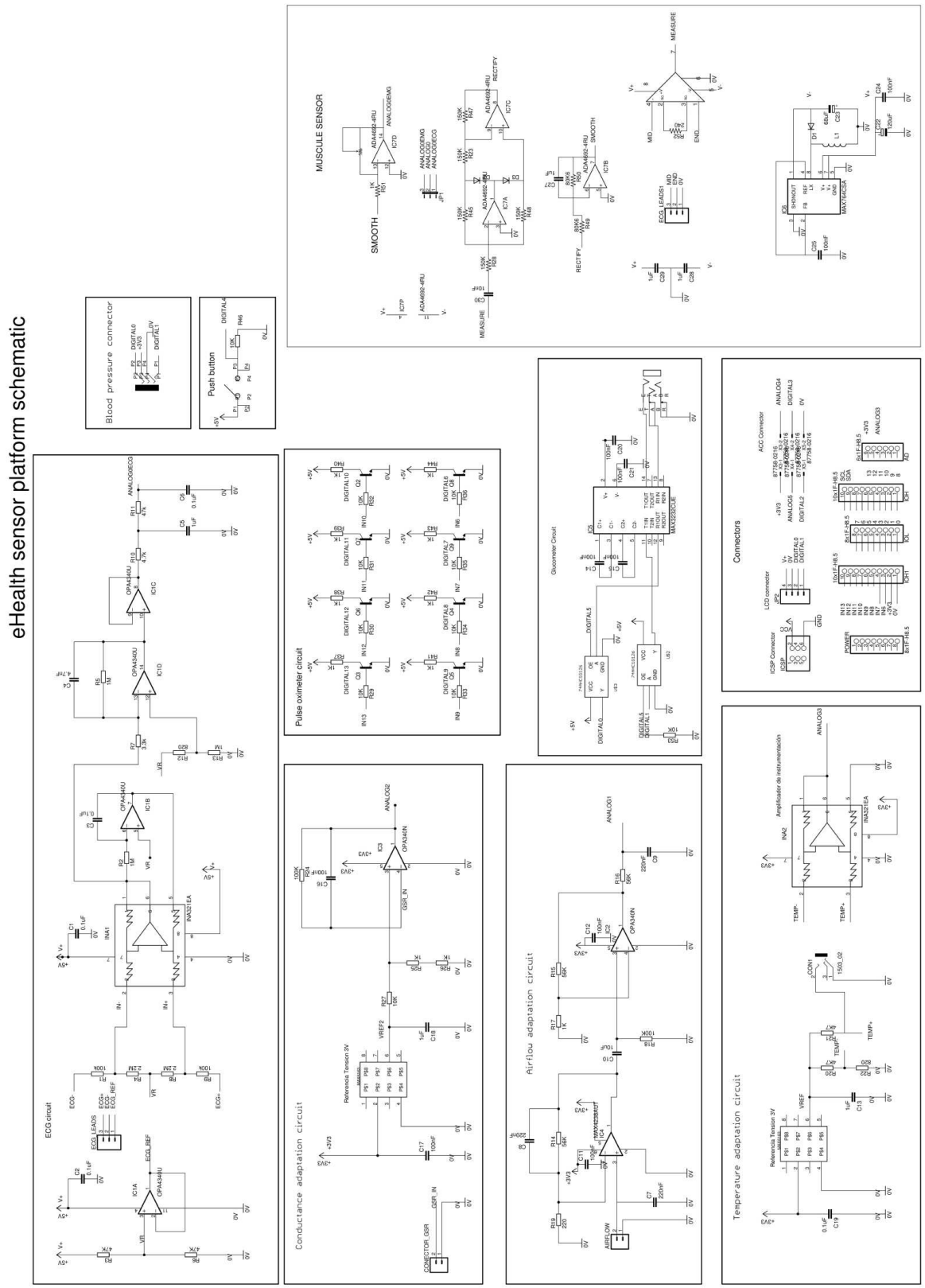


Figura B.1: Esquema electrónico del shield eHealth sensor platform V2.0

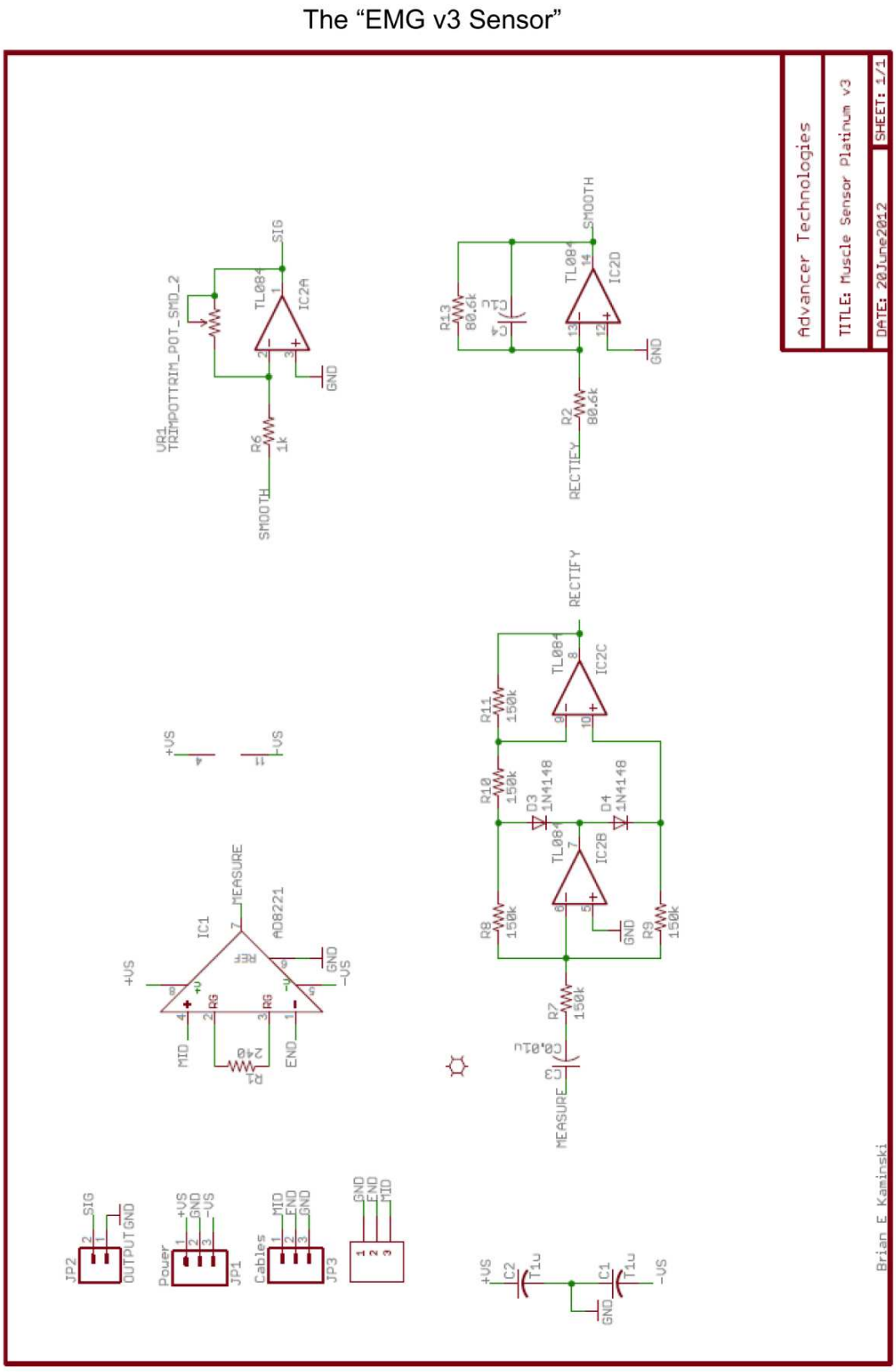


Figura B.2: Esquema electrónico del sensor EMG