Universitat
de les Illes Balears

Título: Mando multimodal para el estudio de neuropatologías

AUTOR: Edgar Tomás Rucián

Memoria del Trabajo de Fin de Máster

Máster Universitario en Ingeniería Industrial

de la

UNIVERSITAT DE LES ILLES BALEARS

Curso Académico *2018-2019*

Fecha: 19/03/2019

Nombre Tutor del Trabajo: Dr. Francisco José Perales López

Nombre Cotutor: Dr. José María Buades Rubio

Abstract

La rehabilitación fisioterapéutica de enfermedades neuropatológicas como la Esclerosis Múltiple y el Parkinson suponen hoy en día, un proceso largo y tedioso. Por este motivo se propone el uso de un dispositivo capaz de afrontar este proceso de una forma lúdica. De esta forma el paciente puede afrontar su problema sin necesidad de pensar que está siendo rehabilitado.

Tabla de Contenidos

Lista de tablas

Lista de figuras

Capítulo 1

Introducción

El presente proyecto consiste en la descripción de los pasos realizados para la elaboración de un mando multimodal que sirva para el estudio de enfermedades relacionadas con la motricidad y neuropatías.

Este tipo de enfermedades suponen una alteración de la capacidad de una persona para desplazarse o manipular objetos, lo que supone una alteración en el desarrollo personal y social [1].

La fisioterapia ha sido lo que ha permitido hasta el momento realizar labores de rehabilitación utilizando técnicas de fuerza, resistencia, elasticidad, entre otras [2]. En algunos casos también se ha optado por la utilización de férulas, bastones o sillas de ruedas.

Sin embargo, con las nuevas tecnologías, una nueva forma de rehabilitación se presenta. Utilizando dispositivos electrónicos diseñados específicamente para estas tareas y combinándolos con videojuegos serios es posible obtener mejoras en los pacientes con neuropatías de forma entretenida y lúdica. De esta forma se consigue una mejora de su condición física y psicológica en los usuarios.

1.1. Objetivos

El objetivo del presente proyecto supone el desarrollo del dispositivo electrónico, llamado a partir de ahora mando multimodal, que pueda ser compatible con el uso de juegos serios para mejorar la condición y atención del paciente.

Dado que el dispositivo pretende ser usado para mejorar el movimiento de la mano, muñeca y antebrazo, se deben incorporar sensores que permitan controlar la acción del paciente. Las características del prototipo serán:

- Capacidad de detección de fuerza
- Medición de la orientación
- Monitorización biométrica
- Inalámbrico

1.1.1. Sensores de fuerza

Se incorporarán sensores de fuerza, tanto para medidas de precisión con los dedos como para la fuerza de agarre del usuario. De esta forma se puede parametrizar cuanta fuerza ejerce el usuario, y, registrando los valores a lo largo de las sesiones, comprobar la mejoría durante el proceso de rehabilitación.

1.1.2. Sensor de orientación

También será necesario incluir un sensor que permita saber el movimiento que realiza el paciente mediante la orientación del dispositivo. Para ello se utilizará una Unidad de Medición Inercial (IMU por sus siglas en inglés).

1.1.3. Sensores biométricos

Además de los sensores mencionados también se diseña el concepto del mando multimodal con sensores biométricos para controlar el estrés que supone la actividad del

sujeto. Entre ellos se incluye un sensor de pulso cardíaco (ECG) que permita observar la actividad cardiovascular. Un sensor de electromiografía (EMG) de forma que se pueda conocer la actividad de los músculos del antebrazo. Y finalmente un sensor de actividad electro-dermal (EDA), útil para saber la ansiedad física durante la actividad al controlar la sudoración de la piel.

1.1.4. Capacidad inalámbrica

Para finalizar y facilitar la comodidad de uso del dispositivo se incluye un módulo bluetooth para enviar los datos al terminal sin necesidad de cable. Para garantizar la autonomía del mando multimodal se instalará también una batería recargable.

1.2. Aplicación

Se ha mencionado a lo largo de este capítulo la función de ayuda al estudio de neuropatías. Sin embargo, hay otras áreas en las que tiene cabida el dispositivo como el área de entretenimiento. El objetivo final es combinar ambas áreas para conseguir una rehabilitación lúdica mediante juegos serios.

De esta forma se pueden hacer dos distinciones:

1.2.1. Medicina

Esta área está relacionada con el estudio de enfermedades. Entre las neuropatías más conocidas se encuentras la Esclerosis Múltiple y la enfermedad del Parkinson.

También se incluyen aquellas enfermedades relacionadas con la motricidad y lesiones musculares.

Los sensores de que dispone el prototipo permitirán realizar el estudio adecuado para que el médico especialista a cargo del proceso de rehabilitación pueda hacer un seguimiento correcto.

1.2.2. Entretenimiento

El mando multimodal también puede ser utilizado como controlador de videojuegos. Gracias a sus sensores es posible crear una experiencia de juego no existente en la actualidad, gracias, por ejemplo, a los sensores de fuerza para saber cuándo agarrar un objeto. Los sensores biométricos por otro lado pueden permitir un mayor *feedback* para crear una mayor experiencia.

Como se ha mencionado anteriormente, combinando ambas áreas es posible tener un proceso de rehabilitación lúdico en el que los pacientes puedan mejorar su condición mediante los videojuegos.

1.3. Aspectos médicos y terapéuticos

En este apartado se explicarán en detalle algunas de las enfermedades que se pueden encontrar en los futuros usuarios del dispositivo, principalmente aquellas neuropatías que tienen relación con la movilidad, como la esclerosis múltiple, o EM [3] [4] [5] [6], y el Parkinson [4], así como sus síntomas, tratamiento, causas y pronósticos.

1.3.1. Esclerosis múltiple (EM)

La EM es una enfermedad desmielinizante, crónica, autoinmune e inflamatoria que afecta a todo el sistema nervioso central, formado principalmente por neuronas. Es la principal causa de discapacidad neurológica entre adultos jóvenes.

La función de las neuronas es transmitir señales por el cuerpo a través de sus axones. Estos axones están recubiertos por un aislante compuesto de mielina. Un símil sería imaginar un cable de teléfono, donde se puede observar el interior del cable (axón) y el recubrimiento del cable (mielina). En caso de haber una distorsión en alguno de los dos, la señal no se transmitiría correctamente. Si eso pasara se verían afectadas las señales responsables de funciones como la vista, la marcha, la memoria, los sentidos y el equilibro.

La EM se define como enfermedad autoinmune, debido a que el sistema inmunitario, aquel encargado de proteger el organismo contra las infecciones, se altera y confunde el propio tejido como un agente invasor externo. Esto produce ataques que dañan la capa protectora de los axones. Los ataques se pueden reparar mediante “parches” de tejido cicatricial, lo que se conoce con el nombre de “esclerosis”. Se denomina “múltiple” porque afecta a diferentes zonas del sistema nervioso central.

El cuadro clínico suele manifestarse en cuadros agudos neurológicos, como brotes o recaídas con remisión posterior. En algunos casos suelen ser progresivos hasta la muerte.

Síntomas

Esta enfermedad afecta de forma diferente a cada persona. Algunos tienen muchos síntomas y otros pocos. Estos síntomas van desde algunos muy graves hasta otros más

leves, pudiendo permanecer en el afectado durante mucho tiempo o en algunos casos desaparecer.

Entre los síntomas más frecuentes que se pueden diagnosticar se encuentran:

- **Síntomas motores:** pérdida de fuerza, dificultad para caminar, para la coordinación, rigidez, espasmos musculares, problemas para el habla.
- **Síntomas visuales**: visión borrosa por un ojo, visión doble.
- **Síntomas sensitivos:** sensación de hormigueo, entumecimiento, acorchamiento, quemazón, tirantez de cara, tronco o extremidades.
- **Síntomas genitourinarios o sexuales:** urgencia e incontinencia urinaria, estreñimiento, dificultad para la erección, disminución de la lubricación vaginal.
- **Síntomas mentales:** alteraciones del estado de ánimo, pérdida de memoria, dificultad para concentrarse.
- **Síntomas paroxísticos:** neuralgia del trigémino (dolor en la cara repetido e intenso), episodios breves y repetidos de dificultad para hablar, inestabilidad, hormigueos o calambres.
- **Fatiga:** sentirse cansado, sin demasiada energía, que no se correlaciona con el grado de actividad realizada, es una queja frecuente.

Es importante tener en cuenta la sintomatología emocional y afectiva. A la EM no sólo afectan los síntomas mencionados anteriormente. Hay trastornos conductuales que pueden aumentar la discapacidad y reducir la participación social y laboral. La depresión supone

uno de los síntomas debilitantes más comunes. También aparecen emociones como la ira y la ansiedad, así como el deterioro del autocuidado y el autoconcepto.

Tratamiento

Se pueden diferenciar varios tipos de tratamiento de la esclerosis múltiple:

- Tratamiento de los brotes agudos
- Tratamiento para modificar la historia natural de la enfermedad
- Tratamiento sintomático
- Tratamiento rehabilitador

El tratamiento de los brotes agudos se realiza con corticoides intravenosos (Metilprednisolona) a altas dosis durante 3 a 5 días.

La EM no tiene cura a día de hoy, sin embargo, existen tratamientos que ayudan a cambiar el desarrollo de la enfermedad. Los Inmunomoduladores son capaces de modular la respuesta del sistema inmunitario para reducir la posibilidad de tener brotes o recaídas. Los Inmunomoduladores se consideran tratamientos de primera línea.

También se encuentran los tratamientos Inmunosupresores, llamados así porque disminuyen las defensas, de forma que se evita la producción de inflamaciones, por el contrario, también se pierde la capacidad de combatir infecciones, algo que puede suponer un riesgo para el paciente. Se consideran tratamientos de segunda línea o en el caso de la EM agresivas desde el inicio.

Actualmente, la inversión en investigación de nuevos fármacos en este ámbito es muy elevada en todo el mundo.

El tratamiento sintomático se basa en el manejo de diferentes síntomas que van apareciendo a lo largo de la evolución de la enfermedad. Puede requerir la ayuda de otros profesionales.

Por último, se encuentra también el tratamiento rehabilitador que no implica el uso de fármacos. Entre ellos se encuentra la logopedia, fisioterapia y la terapia ocupacional. También se tiene muy en cuenta la alimentación y el ejercicio físico como nadar y caminar, así como la eliminación de malos hábitos como el tabaco y el alcohol.

Causas

Actualmente se desconoce qué produce la esclerosis múltiple. Pero parece que hay varios factores implicados:

- **Factores genéticos**: No se trata de una enfermedad hereditaria, sin embargo, hay estudios genealógicos que han demostrado que existe una predisposición genética a padecer EM. Aunque por el momento no se conocen con exactitud los genes implicados.
- **Factores ambientales**: Actualmente son desconocidos. Los agentes más estudiados son el virus Epstein-Barr, virus común en la población. Fumar también aumenta el factor de riesgo, así como la localización geográfica del individuo. Cuanto más cerca del ecuador, menor es el riego de padecer EM. También se han relacionado los niveles bajos de vitamina D con el desarrollo la enfermedad.

Debido a que las causas que provocan EM son desconocidas no es posible por el momento prevenir el desarrollo de esta enfermedad. Sin embargo, mantener un estilo de vida saludable ayuda a sobrellevarla de la mejor manera posible en caso de padecerla.

Pronóstico

El pronóstico en una enfermedad crónica de estas características es incierto. La supervivencia ronda los 35 años, aunque existen estudios que demuestran que el diagnóstico temprano y el tratamiento precoz de la enfermedad pueden ayudar a mantener firmes los síntomas de la EM, a reducir la tasa de brotes y a retrasar la progresión de la discapacidad, mejorando por todo ello la calidad de vida del paciente a largo plazo.

1.3.2. Parkinson

El Parkinson es una enfermedad neurodegenerativa del sistema nervioso, de alta prevalencia en la población y la segunda más frecuente después del Alzheimer.

Su aparición está relacionada con el envejecimiento, causa conocida de las enfermedades neurodegenerativas. Se suele iniciar entre los 60 y los 69 años. Afecta a 3 o 4 casos por cada 100.000 habitantes en menores de 40 años y más de 500 por cada 100.000 en mayores de 70 años.

Síntomas

Los síntomas principales de la enfermedad de Parkinson son las siguientes:

- Temblor en reposo. De todas las manifestaciones es la menos invalidante. Se trata del primer síntoma en el 50-70% de los casos. La ansiedad y el cansancio pueden agravar y aumentar los síntomas de esta enfermedad.
- Rigidez de los músculos flexores. El estrés puede aumentar los efectos. También puede superponerse con el temblor. Resulta molesta y dolorosa.
- Bradiniscesia. Es el síntoma más discapacitante. Produce lentitud en los movimientos, además de retrasos en las órdenes del cerebro a los músculos. Este síntoma supone un inconveniente para realizar tareas cotidianas como vestirse, comer, asearse, etc.
- Inestabilidad postural. Supone problemas para el equilibrio de forma gradual. En estados avanzados de la enfermedad se caracteriza por la imposibilidad de iniciar la marcha o reaccionar ante un obstáculo.

Tratamiento

Existen métodos farmacológicos actualmente que permiten combatir de forma eficaz los síntomas de esta enfermedad, a diferencia de otras enfermedades neurodegenerativas.

Este tipo de tratamiento permite tener calidad de vida durante un tiempo prolongado al afectado, sin embargo, es necesario individualizarlo para cada persona. Para ello hacen falta dos factores, la experiencia del profesional y una buena información por parte del paciente de los síntomas que padece y de las limitaciones que le suponen en su actividad diaria.

Además de la medicación también es fundamental el ejercicio físico mantenido, ya que potencia la independencia por más tiempo. Entre las actividades más recomendadas se encuentran caminar, ejercicio muscular como pilates o yoga y el Tai-chi para la coordinación y la estabilidad.

Causas

No se conocen las causas que producen el Parkinson. Se le involucran ciertas sustancias tóxicas y además mutaciones genéticas. La principal causa se suele relacionar con factores genéticos y medioambientales.

Uno de los problemas que se encuentra para el diagnóstico de esta enfermedad sucede en los ancianos. Al tratarse de un proceso gradual sienten que se trata de causas naturales del envejecimiento, por lo que no acuden al médico y supone un empeoramiento de su salud al no controlar la enfermedad. El temblor en reposo es el síntoma más fácil de identificar para saber que se trata de una situación anormal.

Desde su descripción la clínica de la enfermedad está representada por problemas motores: temblor, bradicinesia, rigidez, y en la evolución, problemas de la marcha y estabilidad. Estos signos son, además, a día de hoy, la clave para el diagnóstico, es decir el diagnóstico es “clínico” y no existe ninguna prueba diagnóstica que sustituya la información que proporcionan los síntomas y la exploración del paciente.

Sin embargo, también se deben tener en cuenta los síntomas no-motores, pues pueden iniciarse antes que los síntomas motores e incluso suponer una causa mayor de discapacidad.

Pronóstico

Actualmente no se conoce una forma de prevenir la enfermedad, pero hay estudios que demuestran que la prevalencia es menor en aquellas personas que consumen tabaco y café, aunque otros riesgos asociados al tabaco descartan su recomendación.

En resumen, gracias a la existencia de los diferentes tratamientos, tanto fármacos como ejercicio, que proporcionan que el paciente tenga una buena calidad de vida, y a su lenta progresión, es posible afrontar de forma más eficaz la evolución de la enfermedad del Parkinson.

Capítulo 2

Estado del arte

En este capítulo se describirán las tecnologías existentes similares al trabajo que se explica en el presente proyecto. Se hará una revisión del mando de fuerza V1.0 de la *Universitat de les Illes Balears* (UIB), modelo anterior al dispositivo multimodal que se presenta. Posteriormente se comentará la patente US2017291066A1 que describe un dispositivo de agarre para proveer información sobre el usuario basada en la interacción del usuario con el dispositivo. Finalmente se hará un análisis de la tecnología que incorpora el controlador *Wiimote* de Nintendo.

2.1. Mando de fuerza

El mando portátil para la detección de movimiento y fuerza de prensión es un dispositivo adecuado para el tratamiento de enfermedades. Entre ellas se encuentran enfermedades neurodegenerativas como la Esclerosis Múltiple y el Parkinson.

La invención [9] propone un dispositivo de PVC integrado por sensores tales como:

- Sensor de movimiento (IMU) de 6 grados de libertad.
- Sensores de fuerza resistivos para los dedos.
- Sensores de fuerza para medir la fuerza de prensión.

El aparato (figura 2.1) está formado por dos cuerpos. En el primero se encuentran cuatro sensores resistivos donde se sitúan los dedos (6, 7, 8 y 9). El segundo es un apoyo para la

palma de la mano (3). Ambos unidos por los extremos donde se sitúan las galgas extensiométricas (4 y 5).

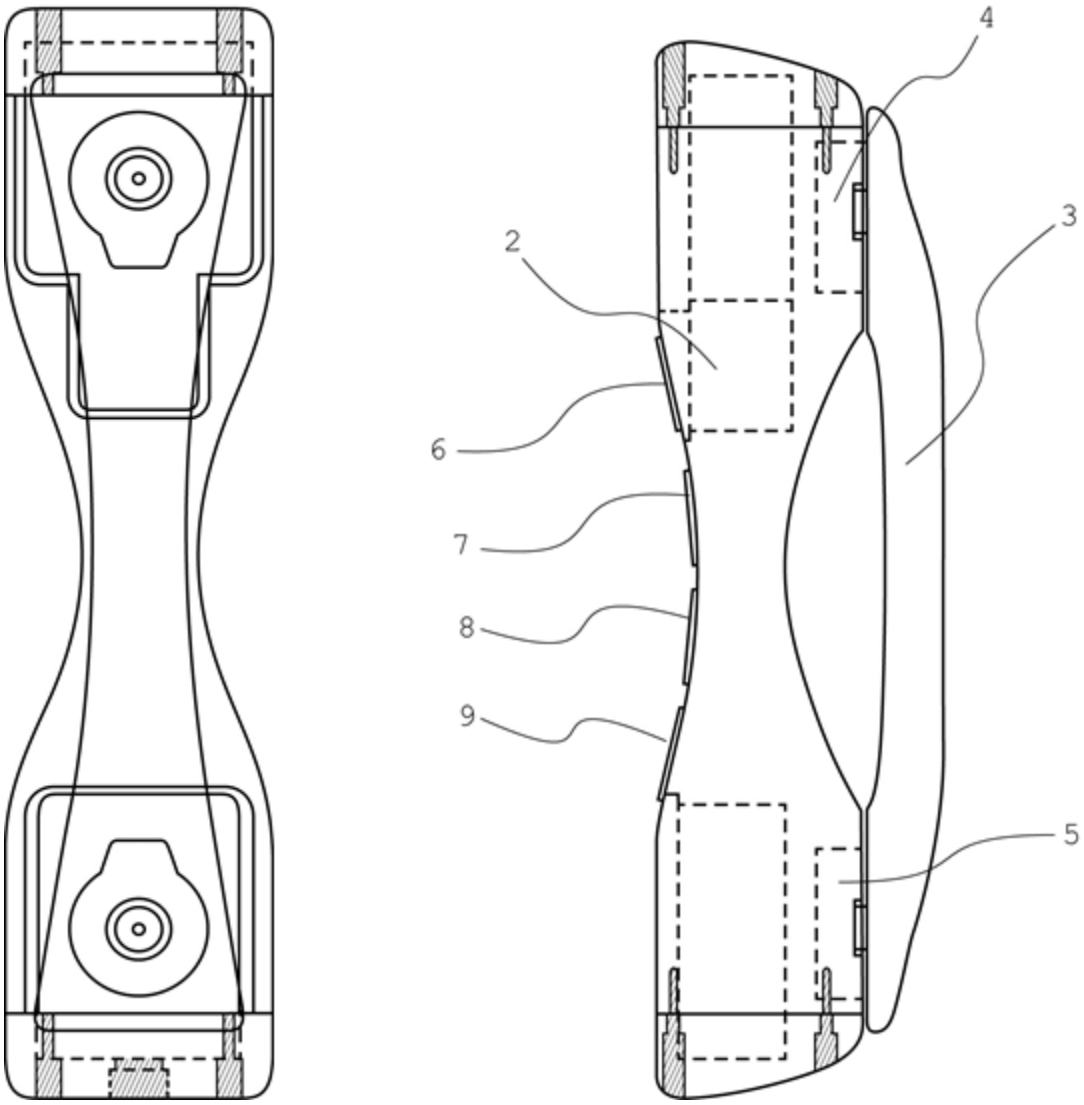


Figura 2.1. Mando de fuerza. Fuente [9]

En el interior del dispositivo también se encuentran una unidad de acondicionamiento de señal, un microprocesador y un dispositivo de envío de datos bluetooth (2).

El rango de aplicaciones varía desde videoconsolas, instrumentos musicales, controles médicos, etc.

En el área médica puede utilizarse para el tratamiento de neuropatías y tendinitis. Gracias a sus sensores de fuerza y movimiento el especialista y el paciente puede realizar un seguimiento del proceso de rehabilitación.

En el área de entretenimiento puede ser utilizado como mando control de videoconsolas. De igual modo es compatible con sistemas informáticos como Tablets y Smartphones.

Este dispositivo fue diseñado por la Unidad de Gráficos y Visión e Inteligencia Artificial (UGIVIA y el grupo de ingeniería electrónica) de la *Universitat de les Illes Balears* como primera versión de mando para el tratamiento de enfermedades. El presente proyecto consiste en una nueva versión de este modelo.

2.2. Nintendo Wiimote

El dispositivo Wiimote o Wii Remote (figura 2.2) es un aparato parecido a un mando de televisión creado por Nintendo para la consola Wii.

Además de los botones también contiene un acelerómetro de 3 grados de libertad, una cámara infrarroja de alta velocidad, un altavoz, un motor de vibración y conexión inalámbrica mediante bluetooth [10]. Sus funciones se especifican a continuación.

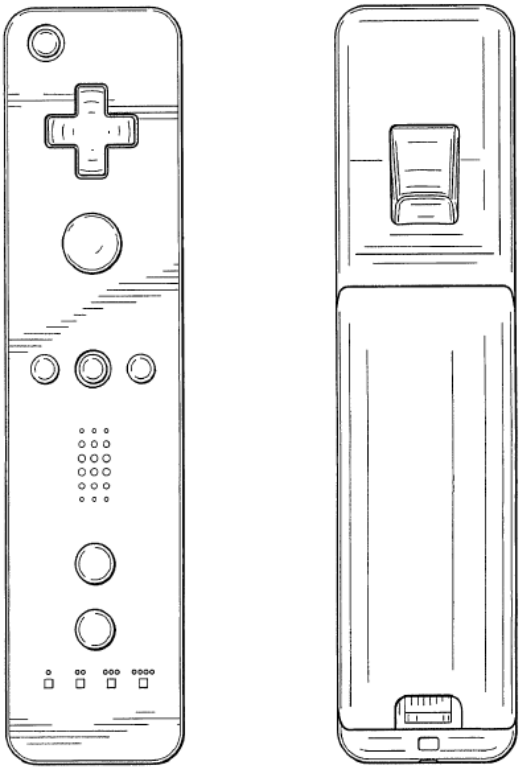


Figura 2.2. Vistas Wiimote. Fuente [11].

Cámara infrarroja

El sensor de la cámara está situado en la parte superior del aparato. El chip integra un motor de detección de objetos de alta velocidad. La resolución es de 1024x768 píxeles a una frecuencia de 100 Hz y 45 grados de visión. Se función es recibir los rayos infrarrojos, de forma que se puede definir la posición del cursor el Wiimote.

Acelerómetro

El acelerómetro que incluye el Wiimote es el modelo ADXL330 de 3 grados de libertad. Proporciona la capacidad de captar el movimiento.

Botones

Contiene 12 botones. Cuatro de ellos en forma de cruz, uno en la parte posterior en forma de gatillo y otros siete botones distribuidos de tal forma que puede ser utilizado igual por zurdos y diestros.

Motor de vibración

Sirve para proporcionar *feedback* al usuario durante el juego. Tiene dos estados ON/OFF, aunque se puede controlar la intensidad mediante uno de los botones.

Sensores de luz

El dispositivo contiene cuatro LEDs en la parte inferior que se activan en función del número de dispositivos conectados. De esta forma se puede saber el número de jugadores presentes.

Altavoz

El dispositivo contiene un pequeño altavoz que al igual que el motor de vibración proporciona *feedback* al usuario durante el juego. El sonido se emite a 4 KHz, similar al de un teléfono.

Conexión inalámbrica

El protocolo de comunicación que integra el dispositivo es Bluetooth. Esto permite además de la comunicación con la consola Wii, la posibilidad de comunicarse con otros dispositivos bluetooth, ampliando el rango de aplicaciones.

Memoria interna

El Wiimote contiene una pequeña memoria flash de 5,5 Kbytes para mantener los ajustes personalizados de cada usuario, así como para mantener el perfil de cada jugador, llamado *Mii* sea cual sea la videoconsola.

Puerto de expansión

En la parte inferior del mando se encuentra un puerto de expansión para conectar otros accesorios creados específicamente para ser compatibles con el Wiimote. Un ejemplo es el Nintendo Nunchuck. La conexión se realiza mediante el protocolo I2C a 3,3V y 400KHz.

Batería

La autonomía del dispositivo viene dada por dos pilas tipo AA. En función del número de accesorios conectados la batería tiene una duración de entre 20 y 40 horas.

Aplicaciones personalizadas

Aunque se trata de un dispositivo diseñado para la consola Nintendo Wii, es posible utilizarlo, gracias a sus componentes, con otro tipo de aplicaciones. Se puede utilizar conectándolo mediante bluetooth a un PC y utilizarlo como controlador.

2.3. Sistema para la construcción de la actividad basada en la recopilación de datos

Esta invención [12] supone una interfaz de agarre para la obtención de información de un usuario. La interacción varía según el agarre y la aplicación. Para ello se integran una gama de sensores de movimiento, salud, medioambiente, entre otros. Todos ellos actuando en la zona de la mano. Entre sus aplicaciones se encuentran deportes, equipamiento deportivo, monitorización de salud, fisioterapia, etc.

Diseño mecánico

La envoltura mecánica está diseñada para incluir todos los sensores y componentes necesarios. Incluye sensores externos de superficie, sensores internos, una batería recargable para alimentar todos los componentes, LEDs programables, *feedback* de fuerza y galgas. Para la conexión entre la electrónica se incluye un bus interno y un puerto de conexión para incluir otros accesorios para el agarre.

Hardware

La electrónica interior incluye un microcontrolador con circuitería extra, entre la que se incluye memoria, red, administración de potencia y respuesta de audio.

Los sensores internos de que dispone son acelerómetros, giroscopios, magnetómetros, sensor de temperatura, de humedad, calidad del aire, luz ambiente y fuerza de presión.

Los sensores externos son superficiales para hacer contacto con la piel, entre ellos se encuentran sensores de fuerza, pulso, ECG, temperatura corporal y respuesta galvánica.

El método de conexión es mediante bluetooth con soporte opcional de GPS y WiFi en función de la necesidad.

Al poderse usar en un gran rango de aplicaciones no tienen por qué incluirse todos los sensores. El dispositivo de agarre se crea de forma personalizada para cada tipo de actividad. Este dispositivo se integra en accesorios que requieran agarre, como por ejemplo una raqueta de tenis, un palo de esquí, una barra de fitness, cuchillos, etc.

2.4. Conclusión

Se han presentado tres tecnologías de similares características a las del presente proyecto. El objetivo es realizar una actualización del hardware de la primera invención descrita (mando de fuerza), incorporando sensores que permitan obtener más información de la persona que lo utilice, de forma parecida a lo que se propone en el Sistema para la construcción de la actividad basada en la recopilación de datos. Sin embargo, esta tecnología carece de un dispositivo concreto con un uso específico, sino que se adapta a las necesidades del objeto en función de la tarea que se quiera medir.

Con este proyecto se consigue el diseño de un dispositivo con un uso concreto para temas lúdico-terapéuticos, permitiendo así crear una ergonomía adecuada para la tarea a realizar, sin depender de diseños ya inventados y poco ergonómicos.

Capítulo 3

Entorno computacional

En este capítulo se explicarán aquellos entornos de programación que se han utilizado para el desarrollo del proyecto. El primero, Arduino, es la base del proyecto debido a que todos los sensores se conectan a un microcontrolador Arduino. El otro programa, Processing, se ha utilizado principalmente para realizar programas que permitan una visualización de los datos mediante una interfaz adecuada.

3.1. Arduino

Arduino [13] es una plataforma electrónica de código abierto con hardware y software de fácil uso. Las placas Arduino (figura 3.1) son capaces de, mediante sus pines de entrada y salida, conectarse a sensores y actuadores para recibir y enviar señales. Por ejemplo, permite saber si se pulsa un botón o si hay una luz encendida. De igual forma permite activar motores y luces LED, entre otros sistemas electrónicos. Esto se consigue mediante el lenguaje de programación de Arduino, basado en Wiring [14]. Gracias a este lenguaje se pueden mandar las órdenes al microcontrolador para que ejecute la tarea que se desea realizar. Para ello se utiliza el IDE de Arduino (figura 3.2).

Figura 3.1. Placas Arudino.

Arduino permite realizar una gran cantidad de proyectos, desde aplicaciones sencillas para el hogar como encender las luces del jardín automáticamente al oscurecer el día, hasta complejos instrumentos científicos. Gracias a esto hay accesible mucha información en Internet a través de plataformas de código abierto, que proveen de ayuda tanto a novatos como a expertos.

La plataforma nació en el Instituto de diseño e interacción de Ivrea en Italia como un proyecto dirigido a estudiantes sin formación previa en electrónica. Poco a poco se fue expandiendo y adaptando a los nuevos desafíos, pudiendo realizar aplicaciones de Internet de las Cosas (IoT), sistemas embebidos, impresoras 3D, etc. Al tratarse de un sistema

abierto, puede adaptarse y modificarse como el usuario quiera en función de sus necesidades, convirtiendo a Arduino en una plataforma muy versátil.

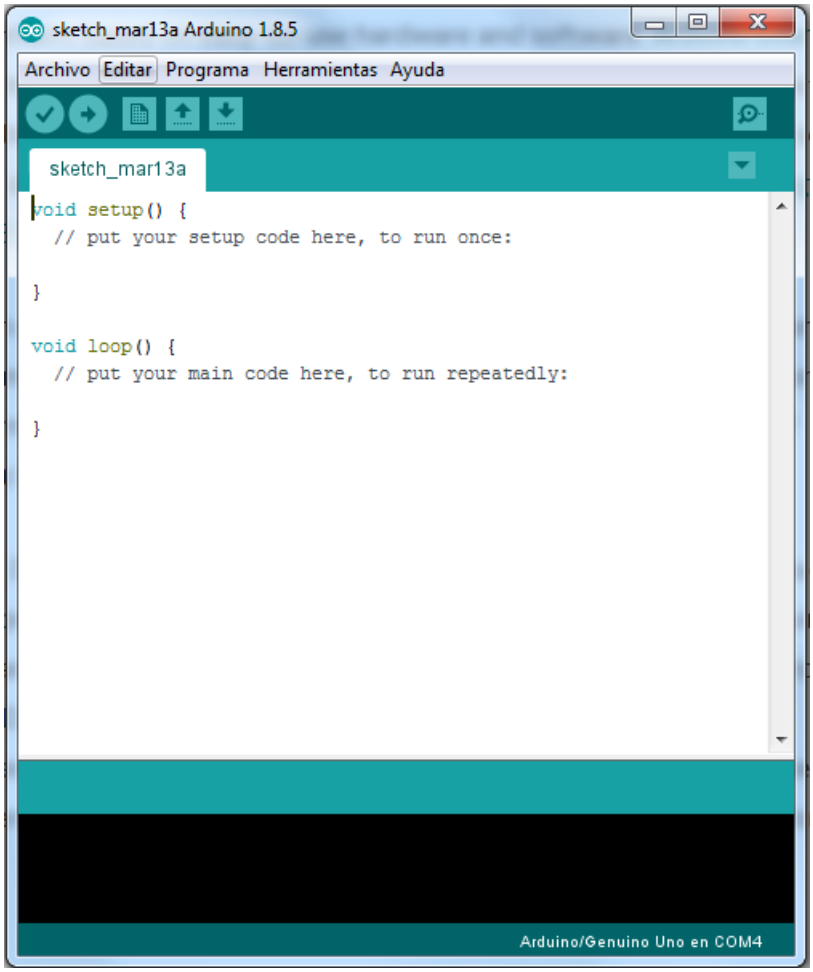


Figura 3.2. Arduino IDE.

Existen otras plataformas basados en microcontroladores en el Mercado, como Parallax Basic Stamp, Netmedia's BX-24, Phidgets, etc. Sin embargo, Arduino ofrece unas ventajas por sobre estos sistemas:

- **Ecnonómico**. Las placas Arduino son relativamente baratas comparadas con otros sistemas. El módulo más caro se puede encontrar por menos de 50€.
- **Multiplataforma**. El IDE puede utilizarse en cualquier sistema operativo, ya sea Windows, Macintosh o Linux.
- **Simple**. Es suficientemente simple como para ser usado por principiantes y a la vez lo suficientemente flexible para permitir a los usuarios más avanzados realizar tareas complejas.
- **Software libre**. Al tratarse de un sistema de código abierto, hay muchos usuarios expertos que extienden sus aplicaciones y transmiten su conocimiento a través de Internet.
- **Hardware libre**. El diseño de las placas está disponible de forma gratuita. Esto permite a diseñadores con experiencia a modificar los módulos y mejorarlos para sus propios proyectos.

3.2. Processing

Processing (figura 3.3) es un lenguaje de programación de código abierto creado por Ben Fry y Casey Reas en 2001 en el MIT Media Lab. Processing se centra en las artes visuales dentro de la tecnología, de forma que permite enseñar programación dentro de un contexto visual [15]. Actualmente ha evolucionado hasta el punto que es usado para aplicaciones profesionales.

Al igual que con Arduino, hay una gran comunidad que promueve el crecimiento de Processing compartiendo librerías y programas. El rango de aplicaciones varía desde visión por computador, visualización de datos, composición de música, programación de electrónica, creación de interfaces, etc.

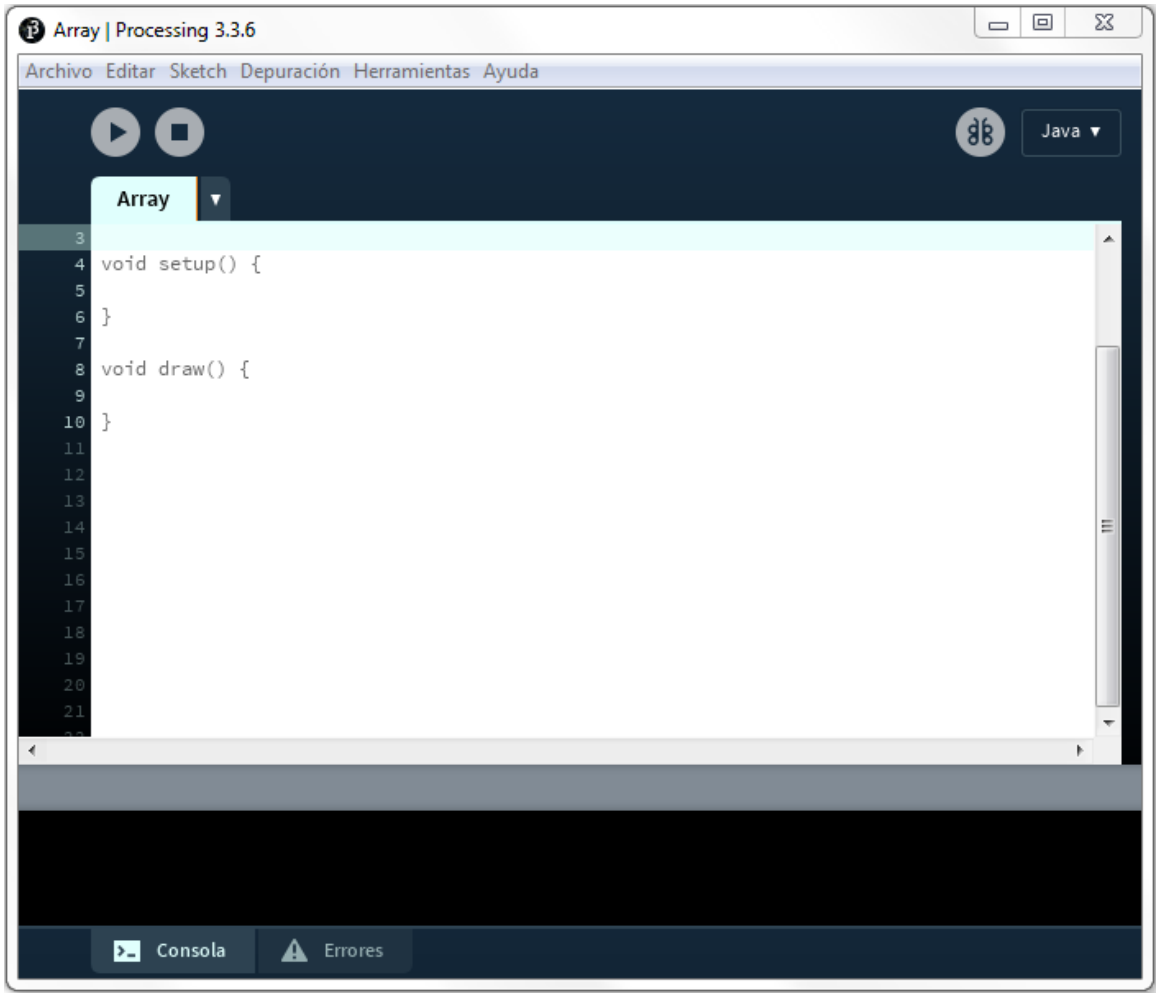


Figura 3.3. Processing IDE.

Este software es utilizado por una gran cantidad de artistas, diseñadores y arquitectos. Algunos de los proyectos se han mostrado en el Museo de Arte Moderno de Nueva York,

el Museo Victoria de Londres y el Centro Georges Pompidou en París. Se ha utilizado para generar imágenes para vídeos musicales y películas.

También es utilizado por grandes corporaciones como Google, Intel y General Electric para diseñar interfaces y servicios o visualizar datos internos. Por ejemplo, el New York Times Company R&D Lab lo utilizan para visualizar cómo varían sus artículos e historias a través de las redes sociales.

Capítulo 4

Diseño electrónico

En este capítulo se describen todos los componentes electrónicos que integran el dispositivo, esto es, el microcontrolador, los sensores utilizados, el sistema de comunicación inalámbrica y batería externa, su función, la circuitería de acondicionamiento de señal que necesitan, así como los instrumentos de validación que se han utilizado. Además, se describen también los programas para la visualización de datos.

4.1. Microcontrolador

Como se mencionaba en el capítulo 3, el controlador del dispositivo que se ha seleccionado es una placa Arduino. El testeo de todos los sensores se ha realizado con la placa Arduino UNO R3, pero debido a su tamaño, la placa más adecuada para integrarla en el mando es la placa Arduino Nano (figura 4.1). El sistema de pines de esta placa es igual al de la placa UNO R3, por lo que no se tienen que realizar modificaciones en el código para leer los datos de los sensores. No posee conector para alimentación externa, y funciona con un cable USB Mini-B. En la tabla 4.1 [16] se detallan los pines de la placa.

Tabla 4.1. Pines Arduino Nano

Nº Pin	Nombre	Tipo	Descripción
1-2, 5-16	D0-D13	I/O	Entrada/Salida digital pines 0-13
3, 28	RESET	Input	Reset (activo en LOW)
4, 29	GND	PWR	Tierra
17	3V3	Output	Salida a +3,3V
18	AREF	Input	Referencia ADC
19-26	A7-A0	Input	Entradas analógicas pine 0-7
27	+5V	Output/Input	Salida +5V (del regulador de la placa) Entrada +5V de fuente externa
30	VIN	PWR	Tensión de alimentación

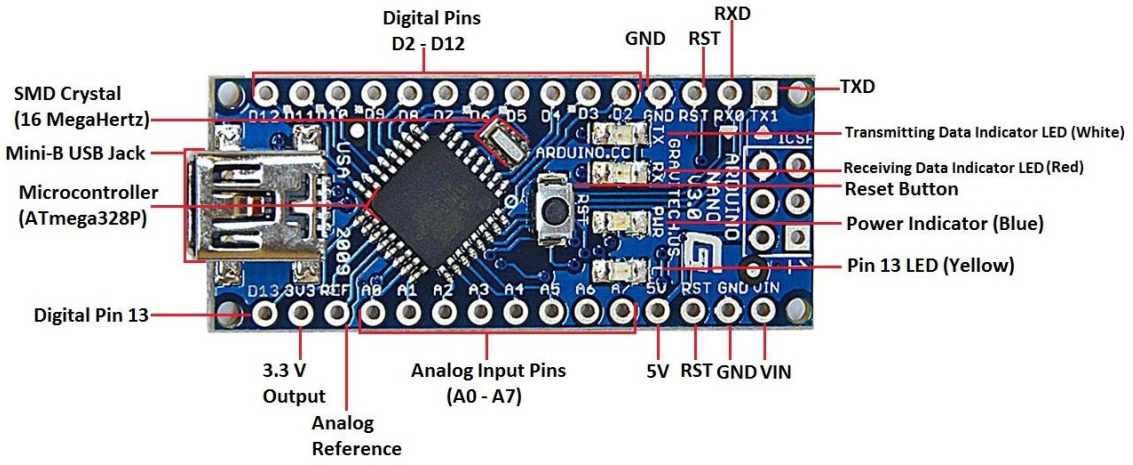


Figura 4.1. Pines Arduino Nano

Como se muestra en la tabla 4.1, el Arduino también funciona como regulador de tensión de los dispositivos electrónicos. Para ello se utilizan los pines de 5V y 3,3V y GND para la referencia. Se ha optado por alimentar todos los componentes utilizados a 5V para homogeneizar la tensión de alimentación de todos los sensores de forma que sea más sencilla la conexión de estos.

4.2. Unidad de Medición Inercial

La unidad de medición inercial (IMU) es un dispositivo formado por giroscopios y acelerómetros, y en algunos modelos también magnetómetros, capaz de medir la velocidad, orientación y fuerzas gravitacionales del dispositivo en el que está integrado. Se utilizan principalmente en aviones, vehículos aéreos no tripulados, satélites, etc.

Sin embargo, una de las grandes desventajas que presentan las IMU es el error acumulativo (*drift*). Debido a estos errores calcular la posición de un objeto únicamente con el uso de IMU es prácticamente imposible, y para ello se tendrían que utilizar otros sistemas para corregir su imprecisión, como por ejemplo GPS, sensores de gravedad, un sistema barométrico para corregir la altitud, etc.

Para el desarrollo del mando no se ha contemplado la posibilidad de incorporarlos debido al tamaño de algunos de estos sensores, por lo que el cálculo de la posición no ha sido posible de conseguirlo con precisión.

La orientación por otro lado sí que es posible de obtener con el uso de la IMU. Para ello se ha decidido utilizar el modelo MPU6050 (Figura 4.2) [17], que incorpora 3 acelerómetros y 3 giroscopios. Utilizando la combinación de todos ellos para obtener datos en tres ejes es posible obtener la orientación de un objeto, el mando en este caso. La programación del chip se ha realizado de forma que el resultado viene dado en cuaterniones [18], representados de la siguiente manera: $Q = w + ix + jy + kz$. Siendo w, x y, z las componentes reales del cuaternión e i, j y k las componentes imaginarias. Entre sus usos destacan las rotaciones en el espacio, motivo principal de su uso en este proyecto.

Para obtener una orientación correcta, el primer paso es calibrar el sensor. Esto proporciona los offset de aceleración y orientación en los ejes X, Y, Z. De esta manera se consigue eliminar las desviaciones del sensor por fabricación.

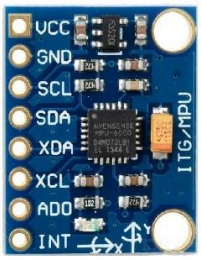


Figura 4.2. IMU modelo MPU6050

Las conexiones de los pines con la placa Arduino son las siguientes:

Tabla 4.2. Conexiones MPU6050

Arduino	MPU6050
A4	SDA
A5	SCL
5V	VCC
GND	GND
GND	AD0
Pin digital 2	INT

Dado que el puerto serie de Arduino no proporciona una buena visualización de los datos se ha utilizado un programa en Processing que permita ver de forma gráfica la orientación del sensor en tiempo real.

Debido a que la IMU no incorpora magnetómetro, cada vez que se enciende el dispositivo toma como referencia la posición en la que se encuentra. Por este motivo se tiene que realizar una segunda calibración para tomar la referencia como toca. Si no se realiza esto la imagen de la orientación en pantalla no irá sincronizada con la orientación

real del dispositivo. Para ello, el sensor deberá estar en horizontal con respecto a la pantalla en la que se visualicen los datos. En la figura 4.3 puede verse la pantalla de visualización que se ha programado para ver los resultados.

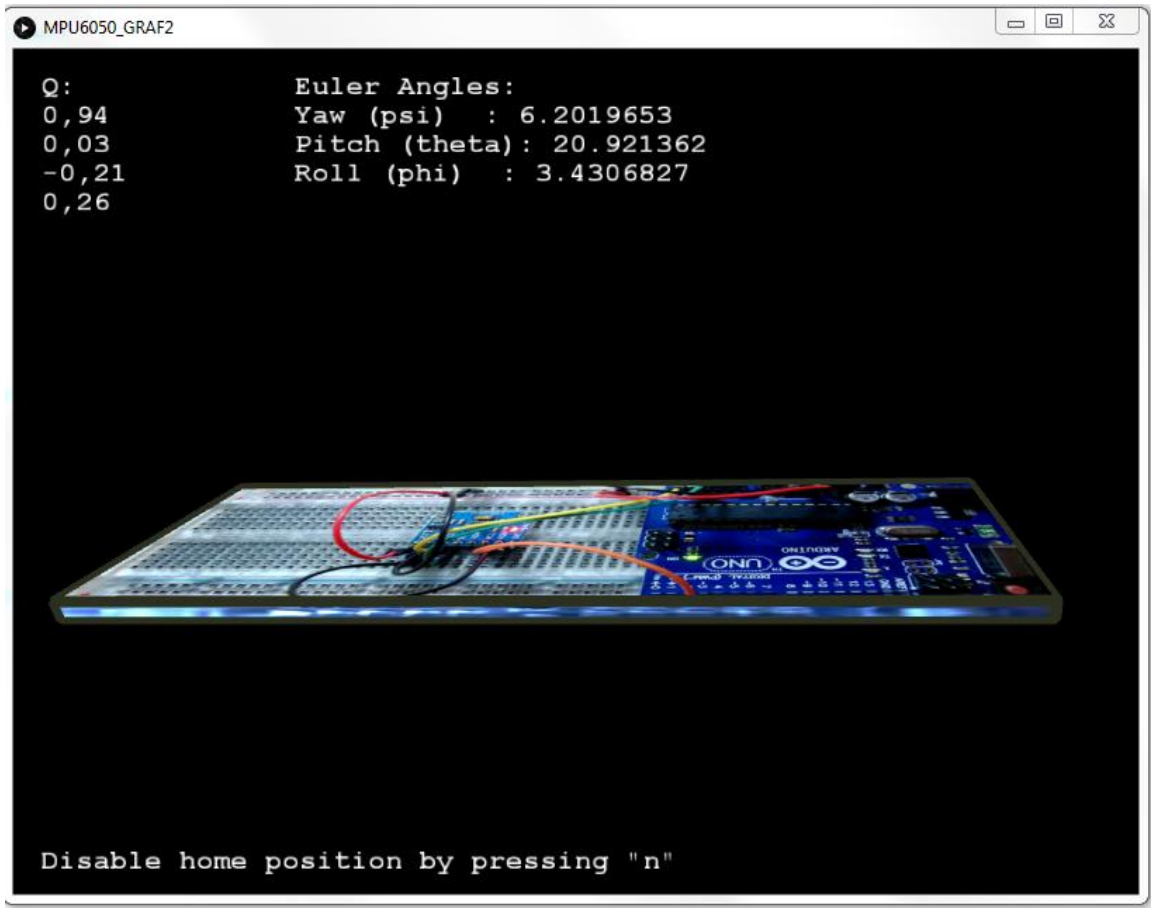


Figura 4.3. Orientación IMU en Processing.

4.3. Sensores de fuerza

Como ya se ha mencionado en el capítulo 1, una de las características del mando es que debe de ser capaz de medir la fuerza de agarre del usuario, así como medidas de fuerza más precisas con los dedos. Para ello se ha dispuesto de dos tipos de sensores diferentes

según el caso. Galgas extensiométricas (figura 4.4) para la fuerza de prensión (agarre) y sensores FSR (figura 4.8) para las medidas de precisión.

4.3.1. Galga extensiométrica

Este sensor [19] es una estructura que puede detectar cargas de compresión, tensión y flexión en función de la deformación que soporta. El modelo utilizado para el mando es el FX1901-0001-0100-L (Figura 4.4), que soporta una carga máxima de 100 libras (45.36 Kg). En un estudio realizado por MAPFRE [20] sobre los trabajadores de General Motors España se determinó que la fuerza de agarre máxima de una persona era de 60 kg. Por ese motivo se ha decidido incorporar al mando dos celdas de carga en lugar de una para poder disponer de una medida de fuerza máxima de 90.72 Kg, en la figura 4.5 se detalla la colocación de las galgas.

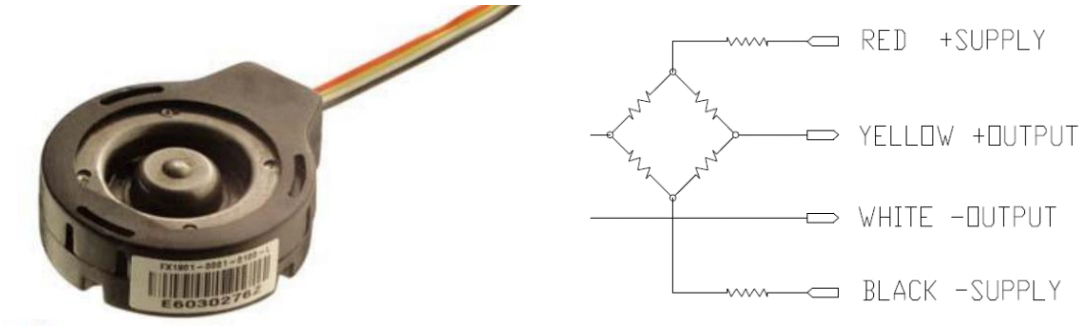


Figura 4.4. Galga extensiométrica FX1901-0001-0100-L.

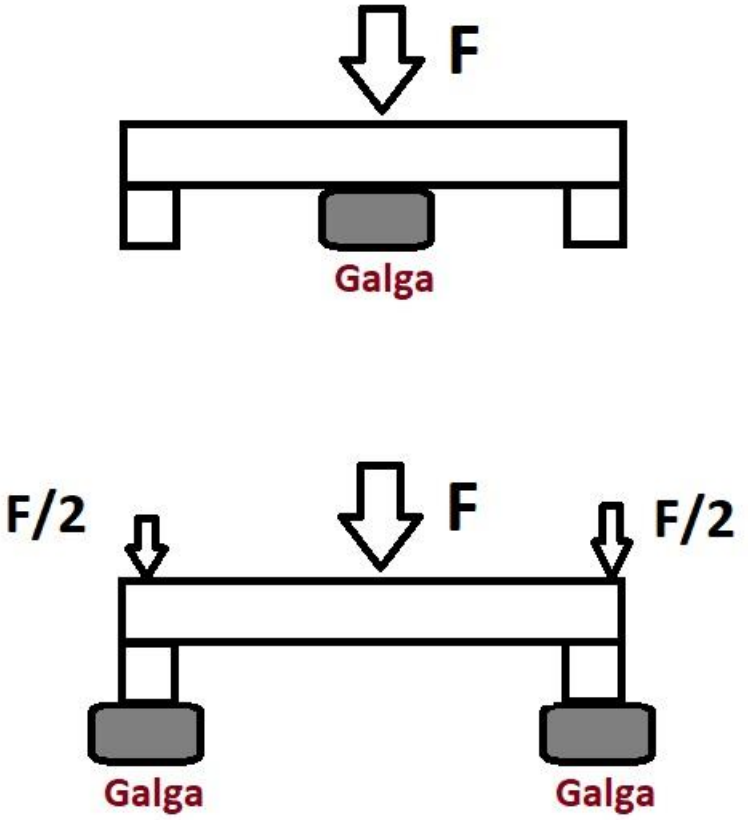


Figura 4.5. Distribución Galgas extensiométricas.

En la figura 4.5 se describe el proceso de distribución de fuerzas con uno y dos sensores. En la imagen superior se observa que toda la fuerza se aplicaría en un único dispositivo. Y como se comentaba anteriormente la carga máxima que soporta el sensor es de 45,36 Kg, por lo que no sería suficiente para detectar la fuerza de prensión real de un usuario capaz de ejercer más fuerza que esa. En cambio, en la segunda imagen se opta por una distribución de dos galgas separadas a una distancia simétrica del punto donde se aplicaría la fuerza, esto es, la fuerza en cada sensor sería $F_{galga} = \frac{F}{2}$, de esta forma la fuerza total sería $F = F_{galga1} + F_{galga2}$, pudiendo leer hasta 90,72 Kg.

Este modelo de galga tiene una sensibilidad de 20 mV/V, esto quiere decir que si se alimenta a 5V la tensión de fondo de escala es 0.1V. Esta medida no es adecuada para trabajar, por lo que será necesario amplificar la salida con un circuito acondicionador de señal para obtener una lectura más adecuada. Dicho valor se ha estipulado en 5V, por lo que se tiene que amplificar la señal por 50. Para conseguir eso se incorporará un circuito de acondicionamiento de señal con un amplificador. En concreto se utilizará el amplificador AD627 [21] (Rail to Rail) gracias a su versatilidad y que es un chip bastante utilizado para circuitos con puente de Wheastone.

4.3.2. Amplificdor AD627 (Rail to Rail)

El amplificador seleccionado para el circuito acondicionador es el AD627 (Figura 4.5), que permite una amplificación variable en función de una resistencia (R_G) que se le incorpore entre los pines 1 y 8.

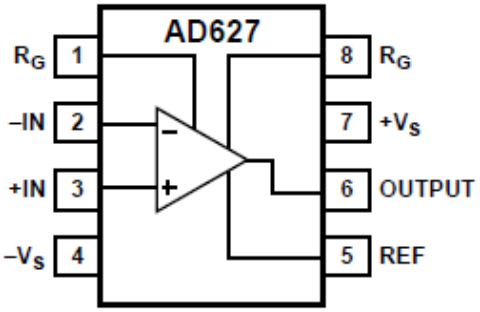


Figura 4.6. Encapsulado AD627

Pin No.	Mnemonic	Description
1	R_G	External Gain Setting Resistor. Place gain setting resistor across R_G pins to set the gain.
2	–IN	Negative Input.
3	+IN	Positive Input.
4	$-V_S$	Negative Voltage Supply Pin.
5	REF	Reference Pin. Drive with low impedance voltage source to level shift the output voltage.
6	OUTPUT	Output Voltage.
7	$+V_S$	Positive Supply Voltage.
8	R_G	External Gain Setting Resistor. Place gain setting resistor across R_G pins to set the gain.

Figura 4.6. Descripción pines AD627

Hay que tener en cuenta que no se van a tener voltajes negativos, por lo que la alimentación negativa $-V_S$ se tiene que conectar a GND, al igual que la referencia (REF).

Para saber qué valor de R_G se tiene que escoger se puede consultar la siguiente tabla.

Tabla 4.3. Valores recomendados de resistencia según ganancia

Desired Gain	1% Standard Table Value of R_G	Resulting Gain
5	∞	5.00
6	200 kΩ	6.00
7	100 kΩ	7.00
8	68.1 kΩ	7.94
9	51.1 kΩ	8.91
10	40.2 kΩ	9.98
15	20 kΩ	15.00
20	13.7 kΩ	19.60
25	10 kΩ	25.00
30	8.06 kΩ	29.81
40	5.76 kΩ	39.72
50	4.53 kΩ	49.15
60	3.65 kΩ	59.79
70	3.09 kΩ	69.72
80	2.67 kΩ	79.91
90	2.37 kΩ	89.39
100	2.1 kΩ	100.24
200	1.05 kΩ	195.48
500	412 Ω	490.44
1000	205 Ω	980.61

Dado que se pretende conseguir una ganancia de 50 el valor de resistencia más adecuado es de 4.53 KΩ.

El circuito final siguiendo la topología del puente de Wheastone se observa en la siguiente imagen.

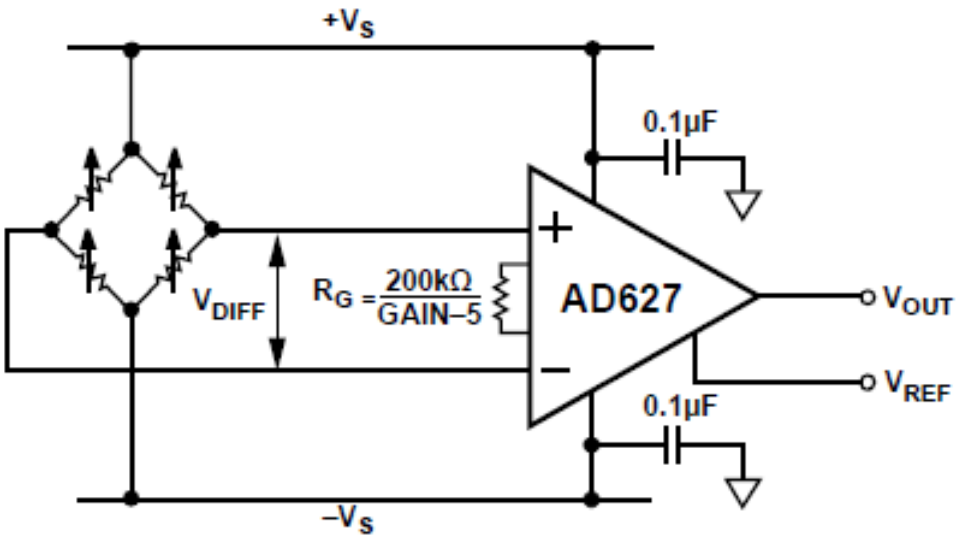


Figura 4.7. Circuito con galga extensiométrica con amplificador AD627.

Como se ha mencionado incorporando dos circuitos de este tipo en el mando se puede llegar a obtener una fuerza máxima de 90.72 Kg, cumpliendo sin problemas las especificaciones deseadas.

4.3.3. Conversión de datos a Kg

Finalmente, una vez que se tiene la señal de la fuerza entre 0 y 5 V, se tiene que realizar una conversión a Kg, que es la unidad habitual para medir la fuerza de la mano. Para ello se debe multiplicar la fuerza máxima que puede leer el sensor por la lectura de fuerza que da el mismo en mV. Esto proporciona un resultado en Kg·mV.

A continuación, se divide el valor anterior por la sensibilidad (mV/V) y la tensión de alimentación del sensor (5V). Con este cálculo se obtendría la fuerza en Kg.

Sin embargo, no hay que olvidar que la fuerza está amplificada por un factor 50, de forma que se tiene que dividir el resultado final por 50 para obtener la fuerza real.

La expresión sería la siguiente:

$$Fuerza_{galga}(Kg) = \frac{Lectura_{sensor}(mV) \cdot Fuerza_{máxima}(Kg)}{Sensibilidad\ \left(\frac{mV}{V}\right) \cdot Tensión_{alimentación}(V) \cdot Amplificación}$$

4.3.4. Sensor FSR

Los sensores de fuerza resistivos o FSR (figura 4.8) se constituyen de un polímero conductivo cuya estructura varía su resistencia en función de la presión aplicada. Cuanto mayor es la fuerza menor es la resistencia. Comparado con otros sensores de fuerza como las galgas extensiométricas tienen la ventaja de ser más pequeños y delgados, además de baratos y resistentes a los golpes. Por el contrario, son poco precisos, por lo que no son recomendables para medidas de alta precisión. El comportamiento que tiene es el siguiente.

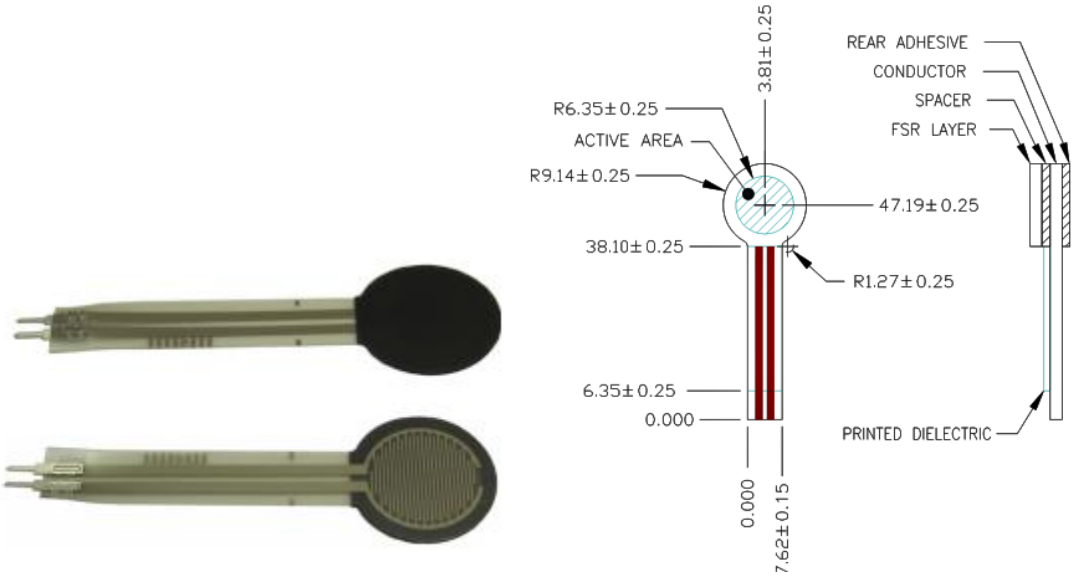


Figura. 4.8. Sensor FSR

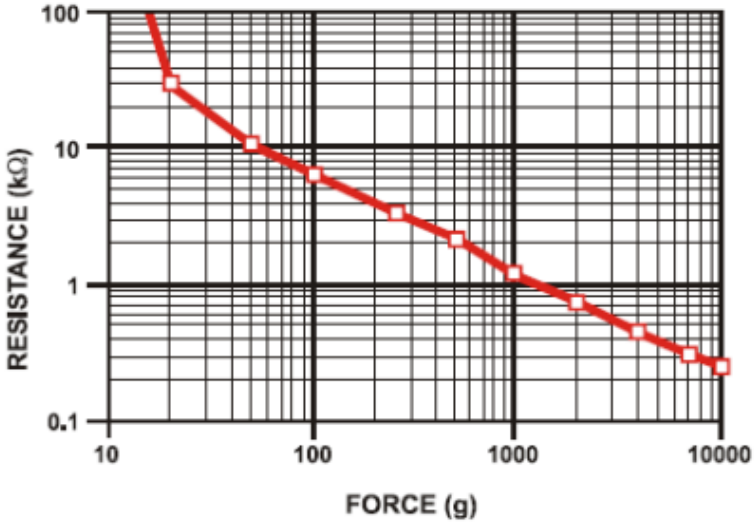


Figura 4.9. Fuerza vs resistencia sensor FSR

La forma más sencilla de medir la fuerza de este sensor es utilizar un esquema con una resistencia R de pull-down (figura 4.10). En este esquema se conecta el FSR a la tensión de alimentación con una resistencia de 10 KΩ en serie con la referencia o masa. *ANALOG* es la señal de lectura de la fuerza aplicada.

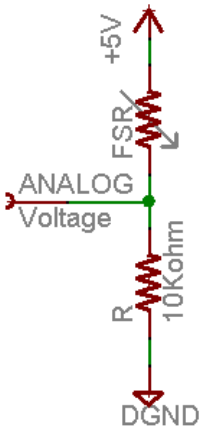


Figura 4.10. Conexión FSR

Se tiene que incorporar uno de estos sensores por cada dedo de la mano que se pretende evaluar, esto son 4 o 5 sensores en función de si se pretende controlar la acción del dedo pulgar o no.

La manera en la que trabaja es la siguiente. El conjunto de ambas resistencias varía entre 100 KΩ y 10 KΩ en función de la fuerza que se aplica. En la tabla 4.4 se puede ver una comparativa de la variación de la resistencia y del voltaje en la resistencia de pull-down para diferentes medidas de fuerza.

Tabla 4.4. Comparativa fuerza/resistencia/intensidad/voltaje en FSR

Fuerza (N)	Resistencia FSR (KΩ)	FSR+R (KΩ)	Intensidad (FSR+R) mA	Tensión (V)
0	Infinito	Infinito	0	0
0.2	30	40	0.13	1,3
1	6	16	0.31	3,1
10	1	11	0.45	4,5
100	250 Ω	10.25	0.49	4,9

La salida Vo (*ANALOG*) se puede medir realizando un divisor de tensión, esto es:

$$V_o = V_{cc}\left(\frac{R}{R + FSR}\right)$$

A partir de esta expresión se puede obtener el valor de la resistencia FSR:

$$FSR = \frac{(V_{cc} - V_o)R}{V_o}$$

Una vez conocido el valor de la resistencia y la gráfica de resistencia vs fuerza dada por el fabricante se puede obtener el valor de la fuerza aplicada.

Para visualizar los datos se ha realizado un programa en processing (figura 4.11) que crea una barra que se va rellenando a medida que se aplica más fuerza. Además de mostrar el valor de la fuerza en Kg.

Figura 4.11. Programa processing fuerza FSR.

Además de este programa también se ha hecho otro combinando este programa y el de la IMU para comprobar que ambos sensores funcionan correctamente. El programa se puede ver en la figura 4.12. La parte inferior permite visualizar la fuerza ejercida en los sensores, de igual forma que se ha comentado en la figura 4.11. La parte superior permite visualizar la orientación del dispositivo gracias a la IMU como se ha explicado en la figura 4.3.

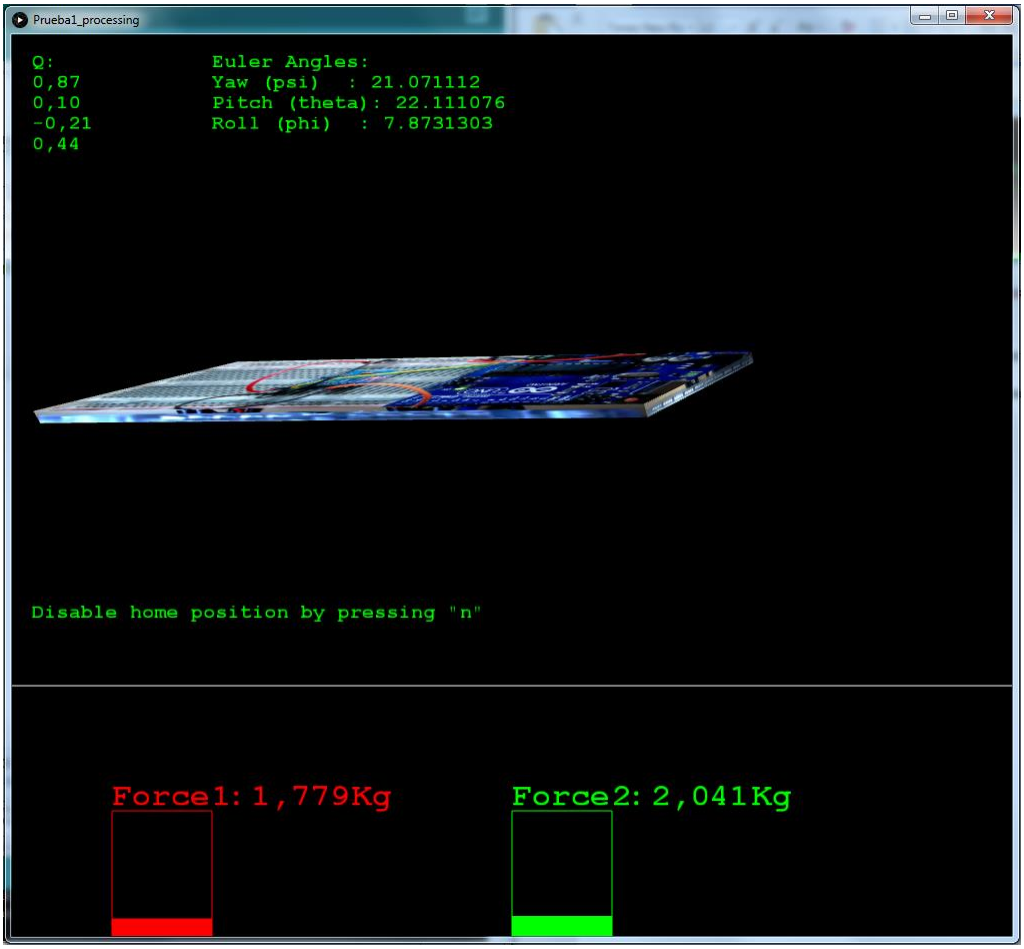


Figura 4.12. Programa processing IMU+FSR

4.3.5. Multiplexor CD74HC4051

Debido al alto número de sensores a incorporar en el mando se excede el número de entradas analógicas que dispone la placa Arduino, por ello se ha optado por introducir un multiplexor al que se conectarán todos los sensores de fuerza, esto es, galgas extensiométricas y sensores FSR.

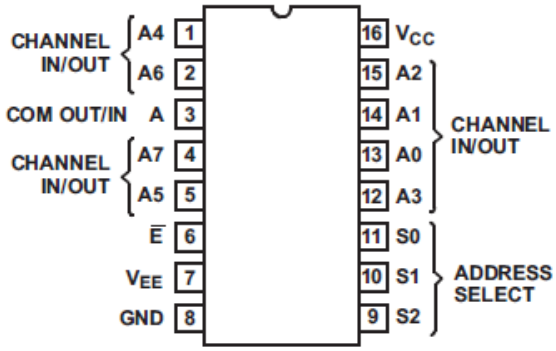


Figura 4.13. Multiplexor CD74HC4051

Donde el pin V_{EE} va conectado a GND y las señales de los sensores se conectan a los canales de entrada. En este caso se han puesto por orden primero los sensores FSR empezando por el pin A0 en adelante y continuando en orden hasta llegar a las celdas de carga. El pin A es la salida que se conecta a una de las entradas analógicas (A0) del arduino. Las conexiones se muestran en la siguiente tabla.

Tabla 4.5. Conexiones multiplexor y sensores

MUX	Sensores
A0	FSR 1
A1	FSR 2
A2	FSR 3
A3	FSR 4
A4	Galga extensiométrica 1
A5	Galga extensiométrica 2

La tabla de selección de canales es la siguiente del dispositivo se muestra a continuación (Tabla 4.6).

Tabla 4.6. Tabla de verdad MUX CD74HC4051

INPUT STATES				ON CHANNEL
ENABLE	S_2	S_1	S_0	
L	L	L	L	A0
L	L	L	H	A1
L	L	H	L	A2
L	L	H	H	A3
L	H	L	L	A4
L	H	L	H	A5
L	H	H	L	A6
L	H	H	H	A7
H	X	X	X	None

4.4. Sensor de pulso (ECG)

El ECG se trata de un sensor capaz de detectar la actividad cardiovascular de una persona. Utiliza una luz infrarroja para detectar pequeñas variaciones en la intensidad de la luz emitida. Estos cambios están relacionados con el volumen de sangre que pasa a través del tejido, de forma que filtrando la señal y amplificándola se puede conseguir una buena señal para calcular el pulso cardíaco. El sensor seleccionado es el *Easy Plugin Sensor* (figura 4.14) [22], un sensor de pulso cardíaco que mide las variaciones de luz emitida por la variación de sangre en el dedo. El filtrado y amplificación de la señal ya vienen implementados en el propio sensor, por lo que no hay que añadir circuitería adicional.

Figura 4.14. Easy Pulse Sensor.

Para el correcto funcionamiento del sensor hay que conectar los siguientes pines.

Tabla 4.7. Conexión sensor de pulso con Arduino

Arduino	Sensor de pulso
5V	5V
GND	GND
Pin analógico (A2)	A0

La salida que da el sensor permite construir directamente el electrocardiograma (ECG), pero si se quiere calcular el pulso cardíaco hay que realizar algunos cálculos. Básicamente, sabiendo cuándo se producen los picos en el ECG y calculando el tiempo que hay entre dichos picos (conocido como IBI) se obtiene cada cuánto tiempo se produce un latido. Al dividir 60 segundos entre el IBI se mide el número de pulsaciones por minuto de una persona.

Para verificar que el sensor funciona correctamente se han utilizado dos dispositivos comerciales para comparar ambos valores y en ambos casos los resultados fueron correctos. Los dispositivos que se utilizaron fueron el *reloj withing* y el *Garmin Vivosmart HR (figura 4.15).*

Figura 4.15. Rejoj withing (izquierda) y Garming Vivosmart HR (derecha)

Además de utilizar productos comerciales para comparar si la actividad del sensor es correcta, también se ha utilizado un programa en processing (figura 4.16) que permita visualizar la señal ECG, el pulso cardíaco y el IBI. En este caso la persona que realizó las pruebas es una persona activa y deportista, por este motivo el número de pulsaciones por minuto es bastante bajo (48 ppm). Sin embargo, se ha demostrado que el sensor funciona correctamente gracias a los dispositivos comerciales y a revisiones médicas que la persona que hizo las pruebas realizó con anterioridad.

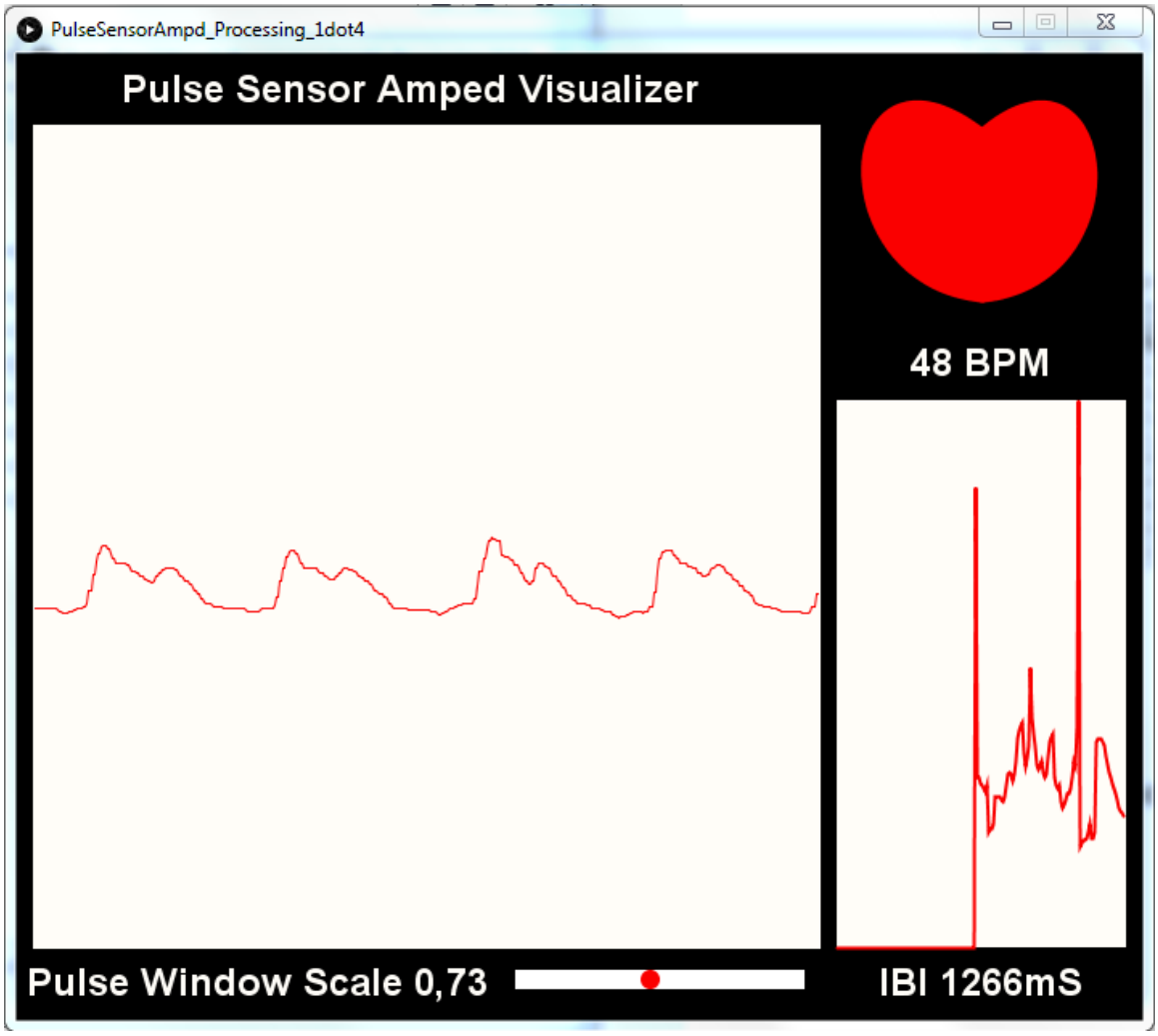


Figura 4.16. Programa processing ECG.

La idea inicial era utilizar el sensor en el dedo pulgar ya que no se pensaba utilizar para ningún sensor, el problema es que no se puede captar el pulso en el dedo pulgar correctamente ya que la arteria que pasa por el dedo es más grande que en el resto, por lo que no es posible para el sensor detectar correctamente los cambios en el volumen de la sangre que pasa a través de él. Esto significa que se tiene que utilizar uno de los dedos restantes para medir el pulso, lo que implica que se perdería uno de los sensores de fuerza

FSR al no ser compatible el uso de ambos con la tecnología seleccionada. Para evitarlo, una de las cosas que se debería estudiar es si se puede modificar el hardware del sensor para medir el pulso en otro lugar de la mano como pueda ser la palma o la muñeca, pero no se ha llegado a comprobar.

Algo que se debe tener en cuenta es que el sensor de pulso funciona correctamente por sí solo, pero al integrarlo con el resto de los sensores la señal se desestabiliza, impidiendo así obtener datos satisfactorios del pulso del usuario. Se han intentado diferentes pruebas para resolver el problema y conocer su causa, sin embargo, no se ha podido saber la causa.

4.5. Sensor electrodermal (EDA)

El sensor electrodermal es un sensor para medir la conductancia eléctrica de la piel. El sistema nervioso reacciona cuando se produce un estímulo, el resultado es la activación de las glándulas sudoríparas. Utilizando este tipo de sensores [23] es posible captar los cambios que se producen debido al sudor para así saber cuánto estrés supone la realización de una actividad o si, por el contrario, el usuario se encuentra tranquilo.

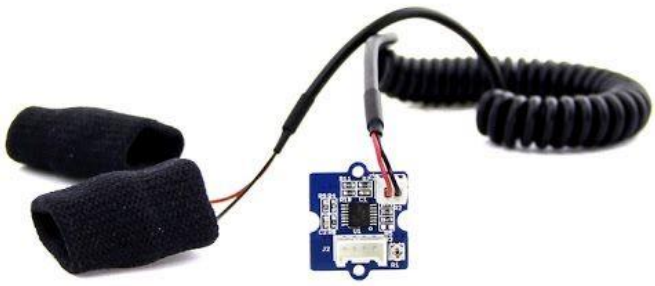


Figura 4.17. Sensor EDA

Las conexiones del sensor con la placa Arduino son las siguientes:

Tabla 4.8. Conexiones Arduino y sensor EDA

Arduino	Sensor EDA
GND	GND (Cable negro)
5V	Vcc (Cable rojo)
NC	Cable blanco
Pin analógico (A1)	Signal (Cable amarillo)

El dispositivo entrega una señal que varía entre 0 y 1024. Con esto no es posible saber directamente cuál es el valor de la conductividad de la piel. Por ello el datasheet del dispositivo ofrece la siguiente expresión para convertir esos valores en resistencia.

$$Resist\ piel = (1024 + 2 \cdot SerialPortReading) * 10000/(512 - SerialPortREading)$$

Donde las unidades son Ohmios (Ω).

Para saber cuál es la conductividad basta con hacer la inversa de la resistencia.

Para testear el sensor se escogieron videos aleatorios de internet para ver la reacción que tenía el usuario, comprobando que el sensor funciona correctamente.

Este sensor tiene la característica de que está diseñado para ser utilizado en los dedos de la mano. Dado que esto no es lo más adecuado para la integración del mando se propone modificar el hardware del sensor para que los metales conductores hagan contacto con la palma de la mano.

4.6. Sensor de electromiografía (EMG)

Como se explica en [24], la electromiografía es la detección y el registro de la actividad eléctrica de una parte del músculo. Esta actividad está relacionada con la estructura de la fibra muscular. Hay diferentes puntos a tratar al medir el EMG:

- Actividad en reposo
- Contracción voluntaria mínima
- Contracción voluntaria máxima

Para poder captar estos potenciales es necesario utilizar electrodos de aguja o de superficie colocados en el músculo.

La actividad en reposo se caracteriza porque después de un primer estado de inserción de los electrodos se produce silencio eléctrico, por lo que la línea isoeléctrica del osciloscopio será plana.

En los otros dos puntos (contracción voluntaria mínima y máxima) se puede apreciar cómo varía la señal eléctrica en función de la contracción que se realizan en los músculos medidos. Cuando la contracción es mínima se considera que el potencial tiene una duración

de entre 3 y 16 milisegundos con una amplitud de entre 300 microvoltios a 5 milivoltios. En el caso de una contracción máxima se acepta que el límite superior de activación de la unidad motora es de 50 por segundo.

Para el desarrollo del mando se ha investigado el uso de dos sensores, aunque no se han conseguido resultados satisfactorios con ninguno de ellos.

El primero de los sensores utilizado ha sido el *Muscle MyoWare* (Figura 4.18) [25]. Se trata de un sensor al que se le tienen que incorporar 3 electrodos para funcionar. Dos para captar la señal y otro para la referencia que permita obtener valores correctos. Los dos electrodos de señal se sitúan directamente en el músculo que se quiere medir, y la referencia se tiene que colocar en un área ajena al grupo muscular en cuestión, como por ejemplo el hueso del codo. En el caso del mando multimodal el grupo muscular que se pretende observar es el músculo interno del antebrazo. Hay que asegurarse de que los electrodos están colocados en el lugar correcto si no la señal leída no será correcta.

Otra de las cosas a tener en cuenta es que se trata de un sensor que se ve afectado en gran medida por el ruido externo. Por lo tanto, si el Arduino está conectado a un PC de sobremesa conectado a la corriente se verá afectada la señal en cierta forma. Para solucionarlo lo más adecuado es conectar el Arduino a una fuente de alimentación aislada de la red eléctrica, como puede ser una batería externa.

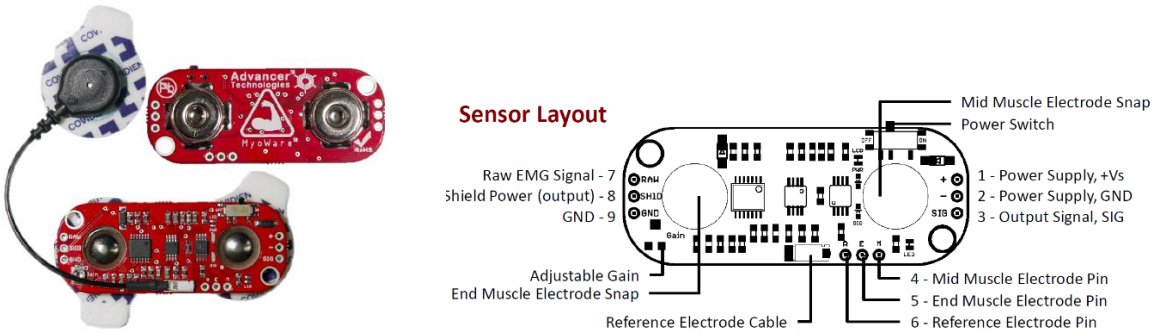


Figura 4.18. Sensor de electromiografía (EMG)

Como se aprecia en el layout de la figura 4.18 las conexiones con el Arduino son las siguientes.

Tabla 4.9. Conexiones Arduino y sensor de EMG

Arduino	Sensor EMG
5V	Pin 1 – +Vs
GND	Pin 2 – GND
A3	Pin 3 – Signal

Como se ha mencionado no se ha conseguido un resultado satisfactorio con este sensor. El motivo se debe a que el resultado esperado es encontrar un nivel de tensión en función de la contracción muscular. En este caso el resultado obtenido consistía en un cambio de potencial cada vez que había una contracción, pero la tensión no se mantenía para esa contracción, sino que volvía al estado de reposo.

Para encontrar soluciones se probó también el sensor de electromiografía de *Bitalino* (Figura 4.19) [26], un hardware específicamente diseñado para estudios médicos.

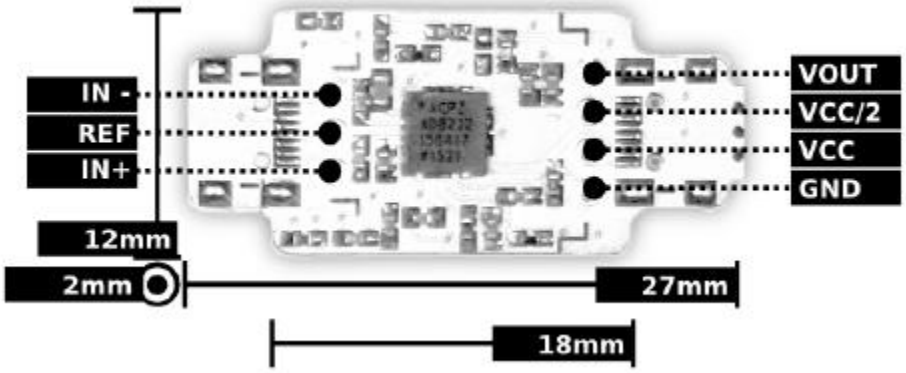


Figura 4.19. Sensor Bitalino

Al tratarse de un hardware específico es necesario utilizar un cable adaptador para Arduino que se puede obtener en la misma página web de Bitalino. También es necesario un cable para conectar los electrodos. De igual forma que con el sensor *MyoWare* se tienen 3 electrodos, dos para la señal y uno de referencia que deben ir conectados de igual forma.

El problema de este sensor es que la señal viene sin filtrar y sin integrar por lo que requiere de circuitería adicional que no se llegó a implementar, por lo que no se ha obtenido un resultado satisfactorio por el momento.

4.7. Módulo Bluetooth

Como se trata de un dispositivo inalámbrico, los datos capturados por los sensores tienen que enviarse a una aplicación que estará en un PC. Para ello se ha dispuesto de un módulo bluetooth HC-06 (Figura 4.20) [27]. Dicho módulo es sencillo de utilizar y servirá como si de un puerto serie se tratase. De esta forma al vincular el dispositivo con el PC y seleccionándolo en la aplicación que se pretenda utilizar ya se obtendrán los datos.

Figura 4.20. Módulo bluetooth HC-06

Para conectarlo correctamente hay que conectar los pines Rx y Tx del bluetooth con los pines Tx y Rx respectivamente del Arduino. Tal como se muestra en la tabla de conexiones.

Tabla 4.10. Conexiones Arduino y módulo bluetooth

Arduino	Bluetooth
5V	Vcc
GND	GND
Tx	Rx
Rx	Tx

Una de las cosas a tener en cuenta es que los pines Tx y Rx del arduino son los que ocupan el bus serie de la placa. Por este motivo al conectar el bluetooth a estos pines no es posible modificar el código del Arduino, y se tiene que desconectar el módulo para poder cargar un nuevo programa.

4.8. Batería

Como se ha mencionado a lo largo de esta memoria, este dispositivo es inalámbrico. Para que los sensores de un dispositivo inalámbrico tengan alimentación es necesario

incorporar una batería pequeña con potencia suficiente para otorgar al dispositivo de autonomía.

Por este motivo se ha seleccionado la una batería recargable de litio de la marca Ansmann de 2600 mAh y 3,7V (figura 4.21) [28]. Combinando este dispositivo con un regulador de tensión se puede elevar la tensión hasta los 5V de alimentación deseados.

Figura 4.21. Batería recargable Ansmann

Además del regulador sería necesario incluir circuitería para cargar la batería. Por ejemplo, se podría utilizar el circuito integrado MAX1811 (figura 4.22). Como se muestra en el datasheet del componente [27], es posible combinar este chip con un puesrto USB para cargar la batería y dar alimentación al sistema. Esto se muestra en la figura 4.23.

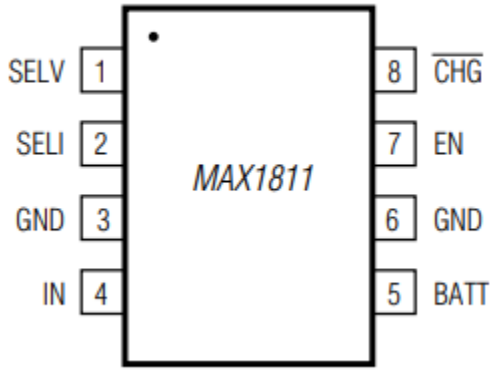


Figura 4.22. Cargador de batería MAX1811

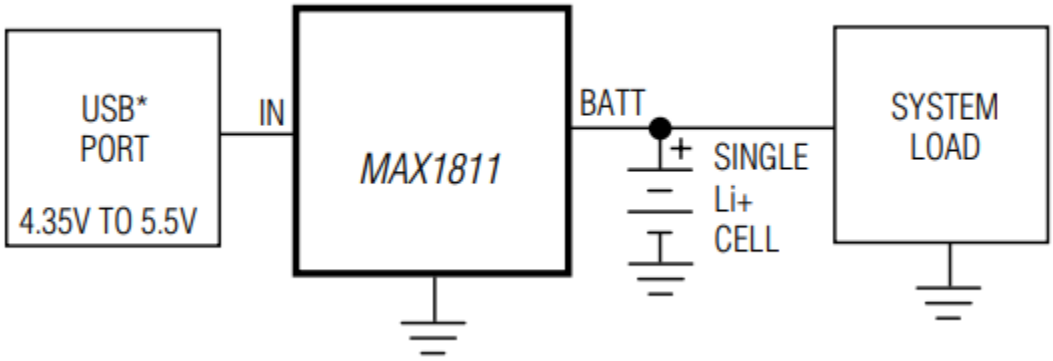


Figura 4.23. Configuración completa cargador de batería

Este sistema no se ha llegado a testear por lo que no es posible asegurar que vaya a funcionar correctamente. Sin embargo, se propone como una posible solución al problema de carga de la batería y se presupone como fiable al ser una configuración recomendada por el propio fabricante como se comenta en [29].

Capítulo 5

Presupuesto

En este capítulo se detallará el presupuesto de los componentes electrónicos para disponer un mando multimodal para el estudio de neuropatías. Debido a que con el sensor de EMG se ha hecho pruebas con dos sensores diferentes se detallarán dos presupuestos diferentes.

5.1. Presupuesto 1

En este apartado se detallará el presupuesto con el sensor EMG MyoWare Muscle Sensor.

Tabla 5.1. Presupuesto 1

Componente	Cantidad	Precio unitario (€)	Precio total (€)
FSR	4	3,88	15,52
Mux 74HC4051D	1	0,27	0,54
Galga FX1901	2	24,67	49,34
Amp AD627ARZ	2	5,58	11,16
MyoWare Muscle Sensor	1	37,95	37,95
Arduino Nano	1	3,1	3,1
IMU MPU6050	1	1,23	1,23
Batería	1	18,55	18,55
MAX 1811	1	3,86	3,86
Sensor EDA	1	9,9	9,9
Sensor ECG	1	14,51	14,51
Conector USB	1	0,376	0,376
Módulo Bluetooth	1	1,82	1,82
		Total (€)	**167,856**

5.2. Presupuesto 2

En la siguiente tabla se muestra el presupuesto si se utiliza el sensor EMG de Bitalino. El motivo por el que se separa en otra tabla es porque además se debe incluir el cable de adaptación de Bitalino a Arduino.

Tabla. 5.2. Presupuesto 2

Componente	Cantidad	Precio unitario (€)	Precio total (€)
FSR	4	3,88	15,52
Mux 74HC4051D	1	0,27	0,54
Galga FX1901	2	24,67	49,34
Amp AD627ARZ	2	5,58	11,16
EMG Bitalino	1	22,5	22,5
Conector Arduino-Bitalino	1	10	10
Arduino Nano	1	3,1	3,1
IMU MPU6050	1	1,23	1,23
Batería	1	18,55	18,55
MAX 1811	1	3,86	3,86
Sensor EDA	1	9,9	9,9
Sensor ECG	1	14,51	14,51
Conector USB	1	0,376	0,376
Módulo Bluetooth	1	1,82	1,82
		Total (€)	**162,406**

Capítulo 6

Conclusiones

Este documento presenta el diseño electrónico de un mando multimodal para el estudio de neuropatías. En él, se ha descrito el problema que se quiere analizar y cuáles son los puntos necesarios para resolverlos. De esta forma se ha concluido que se necesita un dispositivo que sea capaz de medir la fuerza de los dedos y de prensión y la orientación del dispositivo. Así mismo también tiene que ser capaz de medir parámetros biométricos como son el pulso cardíaco, la respuesta galvánica de la piel y la actividad de los músculos del antebrazo. Además, el dispositivo debe de ser inalámbrico para mayor comodidad, siendo capaz de enviar datos a distancia.

Se ha conseguido realizar un prototipo con todos los sensores montado en protoboard. Resolviendo los problemas técnicos que iban apareciendo como el uso de un multiplexor para aumentar el número de sensores conectados al microcontrolador, o la utilización de dos galgas extensiométricas para resolver el problema de la lectura máxima de la fuerza de prensión de la mano.

A pesar de eso, también han surgido diversos problemas que no se han resuelto. Como se ha mencionado en el capítulo 4.4, el sensor de pulso funciona correctamente por sí solo, pero al integrarlo con el resto de los sensores la señal se desestabiliza, impidiendo así obtener datos satisfactorios del pulso del usuario. Otro sensor que no ha otorgado los datos que se esperaban ha sido el sensor EMG (capítulo 4.6). El problema con este sensor ha sido que la respuesta dada únicamente mostraba datos cuando se detectaba un cambio

en la contracción del músculo, en lugar de detectar diferentes niveles de señal en función de lo contraído que estuviera el antebrazo. Por ese motivo se utilizó otro sensor, pero aparecía el problema de que la señal venía sin filtrar, lo que suponía que además se debía de añadir más circuitería para solventar el problema, algo que puede producir problemas en la integración del dispositivo final.

Con este proyecto se ha conseguido la primera fase para obtener un prototipo funcional de mando multimodal. A partir de este trabajo es posible realizar una estructura mecánica que contenga toda la electrónica mencionada, desarrollando previamente placas de circuito impreso para una perfecta integración con la envoltura.

Gracias al proyecto global se puede conseguir un avance significativo en la rehabilitación de personas con problemas motores, tanto si se trata de niños como de ancianos, ya que se ofrece la posibilidad de realizar un proceso de terapia física lúdica.

Lista de referencias

[1]: Discapacidad motriz. Guía didáctica para la inclusión en educación inicial y básica. Página 18.

[2]: https://www.fisiohogar.com/7-consejos-mejorar-tratamiento-tienes-una-neuropatia/

[3]: F. Molina-Rueda, S. Pérez de la Cruz. Esclerosis múltiple y técnicas de relajación. Revista Iberoamericana de FISIOTERAPIA y KINESIOLOGÍA.

[4]: Dania Ruíz García, Luis Alberto Solar Salaverri. Esclerosis múltiple. Revisión bibliográfica.

[5]: Rogelio Domínguez Moreno, Mario Morales Esponda, Natalia Lorena Rossiere Echazarreta, Romás Olan Triano, José Luis Gutiérrez Morales. Esclerosis múltiple: revisión de la literatura médica.

[6]: https://www.infosalus.com/enfermedades/neurologia/em-esclerosis-multiple/que-es-em-esclerosis-multiple-64.html

[7]: Mª Teresa Guerrero Díaz, Mª Cruz Macías Montero, Florentino Prado Esteban, Angélica Muñoz Pascual, Mª Victoria Hernández Jiménez, Jacinto Duarte García-Luis. Enfermedad de Parkinson.

[8]:https://www.infosalus.com/enfermedades/neurologia/parkinson/que-es-parkinson-25.html

[9]: Francisco José Perales López, Miguel Jesús Roca Adrover, Víctor Becerra Sanhueza. Patente número U201330260. Mando portátil para detección de movimiento y fuerza de prensión.

[10]: Johnny Chung Lee. Hacking the Nintendo Wii Remote. IEEE Intelligent Systems.

[11]: Kenichiro Ashida, Junji Takamoto, Masato Ibuki, Shinji Yamamoto, Hirokazu Matsui, Daisuke Kumazaki, Akiko Suga. United States Design Patent. Patente número US D559,254 S. Controller for electronic game machine.

[12]: Vincent Le Chevalier, William Vablais. Número de patente US 2017/0291066 A1. System and method for building activity-based data collection devices.

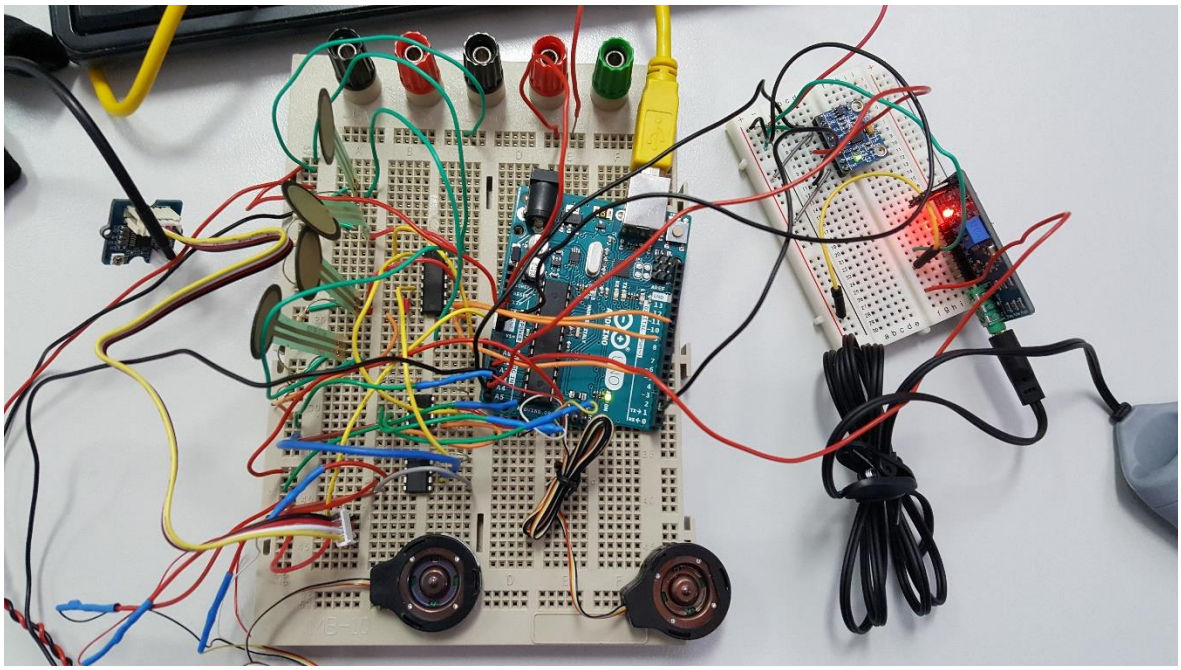
[13]: https://www.arduino.cc/en/Guide/Introduction#

[14]: http://wiring.org.co/
[15]: https://processing.org/
[16]: Arduino Nano. User Manual (https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf)
[17]: MPU6050 Datasheet. (https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf)
[18]: William Rowan Hamilton. On quaternions, or on a new system of imaginaries in albrebra.
[19]: Datasheet Galga extensiométrica. (https://docs-emea.rs-online.com/webdocs/142c/0900766b8142cdcb.pdf)
[20]: Víctor Alcalde Lapiedra, José Manuel Álvarez Zárate, Javier Bascuas Hernández, Ana García Felipe, Ana Germán Armijo, Emilio Rubio Calvo. La carga física de trabajo en extremidades superiores.
[21]: Amplificador AD627 Datasheet. (https://www.analog.com/media/en/technical-documentation/data-sheets/ad627.pdf)
[22]: Easy Pluse Sensor Datasheet. (http://embedded-lab.com/uploads/manuals/EasyPulse_User_Guide.pdf)
[23]: Grave GSR Sensor SeedStudio Datasheet.
[24]: R.D. DE CALDERON. La electromiografía en la evaluación de las enfermedades neuromusculares.
[25]: MyoWare Muscle Sensor Datasheet. (https://cdn.sparkfun.com/datasheets/Sensors/Biometric/MyowareUserManualAT-04-001.pdf)
[26]: Bitalino EMG Sensor Datasheet. (https://bitalino.com/datasheets/REVOLUTION_EMG_Sensor_Datasheet.pdf)
[27]: Bluetooth HC-06 Datasheet.
[28]: Batería de Litio recargable Ansmann Datasheet. (https://docs-emea.rs-online.com/webdocs/1249/0900766b81249861.pdf)

[29]: Cargador batería MAX1811 datasheet. (https://docs-emea.rs-online.com/webdocs/14cd/0900766b814cd104.pdf)

Anexo

Prototipo montado en protoboard



Código Mando multimodal

El código está hecho utilizando diferentes scripts. Cada uno de los scripts implementa la acción de un sensor o de un grupo de sensores con funciones similares (FSR + Galga). En el caso del sensor de pulso se utilizan tres scripts diferentes ya que se implementa con el uso de interrupciones.

Principal

```
//Programa principal que controla la IMU6050 y los FSR
//#include <SoftwareSerial.h>


// SoftwareSerial btMando(8,7); // Se especifican los pines del arduino que actuarán como Rx y Tx respeectivamente, recordar que se cruzan con los pines de Rx y Tx del bluetooth.

////  Variables HEART_RATE_MONITOR
int pulsePin = 2;                 // Pulse Sensor purple wire connected to analog pin 2
int blinkPin = 13;                // pin to blink led at each beat
//int fadePin = 5;                // pin to do fancy classy fading blink at each beat
//int fadeRate = 0;                 // used to fade LED on with PWM on fadePin

// Volatile Variables, used in the interrupt service routine!
volatile int BPM;                   // int that holds raw Analog in 0. updated every 2mS
volatile int Signal;                // holds the incoming raw data
volatile int IBI = 600;             // int that holds the time interval between beats! Must be seeded!
volatile boolean Pulse = false;     // "True" when User's live heartbeat is detected. "False" when not a "live beat".
volatile boolean QS = false;        // becomes true when Arduoino finds a beat.

// Regards Serial OutPut  -- Set This Up to your needs
static boolean serialVisual = false;   // Set to 'false' by Default.  Re-set to 'true' to see Arduino Serial Monitor ASCII Visual Pulse
//END variables HEART_RATE_MONITOR

void setup() {

  Serial.begin(57600);
  //while (!Serial); // wait for Leonardo enumeration, others continue immediately
```

```
  mpu6050_setup();
  FSR_and_galga_setup();
  Pulse_setup();

//  btMando.begin(57600);

}

void loop() {

  //Se ejectuan los programas de cada sensor
  mpu6050_process(); //Sensor IMU 6050
  delayMicroseconds(1);
  fsr_and_galga_process(); //Sensor FSR
  delayMicroseconds(1);

  //Se pasan los datos por puerto serie
  serial_MPU6050_data(); //Envio datos MPU 6050
  delayMicroseconds(1);
  serial_FSR_and_galga_data(); //Envio datos FSR
  delayMicroseconds(1);

  pulse_process(); //Sensor de pulso
  delayMicroseconds(1);
  emg_process(); //Sensor EMG
  delayMicroseconds(1);
  eda_process(); //Sensor EDA
  delayMicroseconds(1);
  Serial.println(); //Para distinguir entre el envío de nuevos datos
  delayMicroseconds(1);

  // if(Serial.available()){//Para mandar datos por el módulo bluetooth
    // float data = btMando.read();
    // btMando.println(data);
  // }
}
```

FSR + Galgas extensiométricas

```
const int num_galga_sensors=2;
```

```
float galgaValue[num_galga_sensors];//Valor leído en mV
float galgaValueN[num_galga_sensors];//Valor leído en Newtons
float galgaValueKg[num_galga_sensors];//Valor leído en Kg
//float messageGalga[num_galga_sensors];
float maxSensingWeight=45.3592; //El maximo son 100 libras, cuya conversion son
45.3592 Kg
float ampGain=50;//Ganancia del amplificador
float alimGalga=5;//Tensión de alimentación de la galga 5V
float sensGalga=0.02;//Sensibilidad de la galga 20mV/V
//float offsetGalga=3049;//Offset de 3049mV??Si se conecta la referencia del AD627 a
tierra no hace falta poner un offset

const int pinReading=A0; //Pin analógico del arduino, salida del MUX
const int S2=9;
const int S1=10;//**********************La interrupción del pulse sensor utiliza el
timer2 que hace que los pine 3 y 11 PWM dejen de funcionar, seguramente haya que
cambiarlo para que vaya bien
const int S0=12;
const int num_fsr_Sensors=4;
float fsrReading[num_fsr_Sensors];
float fsrVoltage[num_fsr_Sensors]; // the analog reading converted to voltage
float fsrResistance[num_fsr_Sensors]; // The voltage converted to resistance, can be very
big so make "long"
float fsrConductance[num_fsr_Sensors];
float fsrForce[num_fsr_Sensors]; // Finally, the resistance converted to force
//float messageFSR [num_fsr_Sensors];

int r0=0; //Variables para cambiar los pines S0, S1 y S2 del MUX de estado
int r1=0;
int r2=0;

void FSR_and_galga_setup() {
  //Serial.begin(57600); // We'll send debugging information via the Serial monitor

  pinMode(S2, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S0, OUTPUT);

  for(int i=0;i<num_fsr_Sensors; i++){
    fsrReading[i]=0;
    fsrVoltage[i]=0;
    fsrResistance[i]=0;
```

```
    fsrConductance[i]=0;
    fsrForce[i]=0;
//    messageFSR[i]=0;
  }

  for(int i=0; i<num_galga_sensors; i++){
    galgaValue[i]=0;
    galgaValueN[i]=0;
    galgaValueKg[i]=0;
  }
}


void fsr_and_galga_process() {

  for(int i=0; i<(num_fsr_Sensors + num_galga_sensors); i++){
    //delayMicroseconds(1);

    r0 = bitRead(i,0);
    r1 = bitRead(i,1);
    r2 = bitRead(i,2);

    digitalWrite(S0, r0);
    digitalWrite(S1, r1);
    digitalWrite(S2, r2);

    if(i<num_fsr_Sensors){
      fsrReading [i] = analogRead(pinReading);
      // analog voltage reading ranges from about 0 to 1023 which maps to 0V to 5V (=
5000mV)
      fsrVoltage[i] = map(fsrReading[i], 0, 1023, 0, 5000);
      //Serial.println(fsrVoltage[i]);

      // The voltage = Vcc * R / (R + FSR) where R = 10K and Vcc = 5V
      // so FSR = ((Vcc - V) * R) / V yay math!
      fsrResistance[i] = 5000 - fsrVoltage[i]; // fsrVoltage is in millivolts so 5V = 5000mV
      fsrResistance[i] *= 10000; // 10K resistor
      fsrResistance[i] /= fsrVoltage[i];

      //Serial.println(fsrResistance);

      fsrConductance[i] = 1000000; // we measure in micromhos so
```

```
      fsrConductance[i] /= fsrResistance[i];

      //Serial.println(fsrConductance);


      // Use the two FSR guide graphs to approximate the force
      if (fsrConductance[i] <= 1000) {
        fsrForce[i] = fsrConductance[i] / 80;
      } else {
        fsrForce[i] = fsrConductance[i] - 1000;
        fsrForce[i] /= 30;
      }

//    for(int j=0;j<num_fsr_Sensors; j++){
//      messageFSR[j]=fsrForce[j];
//    }
    }else if(i>=num_fsr_Sensors){//Entrará cuando se hayan acabado de procesar los
sensores fsr
      galgaValue[i-num_fsr_Sensors]=analogRead(pinReading);
      galgaValue[i-num_fsr_Sensors]=map(galgaValue[i-num_fsr_Sensors], 0, 1023, 0,
5000);
      //galgaValue=galgaValue-offsetGalga;

//    Serial.print("Valor Galga en mV: ");
//    Serial.print(galgaValue[i-num_fsr_Sensors]);
//    Serial.println("mV");

      galgaValueN[i-num_fsr_Sensors]=((galgaValue[i-
num_fsr_Sensors]/1000)*maxSensingWeight)/(ampGain*sensGalga*alimGalga);
//    Serial.print("Valor Galga en Newton: ");
//    Serial.print(galgaValueN[i-num_fsr_Sensors]);
//    Serial.println("N");
//    galgaValueKg[i-num_fsr_Sensors]=galgaValueN[i-num_fsr_Sensors]/10;

      galgaValueKg[i-num_fsr_Sensors]=((galgaValue[i-
num_fsr_Sensors]/1000.0)*maxSensingWeight)/(ampGain*sensGalga*alimGalga);
//    Serial.print("Valor Galga en Kg: ");
//    Serial.print(galgaValueKg[i-num_fsr_Sensors]);
//    Serial.println("Kg");
   }
  }
}
```

```
void serial_FSR_and_galga_data(){
  for(int i=0; i<num_fsr_Sensors; i++){
    Serial.print('#');
    Serial.print(fsrForce[i]);
  }

  for(int i=0; i<num_galga_sensors; i++){
    Serial.print('#');
    Serial.print(galgaValueKg[i]);
  }
  //Serial.println();
}
```

IMU MPU6050

```
/*
============================================================
===
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

J_RPM: modified to suit reading with PROCESSING 2
============================================================
======
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   #include "Wire.h"
#endif
```

```
// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/*
============================================================================
  MPU6050 - ARDUINO UNO
    SDA   -   A4
    SCL   -   A5
    AD0   -   GND (AD0 low = 0x68 / AD0 high = 0x69)
    INT   -   PIN2
    VCC   -   3.3V
    GND   -   GND
 *
============================================================================ */



// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
#define OUTPUT_READABLE_QUATERNION


//#define LED_PIN 13 // (LED Arduino en PIN 13)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

float messageQuaternion [4]={0, 0, 0, 0};
```

```
// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container

//VectorInt16 aa;         // [x, y, z]            accel sensor measurements
//VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
//VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
//VectorFloat gravity;    // [x, y, z]            gravity vector
//float euler[3];         // [psi, theta, phi]    Euler angle container
//float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
//uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };


//
================================================================
=
// ===               INTERRUPT DETECTION ROUTINE                ===
//
================================================================
=

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone
high
void dmpDataReady() {
    mpuInterrupt = true;
}

//
================================================================
=
// ===                    INITIAL SETUP                    ===
//
================================================================
=

void mpu6050_setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
```

```
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
//    Serial.begin(115200);
//    while (!Serial); // wait for Leonardo enumeration, others continue immediately


    // initialize device

    //Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // verify connection

    ///Serial.println(F("Testing device connections..."));
    ///Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") :
F("MPU6050 connection failed"));

// ELIMINATED, to start without pressing a key [J_RPM]
/*
    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again
*/

    // load and configure the DMP
    ///Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

//
==========================================================================
=
// Example, calibration data made with: MPU_6050_calibration
//
==========================================================================
=
/*
```

```
Sensor readings with offsets:  -6  -7  16379 1 -1  2
Your offsets: -89 1027  1453  104 -40 20
Data is printed as: acelX acelY acelZ giroX giroY giroZ
Check that your sensor readings are close to 0 0 16384 0 0 0
If calibration was succesful write down your offsets so you can set them in your projects
using something similar to mpu.setXAccelOffset(youroffset)
//
============================================================
=
// Commented on: Information from previous calibration [J_RPM]
//
============================================================
=

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip
*/

//
============================================================
=
// The current calibration data [J_RPM]
//
============================================================
=

    // supply your own gyro offsets here, scaled for min sensitivity
   mpu.setXGyroOffset(-1733);
   mpu.setYGyroOffset(26);
   mpu.setZGyroOffset(-72);
   mpu.setZAccelOffset(1246);// 16391 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready

        ////Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);
```

```
        // enable Arduino interrupt detection
        ////Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        ///Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    //pinMode(LED_PIN, OUTPUT);
}

//
============================================================
=
// ===                MAIN PROGRAM LOOP                ===
//
============================================================
=

void mpu6050_process() {

////////////////////////////////////////////////////////////////
// ===  Delay to adjust the speed of the PC with PROCESSING 2   === [J_RPM]
// ...if the processor in your PC is fast, you can eliminate this delay
////////////////////////////////////////////////////////////////
    delay(50);
    mpu.resetFIFO();
    delay(50);
```

```
////////////////////////////////////////////////////////////////////

    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
        // .
        // .
        // .
        // if you are really paranoid you can frequently test in between other
        // stuff to see if mpuInterrupt is true, and if so, "break;" from the
        // while() loop to immediately process the MPU data
        // .
        // .
        // .
    }


    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));

    // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
```

```
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        // New protocol for sending data [J_RPM]
        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);

            //Se rellenan las posiciones de memoria del array message con los datos de los
quaterniones
            messageQuaternion[0]=q.w;
            messageQuaternion[1]=q.x;
            messageQuaternion[2]=q.y;
            messageQuaternion[3]=q.z;

//              Serial.print(q.w);
//              Serial.print(",");
//              Serial.print(q.x);
//              Serial.print(",");
//              Serial.print(q.y);
//              Serial.print(",");
//              Serial.print(q.z);
//              Serial.println(",");
        #endif


        // blink LED to indicate activity
        blinkState = !blinkState;
        //digitalWrite(LED_PIN, blinkState);

    }
}

void serial_MPU6050_data(){
  for(int i=0; i<4; i++){
    Serial.print('#');
    Serial.print(messageQuaternion[i]);

  }
  //Serial.println();
}
```

EMG

```
// the loop routine runs over and over again forever:
void emg_process() {
  // read the input on analog pin 3:
  int sensorValue = analogRead(A3);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.print('#');
  Serial.print(voltage);
}
```

EDA

```
//const int GSR=A1; //Cambiar para poder conectar todos los sensores, se necesita el
MUX
float gsr_value=0;
float gsr_average=0;
float gsr_human_resistance=0;
float gsr_conductivity=0;

void eda_process(){
  float sum=0;
  for(int i=0;i<10;i++)          //Average the 10 measurements to remove the glitch
  {
    gsr_value=analogRead(A1);
    sum += gsr_value;
    //delay(5);
  }
  gsr_average = sum/10.0;
  gsr_human_resistance=(1024+2*gsr_average)*10000.0/(512-gsr_average);
  gsr_conductivity = 1/gsr_human_resistance;//En uS
  Serial.print('#');
  Serial.print(gsr_conductivity, 8); //Se multiplica por 1e6 para pasar a uS o se introduce
"," y un valor que indica el número de decimales, si no se pone nada se muestran dos
decimales.
}
```

Pulso

```
/*  Pulse Sensor Amped 1.4    by Joel Murphy and Yury Gitman
http://www.pulsesensor.com

----------------------  Notes ----------------------  ----------------------
This code:
1) Blinks an LED to User's Live Heartbeat   PIN 13
2) Fades an LED to User's Live HeartBeat
3) Determines BPM
4) Prints All of the Above to Serial

Read Me:
https://github.com/WorldFamousElectronics/PulseSensor_Amped_Arduino/blob/master/
README.md
 ----------------------       ----------------------  ----------------------
*/

////  Variables
//int pulsePin = 2;                 // Pulse Sensor purple wire connected to analog pin 0
//int blinkPin = 13;                // pin to blink led at each beat
//int fadePin = 5;                  // pin to do fancy classy fading blink at each beat
//int fadeRate = 0;                 // used to fade LED on with PWM on fadePin
//
//// Volatile Variables, used in the interrupt service routine!
//volatile int BPM;                   // int that holds raw Analog in 0. updated every 2mS
//volatile int Signal;                // holds the incoming raw data
//volatile int IBI = 600;             // int that holds the time interval between beats! Must be
seeded!
//volatile boolean Pulse = false;     // "True" when User's live heartbeat is detected.
"False" when not a "live beat".
//volatile boolean QS = false;        // becomes true when Arduoino finds a beat.
//
//// Regards Serial OutPut  -- Set This Up to your needs
//static boolean serialVisual = false;   // Set to 'false' by Default.  Re-set to 'true' to see
Arduino Serial Monitor ASCII Visual Pulse


void Pulse_setup(){
  pinMode(blinkPin,OUTPUT);         // pin that will blink to your heartbeat!
  //pinMode(fadePin,OUTPUT);          // pin that will fade to your heartbeat!
```

```
  //Serial.begin(115200);             // we agree to talk fast!
  interruptSetup();                 // sets up to read Pulse Sensor signal every 2mS
   // IF YOU ARE POWERING The Pulse Sensor AT VOLTAGE LESS THAN THE BOARD VOLTAGE,
   // UN-COMMENT THE NEXT LINE AND APPLY THAT VOLTAGE TO THE A-REF PIN
//   analogReference(EXTERNAL);
}


//  Where the Magic Happens
void pulse_process(){

    serialOutput() ;

  if (QS == true){     // A Heartbeat Was Found
                       // BPM and IBI have been Determined
                       // Quantified Self "QS" true when arduino finds a heartbeat
        //fadeRate = 255;         // Makes the LED Fade Effect Happen
                                  // Set 'fadeRate' Variable to 255 to fade LED with pulse
        serialOutputWhenBeatHappens();   // A Beat Happened, Output that to serial.
        QS = false;                      // reset the Quantified Self flag for next time
  }else{
    Serial.print('#'); //Para que se sigan mostrando los datos de pulso entre latido y latido
    Serial.print(BPM);
    Serial.print('#');
    Serial.print(IBI);
  }

  //ledFadeToBeat();                    // Makes the LED Fade Effect Happen
  //delay(20);                         //  take a break || Inicialmente activo, se ha desactivado para ver si es lo que producía error en la lectura del BMP
}




//void ledFadeToBeat(){
//    fadeRate -= 15;                         //  set LED fade value
//    fadeRate = constrain(fadeRate,0,255);   //  keep LED fade value from going into negative numbers!
```

```
//    analogWrite(fadePin,fadeRate);          //  fade LED
//  }
```

Interrupción Pulso

```
volatile int rate[10];                    // array to hold last ten IBI values
volatile unsigned long sampleCounter = 0;          // used to determine pulse timing
volatile unsigned long lastBeatTime = 0;           // used to find IBI
volatile int P =512;                      // used to find peak in pulse wave, seeded
volatile int T = 512;                     // used to find trough in pulse wave, seeded
volatile int thresh = 525;                // used to find instant moment of heart beat, seeded
volatile int amp = 100;                   // used to hold amplitude of pulse waveform, seeded
volatile boolean firstBeat = true;        // used to seed rate array so we startup with
reasonable BPM
volatile boolean secondBeat = false;      // used to seed rate array so we startup with
reasonable BPM


void interruptSetup(){
  // Initializes Timer2 to throw an interrupt every 2mS.
  TCCR2A = 0x02;     // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO
INTO CTC MODE
  TCCR2B = 0x06;     // DON'T FORCE COMPARE, 256 PRESCALER
  OCR2A = 0X7C;      // SET THE TOP OF THE COUNT TO 124 FOR 500Hz SAMPLE
RATE
  TIMSK2 = 0x02;     // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2
AND OCR2A
  sei();             // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED
}


// THIS IS THE TIMER 2 INTERRUPT SERVICE ROUTINE.
// Timer 2 makes sure that we take a reading every 2 miliseconds
ISR(TIMER2_COMPA_vect){                         // triggered when Timer2 counts to 124
  cli();                                  // disable interrupts while we do this
  Signal = analogRead(pulsePin);              // read the Pulse Sensor
  sampleCounter += 2;                     // keep track of the time in mS with this variable
```

```
  int N = sampleCounter - lastBeatTime;       // monitor the time since the last beat to
avoid noise

    //  find the peak and trough of the pulse wave
  if(Signal < thresh && N > (IBI/5)*3){       // avoid dichrotic noise by waiting 3/5 of last
IBI
    if (Signal < T){                        // T is the trough
      T = Signal;                         // keep track of lowest point in pulse wave
    }
  }

  if(Signal > thresh && Signal > P){          // thresh condition helps avoid noise
    P = Signal;                             // P is the peak
  }                                        // keep track of highest point in pulse wave

  //  NOW IT'S TIME TO LOOK FOR THE HEART BEAT
  // signal surges up in value every time there is a pulse
  if (N > 250){                                   // avoid high frequency noise
    if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) ){
      Pulse = true;                               // set the Pulse flag when we think there is a pulse
      digitalWrite(blinkPin,HIGH);                // turn on pin 13 LED
      IBI = sampleCounter - lastBeatTime;         // measure time between beats in mS
      lastBeatTime = sampleCounter;               // keep track of time for next pulse

      if(secondBeat){                        // if this is the second beat, if secondBeat == TRUE
        secondBeat = false;                  // clear secondBeat flag
        for(int i=0; i<=9; i++){             // seed the running total to get a realisitic BPM at
startup
          rate[i] = IBI;
        }
      }

      if(firstBeat){                         // if it's the first time we found a beat, if firstBeat ==
TRUE
        firstBeat = false;                   // clear firstBeat flag
        secondBeat = true;                   // set the second beat flag
        sei();                               // enable interrupts again
        return;                              // IBI value is unreliable so discard it
      }


      // keep a running total of the last 10 IBI values
```

```
    word runningTotal = 0;                // clear the runningTotal variable

    for(int i=0; i<=8; i++){              // shift data in the rate array
      rate[i] = rate[i+1];                // and drop the oldest IBI value
      runningTotal += rate[i];            // add up the 9 oldest IBI values
    }

    rate[9] = IBI;                        // add the latest IBI to the rate array
    runningTotal += rate[9];              // add the latest IBI to runningTotal
    runningTotal /= 10;                   // average the last 10 IBI values
    BPM = 60000/runningTotal;             // how many beats can fit into a minute? that's
BPM!
    QS = true;                            // set Quantified Self flag
    // QS FLAG IS NOT CLEARED INSIDE THIS ISR
   }
  }

  if (Signal < thresh && Pulse == true){   // when the values are going down, the beat is
over
   digitalWrite(blinkPin,LOW);            // turn off pin 13 LED
   Pulse = false;                         // reset the Pulse flag so we can do it again
   amp = P - T;                           // get amplitude of the pulse wave
   thresh = amp/2 + T;                    // set thresh at 50% of the amplitude
   P = thresh;                            // reset these for next time
   T = thresh;
  }

  if (N > 2500){                          // if 2.5 seconds go by without a beat
   thresh = 512;                          // set thresh default
   P = 512;                               // set P default
   T = 512;                               // set T default
   lastBeatTime = sampleCounter;          // bring the lastBeatTime up to date
   firstBeat = true;                      // set these to avoid noise
   secondBeat = false;                    // when we get the heartbeat back
   BPM = 0;
   IBI = 600;
  }

  sei();                                  // enable interrupts when youre done!
}// end isr
```

Envío de datos pulso

```
//////////
/////////  All Serial Handling Code,
/////////  It's Changeable with the 'serialVisual' variable
/////////  Set it to 'true' or 'false' when it's declared at start of code.
/////////

void serialOutput(){   // Decide How To Output Serial.
 if (serialVisual == true){
    arduinoSerialMonitorVisual('-', Signal);   // goes to function that makes Serial Monitor
Visualizer
 } else{
     //sendDataToSerial('S', Signal);     // goes to sendDataToSerial function
     Serial.print('#');
     Serial.print(Signal);
 }
}


//  Decides How To OutPut BPM and IBI Data
void serialOutputWhenBeatHappens(){
 if (serialVisual == true){            //  Code to Make the Serial Monitor Visualizer Work
   Serial.print("*** Heart-Beat Happened *** ");  //ASCII Art Madness
   Serial.print("BPM: ");
   Serial.print(BPM);
   Serial.print("  ");
 } else{
       Serial.print('#'); //Para que se sigan mostrando los datos de pulso entre latido y latido
       Serial.print(BPM);
       Serial.print('#');
       Serial.print(IBI);

 }
}



//  Sends Data to Pulse Sensor Processing App, Native Mac App, or Third-party Serial
Readers.
```

```
void sendDataToSerial(char symbol, int data ){
   Serial.print(symbol);

   Serial.println(data);
  }


//  Code to Make the Serial Monitor Visualizer Work
void arduinoSerialMonitorVisual(char symbol, int data ){
  const int sensorMin = 0;      // sensor minimum, discovered through experiment
const int sensorMax = 1024;    // sensor maximum, discovered through experiment

  int sensorReading = data;
  // map the sensor range to a range of 12 options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 11);

  // do something different depending on the
  // range value:
  switch (range) {
  case 0:
    Serial.println("");     /////ASCII Art Madness
    break;
  case 1:
    Serial.println("---");
    break;
  case 2:
    Serial.println("------");
    break;
  case 3:
    Serial.println("---------");
    break;
  case 4:
    Serial.println("------------");
    break;
  case 5:
    Serial.println("--------------|-");
    break;
  case 6:
    Serial.println("--------------|---");
    break;
  case 7:
    Serial.println("--------------|-------");
```

```
    break;
  case 8:
    Serial.println("--------------|----------");
    break;
  case 9:
    Serial.println("--------------|----------------");
    break;
  case 10:
    Serial.println("--------------|------------------");
    break;
  case 11:
    Serial.println("--------------|----------------------");
    break;

  }
}
```