Máster en Ingeniería de Sistemas y de Control

Universidad
Complutense
de Madrid

Universidad
Nacional
de Educación
a Distancia

Trabajo de Fin de Máster

Sistema de captura de movimientos basado en sistemas electrónicos de bajo coste

Curso 2017/2018 - Convocatoria de defensa de septiembre

Autor: Rubén Imbers Cid

Directores: José Sánchez Moreno y David Moreno Salinas

Máster en Ingeniería de Sistemas y de Control

Trabajo de Fin de Máster

Sistema de captura de movimientos basado en sistemas electrónicos de bajo coste

Autor: Rubén Imbers Cid

Directores: José Sánchez Moreno y David Moreno Salinas

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Firma del alumno

Resumen / Abstract

Las técnicas de captura de movimiento buscan cuantificar y digitalizar valores cinemáticos (posición, velocidad, aceleración y orientación) para su posterior análisis o reproducción. Este proyecto implementa una solución de captura de movimiento mediante tecnologías inerciales de bajo coste. Para ello se emplean acelerómetros MEMS y plataformas *open-source; Arduino*, para el *hardware* y la adquisición de datos, y *Processing*, para el procesado y reproducción 3D de los datos capturados en un ordenador. Se busca obtener los ángulos de Roll y Pitch correspondientes a la orientación de los sensores en ausencia de aceleraciones lineales. Se establece un canal de comunicación serie entre el Arduino y el ordenador ejecutando Processing por USB para el envío de los datos en tiempo real.

Movement capture techniques try to quantify and digitalize kinematic values (position, speed, acceleration and orientation) for further analysis or reproduction. This project implements a motion capture solution using low-cost inertial sensors. The project makes use of accelerometers and open source platforms; *Arduino* for the hardware and data acquisition and *Processing* for the data processing and 3D rendering in a computer. The goal is to find the Roll and Pitch angles that represent the orientation of the sensor in absence of linear accelerations. A serial link is stablished between the Arduino and the computer executing Processing via USB for real-time data transfer.

Palabras Clave / Keywords

Acelerómetro, captura de movimiento 3D, orientación 3D, Roll, Pitch, Arduino, Processing 3, código abierto.

Accelerometer, 3D motion capture, 3D orientation, Roll, Pitch, Arduino, Processing 3, open source.

Índice de contenido

Índice de figuras

1. Introducción

El análisis del movimiento humano es un campo que se lleva estudiando miles de años. Aristóteles está considerada la primera persona en realizar estudios biomecánicos por su libro "*De Motu Animalium*" (El movimiento animal), donde analizaba las diferencias entre los movimientos de los animales. Estos análisis basados en la observación eran difícilmente cuantificables con precisión. El estudio del movimiento general fue adquiriendo peso con el desarrollo de la física, con la creación de las disciplinas de dinámica y cinemática. Diferentes científicos han ampliado los estudios del movimiento desde entonces, desde Leonardo Da Vinci hasta Newton. (1)

El exuberante desarrollo tecnológico de las últimas décadas ha permitido desarrollar nuevas técnicas de captura de movimiento más precisas. Esto ha aumentado enormemente las herramientas disponibles para cuantificar movimientos y analizarlos, permitiendo asegurar la replicabilidad de los experimentos.

Debido al alto coste de investigación y desarrollo inicial requerido para los sistemas de captura de movimiento estos no han sido generalmente aplicables a productos de propósito general. La principal aplicación de los sistemas de captura de movimiento ha sido el de soporte a sistemas de navegación (embarcaciones y aeronaves), donde no existe una forma sencilla de cuantificar el movimiento.

Por suerte, gracias a la comunidad académica y a los proyectos *open-source*, donde entusiastas y expertos se unen para crear soluciones abiertas, se ha avanzado mucho en el desarrollo a las alternativas cerradas y propietarias para la captura de movimiento.

Esto, unido al abaratamiento tecnológico, ha ayudado a generar nuevos sectores de aplicación de las tecnologías de captura de movimiento, como la navegación autónoma, el estudio postural médico-deportivo, la animación y la realidad virtual. No es impactante que un teléfono móvil sepa interpretar su posición, o que un dron sea capaz de estabilizarse sólo en el aire, ya que son tecnologías visiblemente incorporadas en la sociedad moderna.

1.1. Objetivos

El objetivo de este proyecto es construir un sistema de captura de movimientos usando electrónica de bajo coste, en concreto placas tipo Arduino y acelerómetros repartidos por diversas partes del cuerpo para capturar los movimientos y posteriormente reproducirlos en un computador.

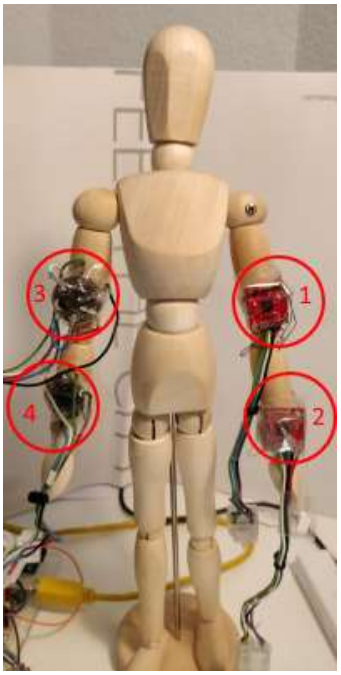


Ilustración 1. Disposición de los sensores.

Se busca desarrollar un sistema que obtenga la orientación de los brazos y antebrazos del usuario y los represente en un entorno 3D. Se hace uso de Processing 3 para realizar la animación 3D en un ordenador. La base desarrollada en este trabajo es ampliable a más sensores y diferentes partes del cuerpo, se ha escogido la representación de los brazos por sentar una base de trabajo.

Al disponer sólo de acelerómetros, se representan únicamente los ángulos de Roll (giro sobre el eje X) y Pitch (giro sobre el eje Y), ya que, como se explica en el capítulo 2, no es posible obtener el giro de Yaw (giro sobre el eje Z) sólo con los datos de las aceleraciones.

El sistema debe tener una velocidad de refresco de al menos 20 Hz para garantizar una reproducción fluida al ojo del movimiento.

El proyecto está dividido en 8 capítulos: El capítulo uno realiza una introducción del trabajo. El capítulo 2 realiza una bibliografía del estado del arte de la captura de movimiento. El capítulo 3 hace una explicación de las tecnologías y dispositivos empleados en este proyecto. El capítulo 4 describe el trabajo desarrollado. El capítulo 5 realiza un análisis del trabajo desarrollado. El capítulo 6 recoge las conclusiones que se extraen del trabajo realizado. El capítulo 7 agrupa la bibliografía utilizada en el desarrollo del proyecto. El capítulo 8 recoge las siglas y acrónimos empleados.

2. Estado del arte

Los sistemas de captura de movimiento deben, como su nombre indica, capturar el movimiento, pero también dar la posibilidad de reproducirlo o analizarlo posteriormente.

2.1. Captura de movimiento

En la actualidad existen dos aproximaciones ampliamente utilizadas para realizar capturas cuantificables de movimiento: los métodos ópticos y los métodos físicos.

2.1.1. Métodos ópticos de captura de movimiento

Los métodos ópticos se basan en una captura de imágenes o video y un procesado de estas mediante técnicas de visión por computador. Para ello es habitual colocar marcadores pasivos (para reflejar) o activos (para generar) luz. A la hora de realizar el procesado de las imágenes se emplean estos puntos de interés para hacer una reconstrucción de las extremidades.

Ilustración 2. Captura de movimiento óptica con marcadores pasivos. Fuente sp.edu.spg.

Los métodos ópticos suelen requerir un espacio cerrado con unas condiciones de luz determinadas. La grabación se realiza con múltiples cámaras sincronizadas entre sí, colocadas alrededor del actor para obtener datos del movimiento desde diferentes puntos de vista. Más tarde se fusionan los datos procesados de la grabación para recrear en un entorno informático el movimiento. El desarrollo de las técnicas de visión por computador está abriendo la posibilidad de prescindir de los marcadores ópticos. (2)

2.1.2. Métodos físicos de captura de movimiento y orientación

Los métodos físicos emplean propiedades mecánicas, cinemáticas y/o electromagnéticas para determinar el movimiento y/o la orientación. Un ejemplo de un método físico empleado durante años es la brújula, que emplea el campo magnético terrestre para orientar su cabezal al norte magnético. Dentro de los métodos físicos para capturar el movimiento se van a analizar los sistemas inerciales, por ser los que se van a emplear en este proyecto.

2.1.2.1. Métodos inerciales

Los sistemas inerciales emplean el movimiento de piezas mecánicas con una masa conocida para determinar su movimiento y/u orientación. La aparición de la tecnología MEMS (*Micro-Electro-Mechanical Systems*) ha contribuido a que estos sensores se desarrollen y reduzcan su precio enormemente, por lo que las aplicaciones de los métodos inerciales de captura de movimiento se han visto ampliadas. Este capítulo se centra en los métodos inerciales de captura de movimiento que emplean dispositivos MEMS. (3)

2.1.2.1.1. Acelerómetros MEMS

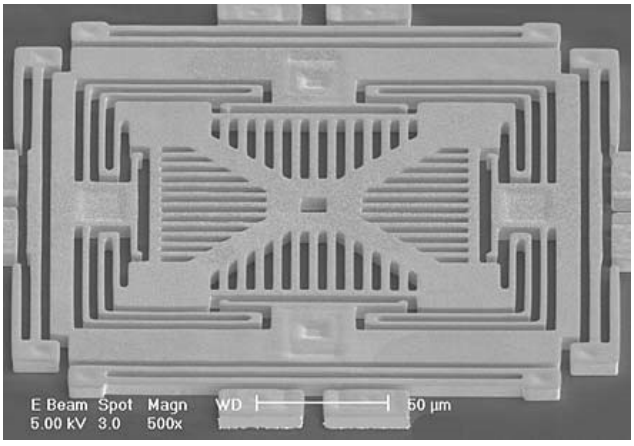


Ilustración 3. Ampliación de un acelerómetro de tres ejes con una única masa. Fuente Sensorsmag.com.

Los acelerómetros MEMS más comunes miden las fuerzas lineales aplicadas a una masa interna mediante una estructura elástica dotada de sensores capacitivos móviles. Las fuerzas aplicadas sobre la masa producen una elongación/contracción de la estructura-muelle que la sostiene en la dirección opuesta a la fuerza. Este desplazamiento se mide mediante condensadores entre las partes móviles. Conocidas la constante del muelle y la masa se puede medir la aceleración aplicada al sistema. (4)

Existen acelerómetros MEMS que emplean otras formas de medir las aceleraciones, como sistemas piezoeléctricos, ópticos o electromagnéticos. Es habitual el uso de acelerómetros de tres ejes ortogonales para obtener tres grados de libertad. (4)

Los acelerómetros tienen un gran inconveniente a la hora de capturar la orientación de un cuerpo. No pueden calcular el ángulo de giro sobre el eje de la gravedad que representa la orientación del cuerpo sobre la superficie terrestre. Al ejercer la gravedad

una fuerza constante sobre la masa del sensor, pueden existir dos medidas iguales a cero simultáneamente en el sensor. Si el sensor tiene un eje perpendicular a la gravedad, las lecturas estáticas serán ±1g para el eje en cuestión, y 0g para los otros dos. Rotar el sensor alrededor del eje perpendicular de la gravedad no modificará las lecturas.

2.1.2.1.2. Acelerómetros + giroscopios

Una solución parcial a este problema es la integración de giroscopios. Empleados de forma conjunta con los acelerómetros constituyen una IMU (*Inertial Measurement Unit*). La mayoría de los giroscopios MEMS hacen uso de la aceleración de Coriolis, proporcional a la frecuencia de oscilación, la masa y a la velocidad angular. Existen dos tipos de giroscopios MEMS ampliamente utilizados, los giroscopios de estructura vibrante (*Tuning fork*) y los de rueda vibrante.

Los giroscopios de estructura vibrante hacen oscilar dos masas a la misma frecuencia, pero en sentido opuesto. Al girar la estructura resonante, la fuerza de Coriolis genera una vibración que es recogida por sensores capacitivos móviles. Se emplean para medir la velocidad angular de los ejes X e Y por su forma plana. (4)

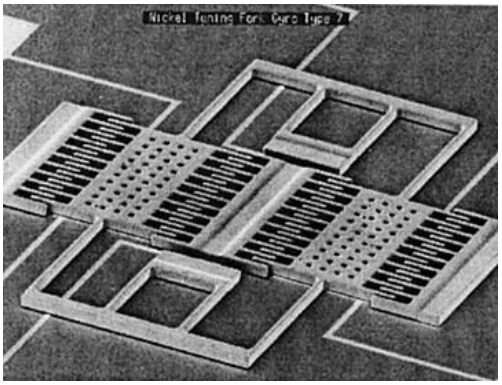


Ilustración 4. Ampliación de un giroscopio de estructura vibrante. Fuente sensorsmag.com.

Los giroscopios de rueda giratoria vibran sobre un eje de rotación. Los giros externos aplicados al sensor son captados por sensores capacitivos. Estos sensores se emplean para medir la velocidad angular del eje Z.

Con la velocidad angular realiza una integración para conocer la posición angular del sensor.

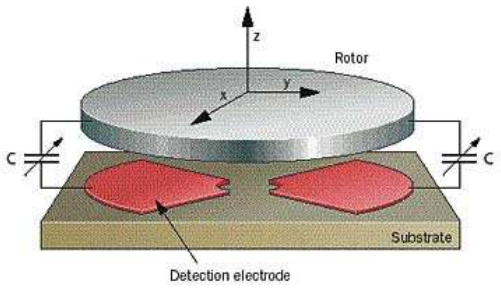


Ilustración 5. Ampliación de un giroscopio de rueda vibrante. Fuente sensorsmag.com.

Para extraer una orientación 3D a partir de las seis medidas de una IMU se emplean algoritmos de fusión sensorial. El empleo de las IMU aporta el giro alrededor del eje Z, pero aportan una deriva incremental al sistema, por lo que no es recomendable emplear los giroscopios como único sistema de referencia.

2.1.2.1.3. Acelerómetros + giroscopios + magnetómetros

Debido a la deriva incremental de la integración de los giroscopios y a la posible existencia de aceleraciones lineales, es habitual añadir un nuevo sensor (como un magnetómetro en aplicaciones de orientación, o un GPS en aplicaciones de navegación) con una baja frecuencia de muestreo. Este sensor corrige las derivas estimadas del sistema inercial de alta frecuencia de muestreo (acelerómetros + giroscopios).

Al igual que en la solución anterior, es necesario emplear algoritmos de fusión sensorial, como las Matrices de Cosenos Directores o filtros de Kalman (estándar o extendido), para obtener la orientación del sensor. Estos algoritmos han resultado ser soluciones robustas al problema de filtrar datos no confiables con *feedback*. (6) (7)

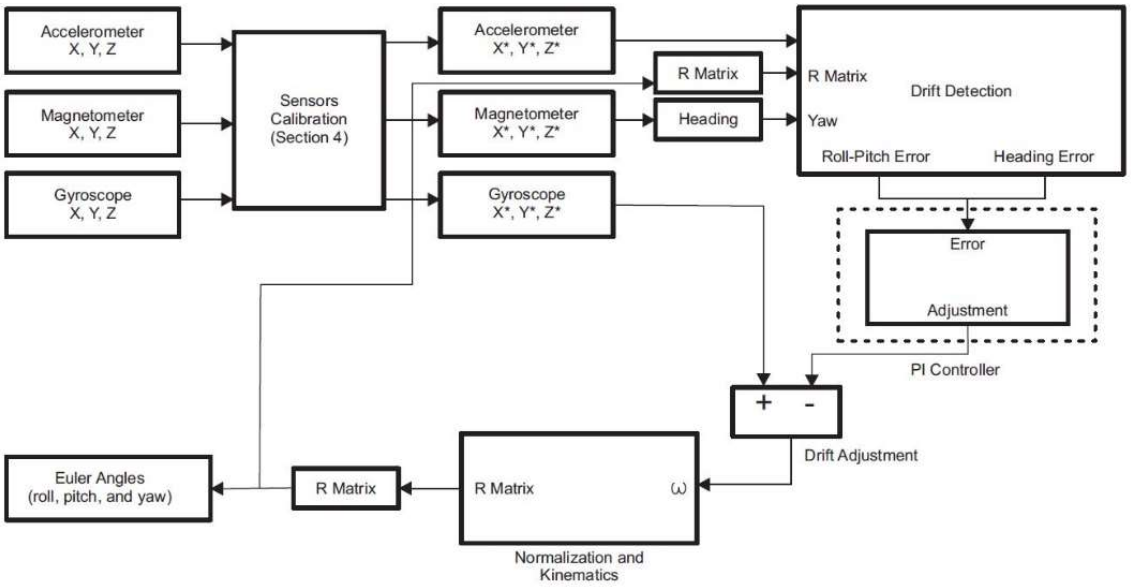


Ilustración 6. Algoritmo de fusión sensorial con una Matriz de Cosenos Directores (DCM) para extraer ángulos de Euler propuesto por Texas Instruments. Fuente ti.com.

Si al montaje de la IMU con magnetómetro se le añade una CPU (microprocesador o DSP) que compute el algoritmo de fusión sensorial el sistema resultante se conoce como AHRS (*Attitude and Heading Reference Systems*). Estos dispositivos generalmente tienen más precisión que los métodos únicamente inerciales, pero tienen un coste asociado más alto que los hace no aptos para aplicaciones sensibles al coste.

2.2. Representación de la orientación

Para la representación de un cuerpo sólido en el espacio se hace uso de la teoría de los espacios euclídeos. Todo cuerpo en un espacio euclídeo puede representarse por tres coordenadas referenciadas a tres ejes ortogonales de referencia.

2.2.1. Matrices de rotación y ángulos de Euler

En un espacio euclídeo es posible realizar una rotación de los ejes de referencia manteniendo la ortogonalidad. Una matriz de rotación contiene los parámetros para transformar una posición referenciada a un sistema de coordenadas a otro diferente. Con este cambio de ejes se puede representar un cambio de orientación de un cuerpo como un cambio en sus ejes de referencia mediante una matriz de transformación.

Los tres ángulos que definen las matrices de rotación para representar la orientación de un cuerpo frente a un sistema de coordenadas fijo (el original) se les denomina ángulos de Euler. Se denominan ángulos Roll, Pitch y Yaw a las rotaciones sobre el eje X, Y y Z respectivamente. (8)

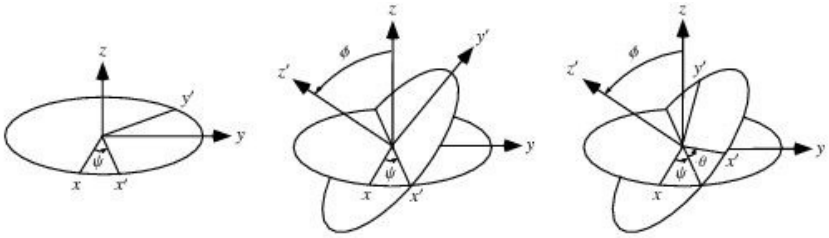


Ilustración 7. Representación de rotaciones con ángulos de Euler. Fuente mathworld.wolfram.com.

Se detalla este método por ser el método empleado para realizar la estimación de orientación de los sensores. Las matrices de rotación de los ángulos de Euler para giros en el sentido de las agujas de reloj son:

$$Rx = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix}, Ry = \begin{bmatrix} cos\vartheta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix}, Rz = \begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La rotación (o cambio de orientación) de un cuerpo se puede definir por una matriz de rotación que multiplique las tres matrices anteriores. Existen dos posibles resultados que sólo involucran los ángulos Roll (ϕ) y Pitch (θ) para definir la orientación de un cuerpo, y son las rotaciones conmutativas XYZ y las rotaciones conmutativas YXZ. Estas son las rotaciones que nos interesan, ya que no es posible medir el ángulo Yaw (ψ) con acelerómetros. (6) (10)

Para emplear estas matrices e integrarlas con los datos de los acelerómetros se parte del sistema referencia terrestre, donde la gravedad está orientada sobre el eje Z. Un cambio de orientación de los ejes de referencia a los del sensor debe distribuir esa fuerza gravitacional en los ejes X, Y y Z del sensor en función de su orientación con la forma $A = R\ (g - a)$, siendo *A* la salida del acelerómetro, *R* la matriz de rotación que indica la orientación del sensor, *g* la gravedad y *a* las aceleraciones externas. Para esta estimación de orientación es necesario que no existan aceleraciones lineales externas que puedan interferir con la medida de la fuerza de la gravedad. La matriz de rotación de los ángulos de Euler en el orden XYZ resultante de la multiplicación de matrices es la siguiente.

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = R_x(\phi)R_y(\theta)R_zx(\psi)\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix}\begin{bmatrix} cos\vartheta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix}\begin{bmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} -sin\theta \\ cos\theta sin\phi \\ cos\theta cos\phi \end{bmatrix}$$

Se normalizan los valores. $\frac{1}{\sqrt{{a_x}^2+{a_y}^2+{a_z}^2}}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} -sin\theta \\ cos\theta sin\phi \\ cos\theta cos\phi \end{bmatrix}$

Resolviendo la igualdad para los ángulos Roll y Pitch obtenemos las ecuaciones que relacionan las aceleraciones medidas con la orientación del sensor se obtiene lo siguiente.

$$Roll\ (\phi) = \tan^{-1}\left(\frac{a_y}{a_z}\right)$$

$$Pitch\ (\theta) = \tan^{-1}\left(\frac{-a_x}{\sqrt{{a_y}^2 + {a_z}^2}}\right)$$

Observando las fórmulas obtenidas podemos advertir un problema en el cálculo del ángulo ϕ cuando el eje X está paralelo al vector de la gravedad. Según vayan disminuyendo las aceleraciones de los ejes Y y Z la fracción irá acercándose a $\frac{0}{0}$, indeterminación matemática que dejará sin estimación de Roll al sistema. Se emplea la siguiente aproximación.

$$Roll\ (\phi) = \tan^{-1}\frac{a_y}{sign(a_z)\cdot\sqrt{{a_z}^2+\mu\cdot{a_x}^2}}$$

La solución pasa por incluir en el cálculo una fracción (μ) del orden de una centésima de la aceleración leída en el eje X (a_x) para evitar que este pueda tomar valores de cero. Se incluye también el signo de la aceleración del eje Z (a_z), pues este se pierde al realizar la elevación al cuadrado y posterior raíz.

La aproximación hace que en ausencia de ángulo θ, el denominador tenga el mismo valor que en la ecuación original. Cuando el ángulo real ϕ es nulo (no hay rotación sobre el eje X), la medida del acelerómetro del eje Y (a_y) es cero, lo que hace que el ángulo calculado sea también cero. Para el resto de las situaciones el peso de la aceleración del eje Z (a_z) tendrá más peso que el del eje X (a_x), por lo que la variación del denominador será mínima. De esta forma, aunque el valor del ángulo calculado no sea exacto al aproximarse el eje X al eje de la gravedad (ya que se está forzando la aproximación a 0 del mismo), la función pierde la inestabilidad.

2.2.2. Cuaterniones

Los cuaterniones representan un espacio cuatridimensional sobre el que definen una orientación. Tienen una base matemática similar a la de los números complejos, pero en un espacio tridimensional, lo que hace que operaciones como la multiplicación entre cuaterniones no sea conmutativa. Se emplean a la hora de representar espacios tridimensionales en los que intervienen rotaciones y/o escalados. (11)

Los cuaterniones pueden simplificarse a nivel conceptual como un vector y un ángulo de giro sobre el mismo, de modo que toda la rotación tiene lugar en el sistema referencial inicial. Aunque son mucho más eficaces que los ángulos de Euler (puesto que no existe el concepto de *gimball lock,* no se pierden grados de libertad ni existen singularidades), su computo es notablemente más complejo que los ángulos de Euler.

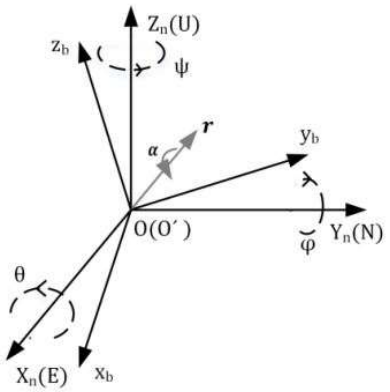


Ilustración 8. Quaternion (r,α). Fuente journals.sagepub.com.

3. Tecnología empleada

Este capítulo presenta la tecnología y los diferentes dispositivos hardware y software empleados para el desarrollo del proyecto. Se realiza una descripción funcional de los elementos, así como un listado de las características principales de los mismos.

3.1. Hardware

3.1.1. Controlador Arduino

Arduino es una plataforma *open-source* de desarrollo electrónico con finalidad educacional que manufactura microcontroladores programables. Integra un IDE (*Integrated Development Environment*) desde el que programar dichos microcontroladores. El proyecto Arduino engloba tanto el hardware como el software, por lo que toda la documentación sobre las placas está disponible para la comunidad; desde los diseños electrónicos de las PCBs (*Printed Circuit Board*), a los *firmwares*, *bootloaders* y el IDE, todo es público y está licenciado para que cualquiera pueda usar y modificar los componentes. El proyecto Arduino deriva del proyecto Wiring, plataforma *hardware* con la misma filosofía que Arduino, que deriva a su vez del proyecto Processing. (7)

Dado el cariz educacional del proyecto Arduino, las placas tienen un bajo coste, lo que las ha convertido mundialmente en un referente de prototipado rápido entre entusiastas de la electrónica. Las placas integran, además del microcontrolador, la electrónica pasiva necesaria para hacerlo funcionar, un puerto USB y puertos I/O analógicos y digitales. Gracias a esto, es posible realizar desarrollos software en lenguajes de alto nivel (C++ y un lenguaje propietario similar a Wiring) para controlar entornos electrónicos normalmente desarrollados en lenguajes de bajo nivel (ensamblador).

Existe una gran variedad de placas oficiales compatibles con la plataforma de Arduino, cada una enfocada a un área diferente. Se pueden encontrar en la tienda del proyecto oficial desde placas de desarrollo general (UNO y Genuino), a placas centradas en la conectividad (YUN y MKR), placas para wearables (Lilypad) o para controladores (Esplora).

Existen módulos de ampliación (llamados *shields*) para dotar de funciones adicionales a las placas ya existentes, ampliando conectividad (bluetooth, wifi, CanBus, RS485, GSM, Ethernet…), almacenamiento (tarjetas SD), sistemas de potencia (control de motores, relés…) e incluso placas de desarrollo vacías para que sea el usuario quien decida qué

componentes montar. Arduino dispone de una gran comunidad de usuarios que desarrollan librerías y módulos de hardware para simplificar la redacción del código y ampliar las capacidades de las placas.

Para este proyecto se emplea una placa **Arduino Uno** (rev3). Esta placa es la más usada en el entorno de Arduino, lo que hace que sea la que mejor documentada se encuentre. No se considera necesario instalar ningún *shield* adicional. (8)

Las características técnicas más relevantes de la placa Arduino UNO rev3 son:

- Conector de alimentación externa de $2{,}1\ mm$ ($7 \div 12\ V$).
- Transformador de tensión a $3{,}3\ V$ ($50mA$).
- Cristal de cuarzo de $16\ MHz$.
- Puerto USB-B.
- Puerto ICSP de 6 pines.
- Microcontrolador ATmega328P de $8\ bits$.
- $32\ kB$ de memoria *flash* para código (0,5kB reservados para el *bootloader*).
- $2\ kB$ de memoria SRAM (*Static Random-Access Memory*) para variables.
- $1\ kB$ de memoria EEPROM (*Electrically Eraseble Read-Only Memory*) para datos.
- 14 pines I/O digitales TTL (*Transistor-Transistor Logic*), de los cuales:
  - 6 (pines 3, 5, 6, 9, 10 y 11) son también salidas PWM.
  - 2 (pines 1 y 2) son también un puerto serie UART.
  - 4 (pines 10, 11, 12 y 13) son también un puerto SPI.
  - 2 (pines 2 y 3) son también entradas de interrupción hardware.
- 6 pines de entradas analógicas con $10\ bits$ de resolución, de los cuales:
  - 2 (pines A4 y A5) funcionan también como puerto I^2C.

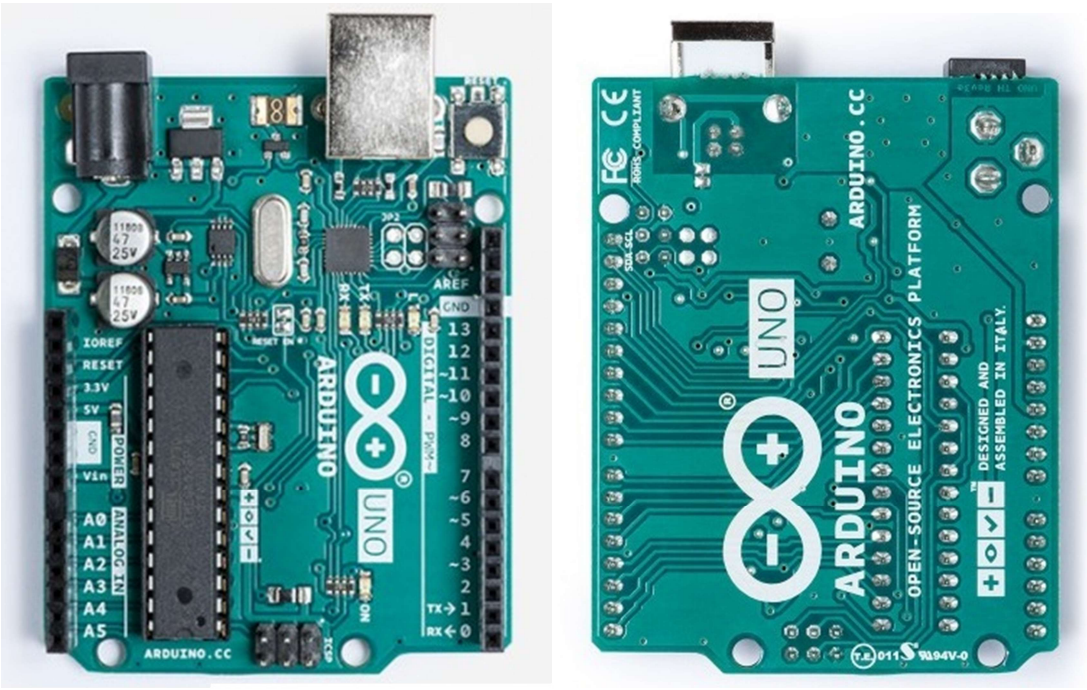


Ilustración 9. Arduino UNO rev3 frontal y reverso. Fuente store.arduino.cc.

3.1.1.1. ATmega328P

El microcontrolador ATmega328P de Atmel es el núcleo del Arduino. Está basado en la arquitectura RISC (*Reduced Instruction Set Computer*) de AVR y permite ejecutar 131 instrucciones distintas de código ensamblador. (9)

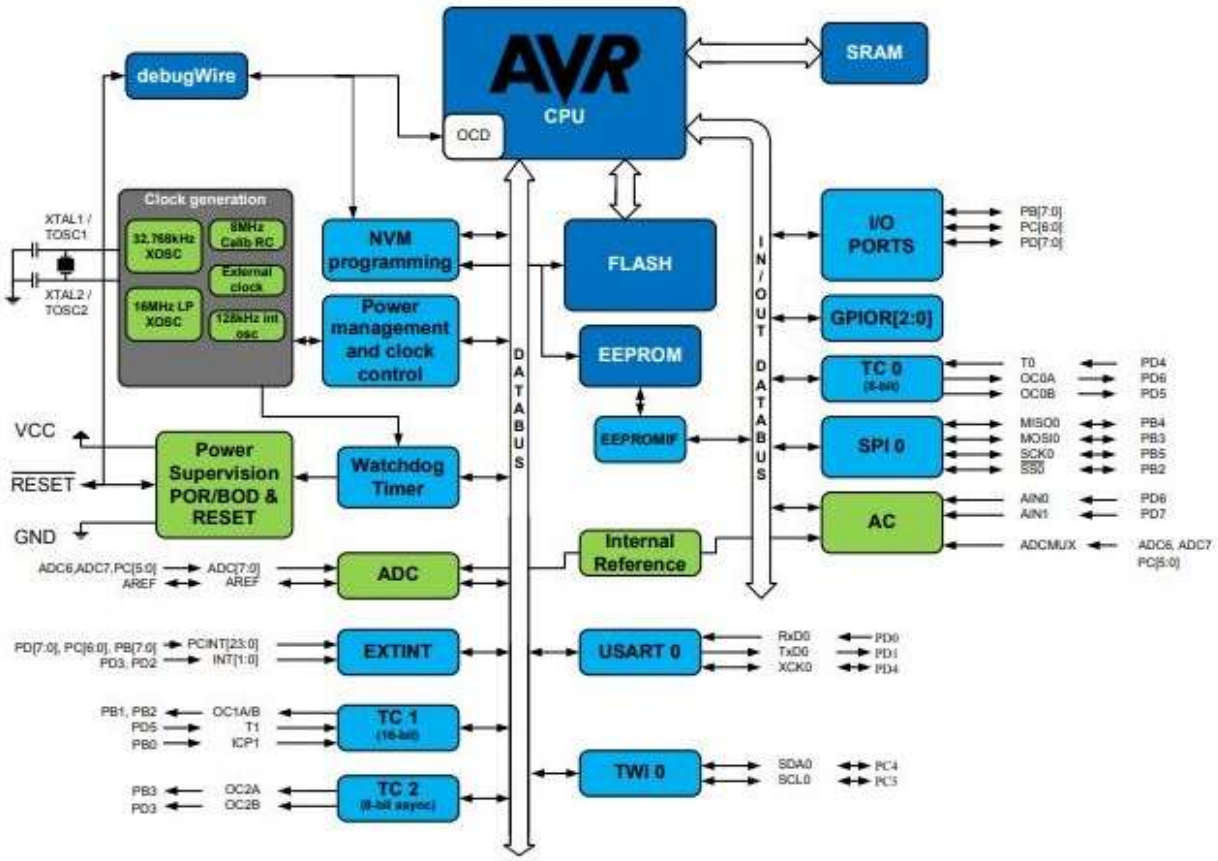


Ilustración 11. Microcontrolador ATmega328P de Atmel. Fuente datasheet.

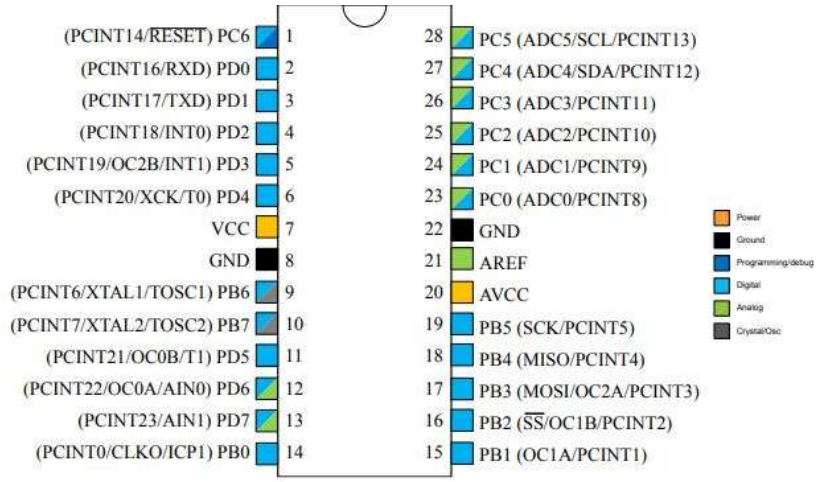


Ilustración 10. Diagrama de pines del microcontrolador ATmega328P de Atmel. Fuente datasheet.

El microcontrolador integra:

- 1 CPU.
- 1 generador de pulsos de reloj (Clock generation).
- 1 módulo de gestión de consumo (Power management and clock control).
- 1 módulo de control del sistema y reset (POR/BOD & RESET).
- 1 gestor de interrupciones externas (EXTINT).
- 1 *watchdog* programable (Watchdog timer).
- $32\ kB$ de memoria para código (FLASH).
- $2\ kB$ de memoria para variables (SAM).
- $1\ kB$ de memoria para datos (EEPROM).
- 2 temporizadores/contadores de 8 bits (TC0 y TC2).
- 1 temporizador/contador de 16 bits (TC1).
- 1 conversor analógico-digital de 10 bits (ADC).
- 1 gestor de puertos de entradas y salidas (I/O PORTS).
- 1 gestor de comunicación serie (SPI).
- 1 gestor de comunicación asíncrona (USART 0).
- 1 gestor de comunicaciones a dos hilos (TWI 0).
- 1 comparador analógico (AC).
- 1 debugeador (debugWire).

Es interesante que el microcontrolador disponga un modo de consumo reducido, ya que en el caso de operar en un futuro el dispositivo con baterías, estas tendrán una autonomía notablemente mayor. El consumo en estado activo es de $5{,}2\ mA$, en un estado *idle* de $1{,}2mA$ y en un estado de bajo consumo de $0{,}9\ \mu A$. Una reducción del voltaje del procesador implica también una reducción de consumo de este, a costa de una reducción de la frecuencia de trabajo.

Es importante saber también que las entradas analógicas sólo cuentan con un único ADC, por lo que no será posible realizar las lecturas de todas a la vez, sino que habrá que realizarlo secuencialmente. Según la hoja de características del fabricante, la lectura de una entrada analógica tarda $13 \div 260\ \mu s$, lo que puede condicionar el rendimiento de una aplicación que haga uso de forma intensiva del ADC.

3.1.2. Acelerómetros

Para este proyecto se emplean acelerómetros de 3 ejes ADXL335 de Analog Devices. Estos acelerómetros están enfocados a aplicaciones de bajo coste y bajo consumo, tanto para mediciones de movimiento como de posición.

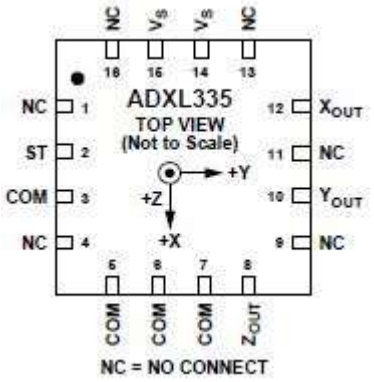


Ilustración 13. Diagrama de pines del acelerómetro ADXL335 de Analog Devices. Fuente datasheet.

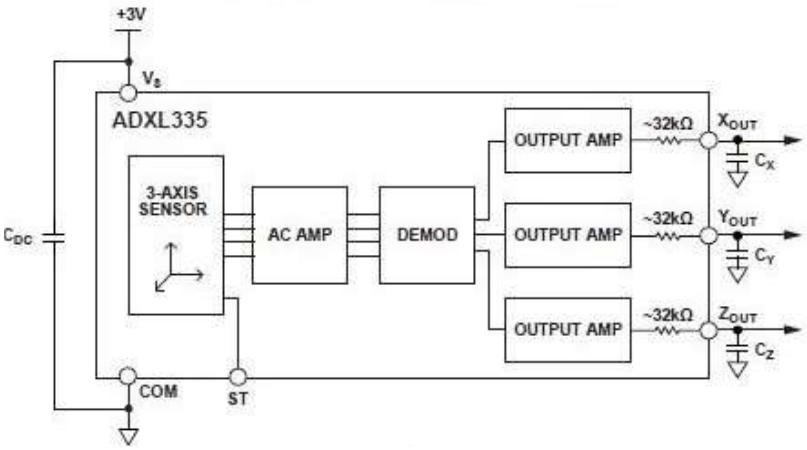


Ilustración 12. Diagrama funcional del acelerómetro ADXL335 de Analog Devices. Fuente datasheet.

Cada acelerómetro emplea una única superficie micro-tratada de poli-silicona suspendida mediante muelles. La compresión o elongación del muelle se mide empleando condensadores diferenciales sujetos a la estructura móvil. La amplitud de la onda en la salida del condensador es proporcional a la aceleración aplicada al conjunto, que es amplificada por el módulo AC AMP. La señal recibe una demodulación sensible a la fase en el módulo DEMOD para determinar la magnitud y dirección de la aceleración, y descomponer la misma en las aceleraciones propias de los tres ejes de referencia del acelerómetro. Esta es amplificada por los módulos OUTPUT AMP que dan, tras una resistencia interna del circuito, a los pines de salida. (10)

Al ser una única pieza mecánica la encargada de recoger las aceleraciones que ocurren en el sistema existe muy poco error en la medida, ya que las aceleraciones resultantes son altamente ortogonales y tienen poca sensibilidad inter-axial. El error más común en esta construcción, según el fabricante, es la desviación de los ejes internos del sensor mecánico y los del encapsulado, pero estos pueden ser medidos y calibrados a nivel de software.

Las características técnicas más relevantes de los sensores son:

- Rango de medida mínimo de ±3g, llegando hasta ±3,6g.
- No linealidad ($\pm 0.3\ \%$) y desviación térmica ($\pm 0.01\ \%/ºC$) despreciables.
- Sensibilidad relativa a la tensión de entrada ($0{,}3\ V/g\ para\ V_s = 3V$).
- Tensión de entrada variable ($1{,}8 \div 3{,}6\ V$).
- Muy bajo consumo ($350\ \mu A$).

Para este proyecto se emplean placas soldadas de dos fabricantes diferentes, Lilypad (un proyecto basado en Arduino enfocado a *wearables*) y Sparkfun (un distribuidor electrónico referente en prototipado y diseños *open-source*).

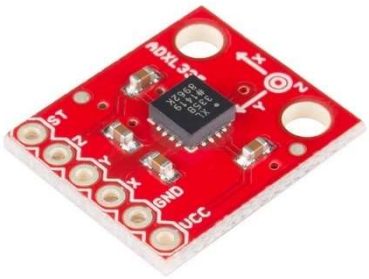


Ilustración 14. Placa de prototipado de Sparkfun para el acelerómetro ADXL335. Fuente Sparkfun.com.

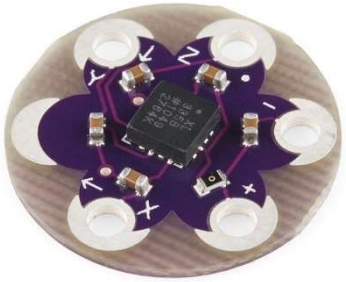


Ilustración 15. Placa de prototipado de Lilypad para el acelerómetro ADXL335. Fuente Sparkfun.com.

Si se analiza el esquema de Sparkfun (Anexo III.II. Placa de prototipos Sparkfun para ADXL335) se advierte que el fabricante ha seleccionado un ancho de banda de 50Hz conectando un condensador de 0,1µF entre el pin de salida (XOUT, YOUT, ZOUT) y tierra (GND). También ha conectado un condensador de 0,1µF entre el pin de entrada de tensión (VCC) y tierra (GND) para estabilizar y desacoplar la tensión de entrada.

Si se analiza el esquema de Lilypad (Anexo III.I. Placa de prototipo Lilypad para ADXL335) se advierte que el fabricante ha seleccionado el mismo ancho de banda. Sin embargo, se observa que se ha colocado un filtro pasivo paso-bajo para eliminar las distorsiones de alta frecuencia en la alimentación de la placa, a la vez que se ha mantenido el condensador de desacople de tensión entre el pin de entrada (VCC) y el de tierra (GND). Esto debería resultar en unas medidas ligeramente más estables que las de la placa de Sparkfun.

3.1.3. Multiplexores

Para este proyecto se emplean multiplexores CD4051B CMOS de 8 canales analógicos de Texas Instruments. Estos multiplexores están enfocados a sensórica y electrónica de consumo.

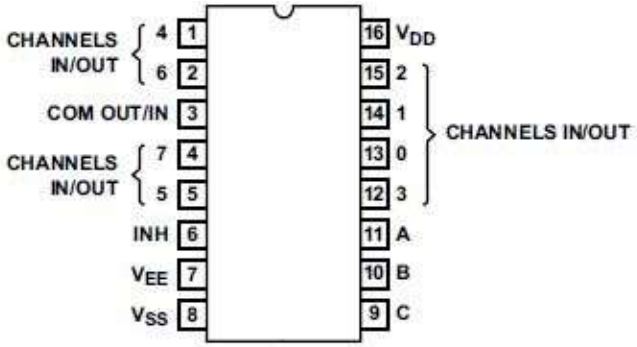


Ilustración 16. Diagrama de pines del multiplexor CD4501B de Texas Instruments. Fuente datasheet.

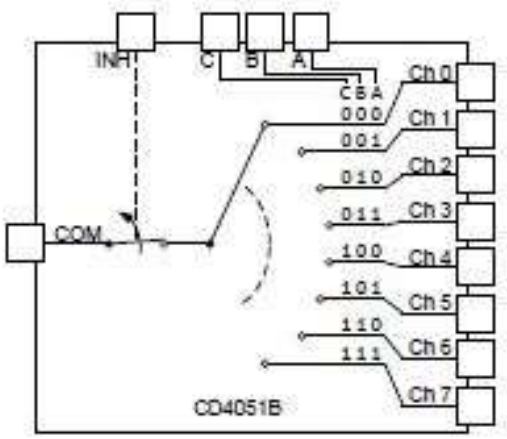


Ilustración 17. Diagrama funcional del multiplexor CD4501B de Texas Instruments. Fuente datasheet.

Los multiplexores CD4501B emplean los pines digitales A, B y C para seleccionar el canal a conectar con el pin COM. Dispone de un pin para inhibir el enrutamiento (INH) que permite desacoplar los circuitos a voluntad. (11)

Las características técnicas más relevantes de los multiplexores son:

- Tensión de entrada variable ($-0{,}5 \div 20\ V$).
- Señales analógicas y digitales.
- Baja resistencia ON ($470\ \Omega$).
- Alta resistencia OFF, baja corriente de fuga ($\pm 0{,}1\ nA$).
- Muy bajo consumo residual ($0{,}04\ \mu A$).
- Bajo tiempo de respuesta ($450\ ns$).

Para este proyecto se emplean tres multiplexores, uno para cada eje de referencia (X, Y y Z). Cada uno recoge las medidas de un eje de los acelerómetros a través de cuatro canales y las saca por el puerto COM. Se emplean los pines A, B y C para seleccionar el puerto a leer en cada ocasión.

Ilustración 18. Encapsulado de multiplexor CD4051B de Texas Instruments. Fuente ti.com.

3.1.4. Temporizador

Para este proyecto se emplea un temporizador NE555 de Texas Instruments. Estos temporizadores están enfocados a aplicaciones de reconocimiento, pero pueden ser usados para propósitos generales, ya que son capaces de realizar temporizaciones y oscilaciones cíclicas.

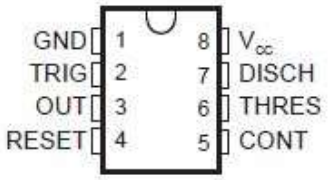


Ilustración 20. Diagrama de pines del temporizador NE555 de Texas Instruments. Fuente datasheet.

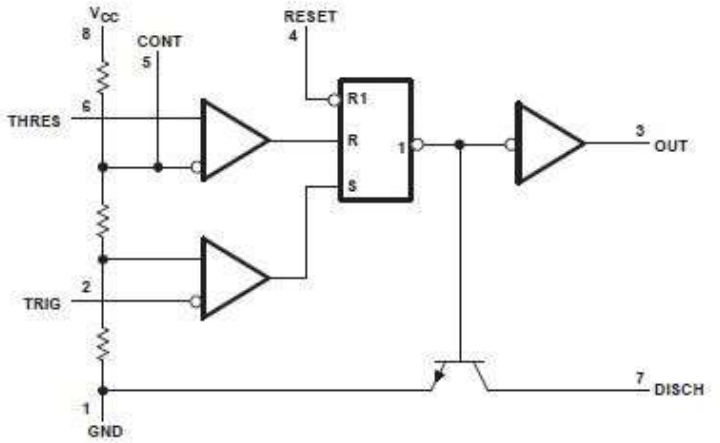


Ilustración 19. Diagrama funcional del temporizador NE555 de Texas Instruments. Fuente datasheet.

El temporizador cuenta con una báscula biestable SR. Cuando la entrada de TRIG baja del umbral de disparo (*trigger*), el biestable pasa activarse SET y el pin de salida OUT se activa (*high*). Si la entrada TRIG sobrepasa el umbral de *trigger* y la entrada THRES sobrepasa el umbral límite (*threshold*), el biestable se resetea y la salida se desactiva. El estado desactivado de la entrada de RESET sobrescribe el valor del biestable, desactivando la salida. (12)

En función de las conexiones eléctricas que se realicen en el temporizador este podrá funcionar en modo monoestable (pulsos a intervalos regulares), astable (con un *duty cycle* regulable) o divisor de frecuencia. El fabricante propone diversas aplicaciones prácticas, como la detección de ausencia de pulsos, PWM (*Pulse-Width Modulation*), PPM (*Pulse-Position Modulation*) o la temporización secuencial.

Las características técnicas más relevantes de los multiplexores son:

- Tensión de entrada variable ($4{,}5 \div 18\ V$).
- Bajo tiempo de activación/desactivación ($100\ ns$).
- Gran rango de temporización ($10\ \mu s\ \div > 1\ h$).
- Gran rango de oscilación ($< 1\ mHz\ \div 100\ kHz$).

Para este proyecto se emplea el temporizador como báscula astable para generar una onda cuadrada de frecuencia fija.

3.2. Software

3.2.1. Processing

Processing es un proyecto *open-source* desarrollado por Ben Fry y Casey Reas enfocado en la representación gráfica mediante instrucciones simples. Es un software multiplataforma, con soporte para Windows, Mac y Linux. Pensado para realizar prototipos software, está considerado como el origen de Wiring, precursor de Arduino. El IDE integra instrucciones en un lenguaje propio, pero permite a los usuarios crear y publicar librerías personalizadas a fin de ampliar las funcionalidades del software. (13)

Ilustración 21. Display (izquierda) y editor de código (derecha) de Processing. Fuente processing.org.

Processing trabaja con dos elementos, un editor de código avanzado y un display que muestra el programa en ejecución. Se emplea por defecto un área de trabajo (*canvas*) bidimensional. El sistema de coordenadas tiene el origen en la esquina superior izquierda. Es posible emplear diferentes *renderers* (módulo de software encargado de procesar y graficar las imágenes), pudiendo realizar una vista plana o una vista con volumen que permite trabajar en un entorno 3D.

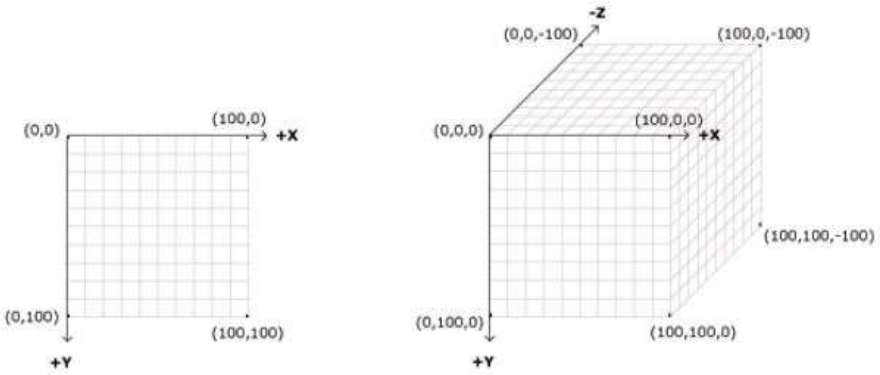


Ilustración 22. Sistemas de coordenadas de los canvas 2D y 3D en processing. Fuente processing.org.

El programa a ejecutar en Processing está dividido en dos secciones principales: el `setup()`, que se ejecuta una única vez al iniciar el programa, y el `draw()`, que se ejecuta cíclicamente para realizar el procesado gráfico.

Processing está orientado a objetos, por lo que es posible generar clases, definiendo constructores y métodos. Emplea un lenguaje propio que combina estructuras de control, interacciones de usuario, funciones matemáticas y estructuras de datos similares a otros lenguajes de programación con funciones enfocadas al diseño gráfico, como formas, colores, cámaras, imágenes o textos.

3.2.1.1. Peasycam

Peasycam es una librería *open-source* para Processing desarrollada por Jonathan Feinberg para integrar de forma simple e intuitiva el control de la cámara en un *canvas* 3D. Eso permite cambiar el punto de vista sobre el que se ve el display sin interferir con los elementos presentes en el mismo. Permite emplear el ratón para mover la cámara alrededor del punto central del *canvas* haciendo *click* izquierdo. Permite también realizar *zoom* pulsando *click* derecho y rotar sobre ejes fijos con la tecla *Shift*. Un doble *click* permite restaurar la vista por defecto del cámara. (19)

3.2.2. IDE Arduino

El IDE de Arduino es el software de programación para las placas de Arduino. Es un software multiplataforma, con soporte para Windows, Mac y Linux. Está basado en Processing y deriva del trabajo realizado en Wiring, un *framework open-source* de programación para microcontroladores enfocado al prototipado.

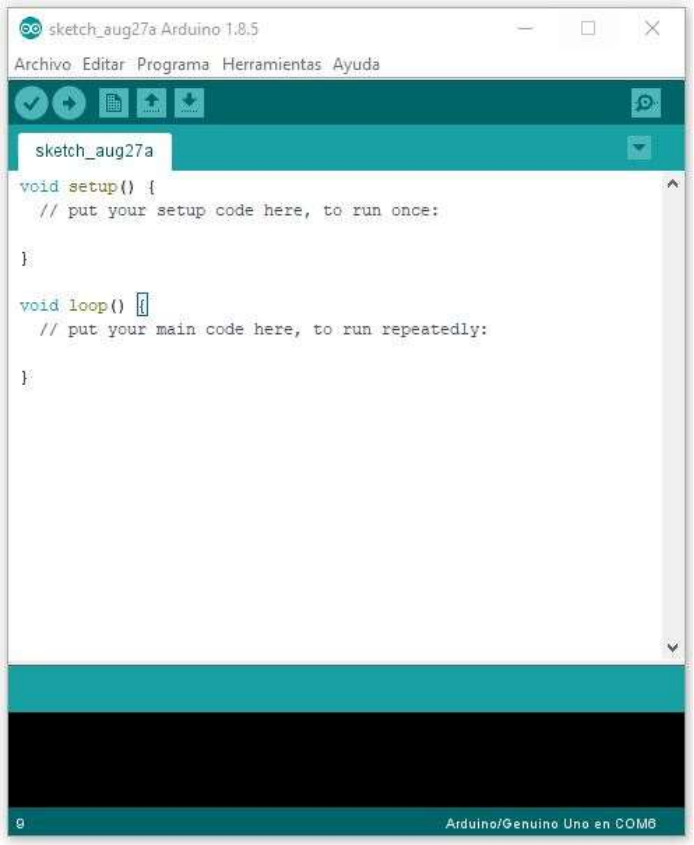


Ilustración 23. Entorno de desarrollo de Arduino.

El entorno de desarrollo es muy similar al de Processing, pero al no estar enfocado a un entorno gráfico de ejecución no existe una ventana de *canvas*. El IDE integra un monitor del puerto serie para depurar el programa en ejecución con la placa.

El programa a ejecutar en la placa Arduino, de forma análoga a como se hacía en Processing, está dividido en dos secciones principales: el `setup()`, que se ejecuta una única vez al iniciar el programa, y el `loop()`, que se ejecuta cíclicamente.

La programación de Arduino es similar a la de Processing, sólo que en está orientada a un entorno de más bajo nivel (microprocesadores). Esto hace que no existan instrucciones como el *cath&try*, ni funciones gráficas, pero existen en cambio funciones enfocadas a las entradas/salidas físicas de la placa, a comunicaciones serie o a rotaciones de bits.

4. Proyecto desarrollado

El proyecto desarrollado está formado por tres sistemas:

Un sistema eléctrico compuesto por la placa Arduino, cuatro acelerómetros (para la medida de los brazos y antebrazos), tres multiplexores (para leer doce señales con tres entradas analógicas), un temporizador (para generar un tren de pulsos) y elementos de electrónica pasiva. El sistema eléctrico se desarrolla sobre placas de prototipado.

Un programa ejecutado en la placa de Arduino que realiza la extracción de los datos de los sensores y el envío a un ordenador a través de una comunicación serie (utilizando un cable USB).

Un programa ejecutado en un ordenador en Processing, que realiza la lectura de los datos del puerto serie, el procesamiento para transformar las aceleraciones en ángulos Roll y Pitch (puesto que sólo se dispone de acelerómetros), y la representación gráfica.

4.1. Sistema eléctrico

Para el montaje eléctrico se aprovecha el trabajo de cableado realizado por Javier Garrido Nieto, en su trabajo de fin de máster “Sistema de captura de movimientos basado en sistemas electrónicos de bajo coste” en el año 2017.

Se eliminan tres de los siete acelerómetros para representar únicamente el movimiento de los brazos y antebrazos del usuario. Se emplean dos acelerómetros de Sparkfun y dos acelerómetros de Lilypad, empleando cada conjunto para un brazo y antebrazo. El brazo y antebrazo izquierdos están instrumentado por Sparkfun y los derechos por Lilypad.

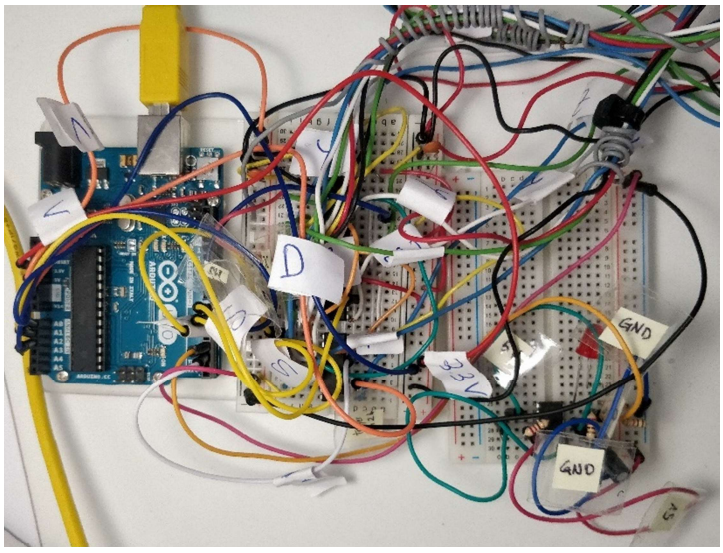


Ilustración 24. Sistema eléctrico implementado.

Se añade un circuito secundario con un temporizador NE555 de Texas Instruments para gestionar las interrupciones cíclicas externas. Se persigue obtener un oscilador que despierte al microcontrolador a una frecuencia determinada.

4.1.1. Temporizador NE555

Se realiza un conexionado del temporizador NE555 para que funcione en modo astable. Los pines de *trigger* y de descarga quedan unidos por la resistencia R_B, y el pin de *threshold* queda unido al de *trigger*. Esto provoca que el temporizador realice ciclos de carga y descarga del condensador C entre los valores de *threshold* ($0{,}67\ Vcc$) y trigger ($0{,}33\ Vcc$). Los tiempos de carga y descarga del condensador son independientes de la tensión de alimentación del circuito. El modo astable funciona de forma autónoma (sin señales externas).

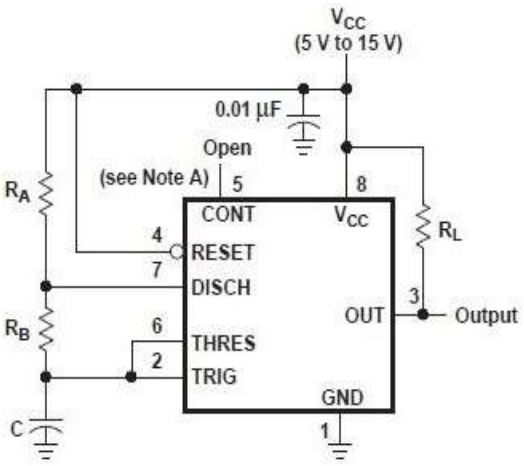


Ilustración 25. Conexión astable del temporizador NE555. Fuente datasheet.

El funcionamiento del temporizador en modo astable viene definido por las siguientes funciones, siendo T_H el tiempo que dura la salida activada y T_L el tiempo de que dura la salida desactivada:

$$T_H = 0{,}693\ (R_A + 2R_B)\ \times\ C$$

$$T_L = 0{,}693\ (R_B)\ \times\ C$$

El tiempo de ciclo del temporizador es:

$$T = T_H + T_L = 0{,}693\ (R_A + 2R_B)\ \times\ C$$

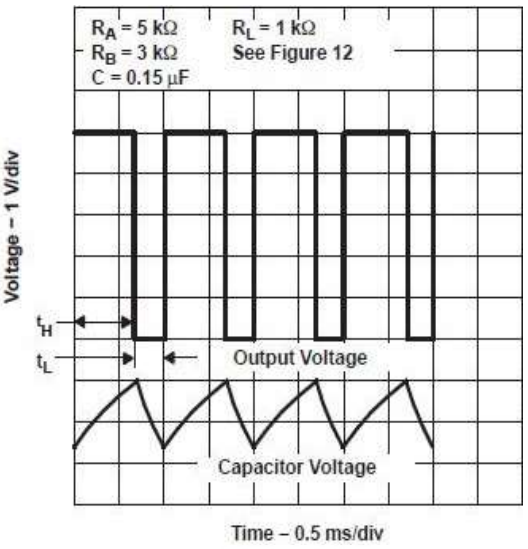


Ilustración 26. Onda astable del temporizador NE555. Fuente datasheet.

Se desea obtener una tasa de refresco de datos unos 20 Hz, por lo que, empleando el material disponible, se escogen los valores $R_A = 220\ \Omega$, $R_B = 1\ k\Omega$, $R_L = 220\ \Omega$, y $C = 100\ \mu F$.

El circuito tiene un tiempo de ciclo teórico de $\mathbf{45{,}73\ ms}$**.** Realizando mediciones con el Arduino se observa que el tiempo medio de ciclo generado por el temporizador está en torno al valor previsto, lo que nos da una frecuencia de trabajo de unos **22 Hz**.

4.1.2. Esquema de conexiones

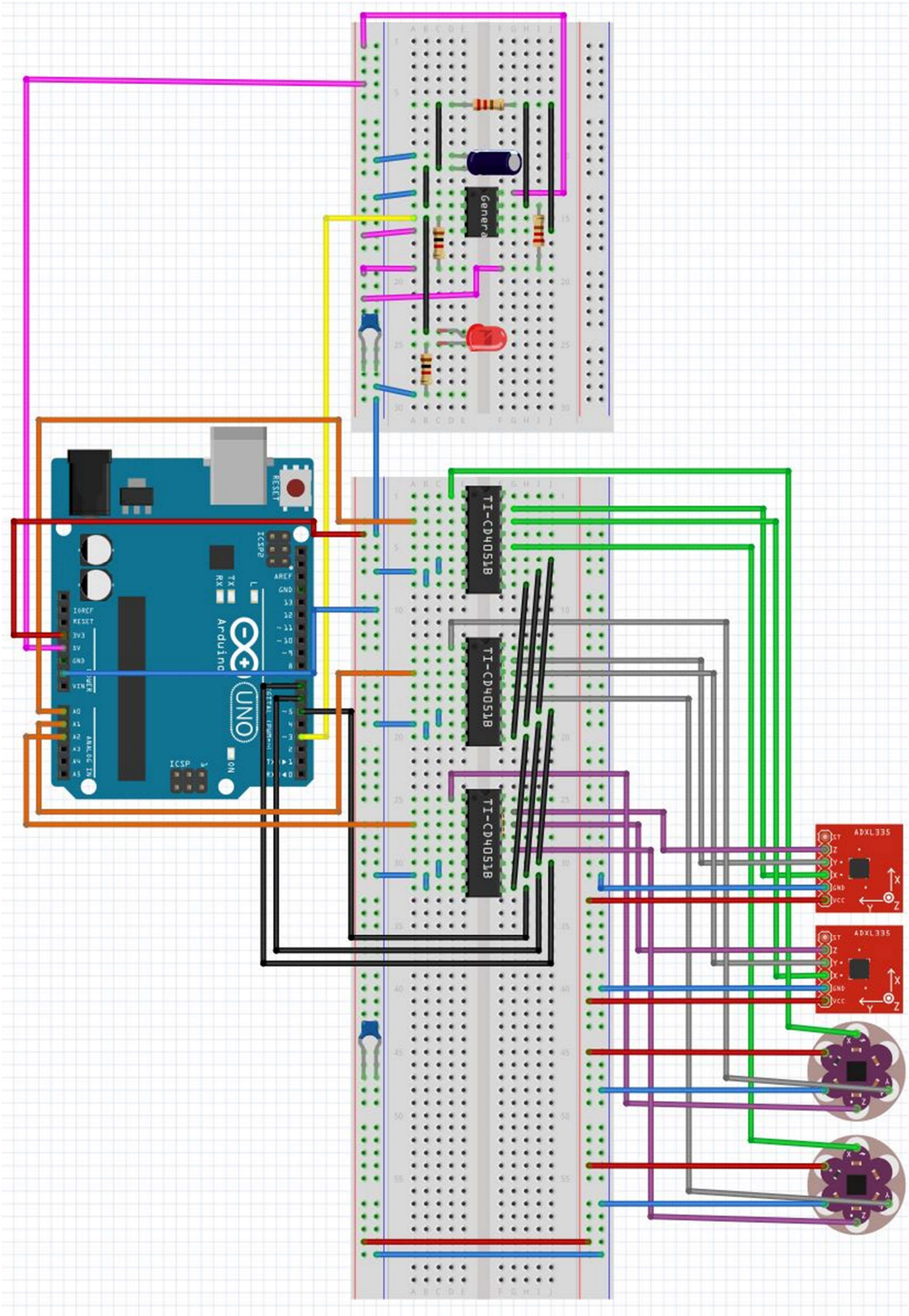


Ilustración 27. Esquema eléctrico de montaje desarrollado. Desarrollado con Fritzing.

4.2. Arduino

El software del Arduino está encargado de realizar tres funciones: Controlar el ciclo de bajo consumo del microprocesador, realizar las lecturas de los acelerómetros y enviar los datos por el puerto serie al ordenador donde se ejecuta Processing.

El circuito del temporizador NE555 genera una señal cuadrada con la descarga del condensador C cada, aproximadamente, 45ms. El flanco ascendente de esta señal cuadrada despierta, a través de la entrada digital 3, al Arduino del modo bajo consumo. Al reactivarse, este desactiva las interrupciones, termina la ISR (*Interrupt Service Routine*) y ejecuta el código cíclico. Tras esto, establece una comunicación asíncrona por el puerto serie y, empleando las salidas digitales 5 (C), 6 (B) y 7 (A), escoge el canal de los multiplexores para leer las aceleraciones de los ejes X, Y y Z del primer acelerómetro. Realiza un escalado de los valores leídos y los envía por el puerto serie. Acto seguido selecciona el siguiente acelerómetro y repite el proceso para todos los sensores. Tras terminar el envío del último sensor reactiva las interrupciones y entra en modo bajo consumo, a la espera de la siguiente interrupción hardware.

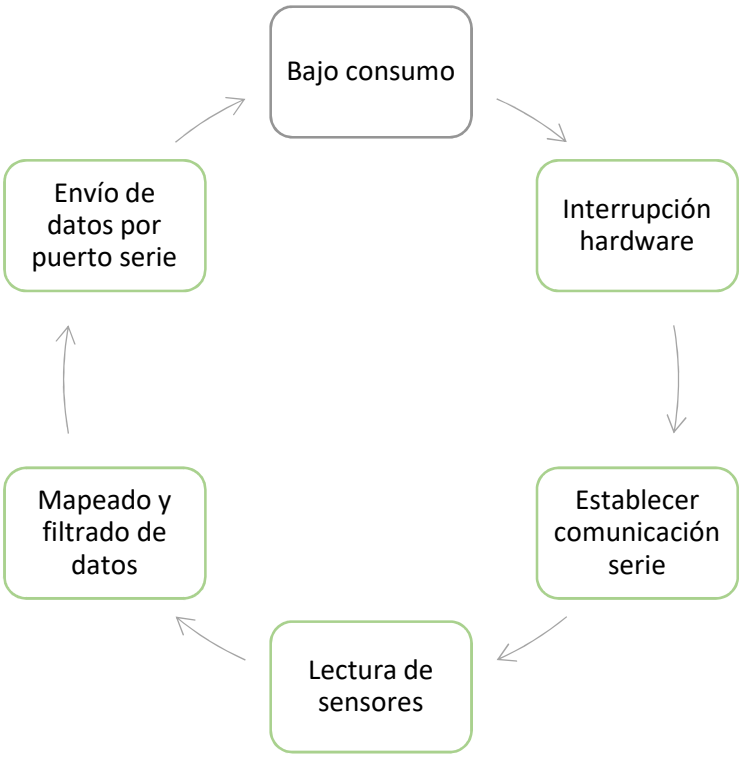


Ilustración 28. Ciclo de ejecución del programa de Arduino.

4.2.1. Sleep control – bajo consumo

Para controlar el modo de bajo consumo del microcontrolador es necesario disponer de un método para despertar al mismo. Por ello, la gestión de energía del microprocesador pasa tanto por el modo de bajo consumo como por el control de las interrupciones que lo reactivan.

Existen 5 modos de bajo consumo en el microcontrolador ATmega328P, cada uno con un nivel diferente de ahorro energético. El modo *Idle* es el método que menos ahorro energético aporta, interrumpiendo la ejecución del código de la CPU, pero manteniendo activos los SPI, USART, AC, 2-wire, temporizadores, *watchdog* e interrupciones. El modo *ADC Noise reduction* está enfocado a reducir el ruido a la hora de realizar lecturas analógicas y también detiene la ejecución de la CPU, pero mantiene activo el conversor ADC para realizar la lectura. El modo *Power-down* es el modo de mayor ahorro energético, dejando activos únicamente los watchdogs, el interfaz 2-wire y las interrupciones hardware. El modo *power-save* es similar al *power-down*, pero mantiene el temporizador 2 en marcha si se estuviese utilizando. El modo *Standby* es también similar al de *power-down*, sólo que mantiene los osciladores energizados.

Se puede realizar una reducción de consumo adicional deshabilitando los sistemas que no se vayan a utilizar, como el ADC, AC o watchdog.

Se escoge el modo *idle* por ser el único que permite realizar una interrupción por flanco ascendente/descendente y ser compatible con la tensión de alimentación del microprocesador. Se considera también oportuno habilitar y deshabilitar el ADC a lo largo de la ejecución del programa por el alto consumo que supone

Se emplea la librería <avr/sleep.h>, integrada de forma nativa en el IDE de Arduino, para gestionar el modo de bajo consumo. La librería *sleep* permite gestionar el modo de bajo consumo sin necesidad de acceder a los registros internos del ATmega328P. La función `sleep_cpu()` habilita el modo de bajo consumo de forma manual, siendo necesario realizar los reseteos de los flags correspondientes de los registros del microcontrolador con las instrucciones `sei()` y `cli()`. La función `sleep_mode()` gestiona los *flags*, no siendo necesario realizar más operaciones. (16)

Se hace uso también de la librería integrada <avr/power.h> para gestionar la energización de los sistemas del microprocesador. Esta librería permite habilitar y deshabilitar periféricos del microprocesador sin necesidad de acceder a los registros internos del ATmega328P. La función `power_adc_disable()` permite desactivar el ADC para ahorrar energía, y la función `power_adc_enable()` permite rehabilitarlo. (15)

Para gestionar la reactivación de la CPU se emplean las interrupciones nativas del lenguaje de Arduino. La función `attachInterrupt(pin, ISR, mode)` enlaza la entrada digital `pin` a una interrupción hardware, que ejecutará la función `ISR`. El parámetro `mode` indica cómo debe detectarse la interrupción, pudiendo ser CHANGE (la interrupción se activa cuando la entrada digital cambia de estado), LOW (la interrupción se activa cuando la entrada digital está a 0), RISING (la interrupción se activa con el flanco ascendente de la entrada digital) y FALLING (la interrupción se activa con el flanco descendente de la entrada digital). Los modos de bajo consumo que no son idle sólo gestionan las interrupciones por LOW. En las placas Arduino UNO sólo se pueden enlazar interrupciones a las entradas digitales 2 y 3. (17)

Cuando la CPU se encuentra atendiendo una interrupción se deben deshabilitar las interrupciones hasta que termine de procesarla para asegurar una correcta ejecución del código. De no ser así, una segunda interrupción podría interrumpir el servicio a la interrupción activa. El microcontrolador emplea las interrupciones internamente para una gran cantidad de tareas que deben ser evitadas dentro de una función ISR, tales como la comunicación serie (SPI, 2-wire…), contadores y temporizadores internos o *delays*. Si la interrupción va a acceder a datos de programa, estos deben ser declarados como *volatile*, para indicar al microcontrolador que pueden cambiar en cualquier momento y que debe recargar la variable en cada referencia a la misma.

4.2.2. Lectura, escalado y filtrado de aceleraciones

Para la lectura y escalado de los sensores se emplean las funciones nativas del lenguaje Arduino `readAnalogInput()` y `map()`.

Al comenzar la ejecución del código de lectura y escalado de sensores se realiza un *loop* que recorre todos los sensores. En cada iteración del *loop* se activan las salidas digitales de control de los multiplexores para habilitar las señales del acelerómetro deseado. La codificación de las señales de los multiplexores es la siguiente:

A	B	C	INH	SENSOR
0	0	0	0	Ch0, no empleado
1	0	0	0	Ch1, brazo izquierdo
0	1	0	0	Ch2, antebrazo izquierdo
1	1	0	0	Ch3, brazo derecho
0	0	1	0	Ch4, antebrazo derecho
1	0	1	0	Ch5, no empleado
0	1	1	0	Ch6, no empleado
1	1	1	0	Ch7, no empleado
X	X	X	1	Ningún canal

Para evitar calentamientos o descargas parciales del ADC al realizar varias lecturas analógicas de diferentes pines seguidas se realiza una doble lectura con la función `analogRead(pin)` para cada `pin`. La primera lectura se descarta y la segunda se mantiene para el procesado del dato.

Como los sensores están alimentados a $3{,}3\ V$ y las entradas analógicas del Arduino tienen un rango de $0 \div 5\ V$, es necesario descartar los valores superiores para asegurar un correcto escalado. El punto de equilibrio del sensor está aproximadamente en la mitad de la tensión de alimentación, lo que lo sitúa en $1{,}65\ V$. La sensibilidad del sensor es una décima parte de la tensión de alimentación, $0{,}33\ V/g$. Con un rango de medida de $\pm 3g$, el sensor oscila entre $0{,}66\ V$ y $2{,}64\ V$. Como el ADC tiene una resolución de 10 bits, una medida de $0 \div 5\ V$ se traduce en una lectura de 1023 puntos. El rango del sensor se traduce en una lectura que oscila entre $135\ y\ 540$ puntos.

Se quiere que los sensores nos devuelvan la aceleración en el sentido del eje, por lo que se invierten las señales resultantes. De esta forma cuando un eje apunte en la dirección de la gravedad, tendrá un valor positivo, y cuando apunte en contra, negativo.

Se emplea la función `map(value, fromLow, fromHigh, toLow, toHigh)` para escalar y controlar los *offsets* de las medidas. La función escala la variable `value` del rango inicial en puntos (`fromLow` y `fromHigh`), al rango final en unidades "g" (`toLow` y `toHigh`). Como la función `map` devuelve un valor entero, el rango de aceleración (en g) se multiplica por 100, a fin de tener dos posiciones decimales en el dato final.

Para eliminar ruidos y distorsiones de la medida se emplea un filtro de primer orden pasa-baja. El filtro realiza una ponderación entre la señal de salida del filtro del muestreo anterior y la nueva señal de entrada al filtro.

$$y_k = a \cdot y_{k-1} + (1 - a) \cdot x$$

La ecuación superior describe el funcionamiento del filtro siendo y_k el valor de salida del filtro, y_{k-1} el valor de salida del filtro en el muestreo anterior y x la nueva medida de entrada al filtro. El parámetro a es el que determina el nivel de actuación del filtro, modificando la ponderación del primer término y_{k-1} y del segundo término x. Cuanto más grande es *a*, más pesa el valor existente, por lo que la salida se ve menos afectada por los cambios presentes de la entrada (esto es, el filtro tiene una frecuencia de corte más baja). Por el contrario, un valor más pequeño de a hace que el filtro sea más sensible a las variaciones de la entrada, sin que estas supongan el total de la señal de salida.

Se han preprogramado cuatro valores de *a* para poder parametrizar la dureza del filtro en el código de Arduino. Para cada valor de *a* existe una constante de tiempo del filtro (el punto donde el nuevo valor supone el 63% del peso del resultado), que será proporcional al tiempo de ciclo de ejecución (T). Existe un tiempo de establecimiento (el punto donde el nuevo valor supone casi la totalidad del resultado, el 95%) que, en sistemas de primer orden, coincide con el triple de la constante de tiempo. El tiempo de establecimiento al 2% de error en lugar del 5% se sitúa en 4 veces la constante de tiempo en sistemas de primer orden.

a	Constante de tiempo (63%)	Tiempo de establecimiento (95%)
0	0	0
0,75	$4 \times T$	$12 \times T$
0,875	$8 \times T$	$24 \times T$
0,992	$128 \times T$	$384 \times T$

Se trabaja con un valor de *a* de 0,875, que se ha encontrado estable a la hora de realizar ensayos prácticos, pero el resto de los valores quedan operativos para posibles modificaciones.

4.2.3. Comunicación serie

Para la comunicación serie se emplea el puerto USART integrado en el Arduino, que está eléctricamente conectado a los puertos 0 (Rx) y 1 (Tx) del Arduino, y un bus USB para la transmisión de datos al ordenador.

Se establece una tasa de 115200 baudios para que la transmisión sea lo más rápida posible, a fin de poder finalizarla y devolver el microprocesador al estado de bajo consumo.

Se emplea la función `Serial.begin(baudRate)` para abrir el puerto serie a la velocidad de transferencia indicada, y la función `Serial.end()` para cerrarlo. Para evitar errores de ejecución, antes de enviar un mensaje por el puerto serie se comprueba que este esté abierto, tras lo que se realiza un envío secuencial de valores con la función `Serial.print(val)`.

La comunicación serie puede realizarse con valores ASCII. La función de escritura `Serial.print(val)` realiza una conversión del valor `val` a caracteres de código ASCII, por lo que cada carácter enviado tiene un peso de 1 *byte* en formato `char`. La función `Serial.println(val)` realiza la misma función, pero añade un retorno de carro (ASCII 13) y una nueva línea (ASCII 10) al finalizar la escritura.

A la hora de construir el mensaje se realiza un encapsulado con un identificador de inicio de trama y un identificador de fin de trama. Los valores a enviar se separan por comas, de modo que sean fácilmente identificables al llegar al destino. Los mensajes siguen el formato siguiente:

$$\#\#\#\#X_1, Y_1, Z_1, X_2, Y_2, Z_2, X_3, Y_3, Z_3, X_4, Y_4, Z_4 \%\%\%\%$$

Cada término (X, Y y Z) consiste en un número flotante con tres cifras significativas y un punto decimal. Si el valor es negativo, incluye también el símbolo del signo "-", por lo que cada medida analógica tiene un tamaño de $4 \div 5$ *bytes*. Cada trama realiza el envío de las doce mediciones de los sensores.

Haciendo la suma del tamaño de los datos vemos que el buffer enviado cada ciclo es de $67 \div 79$ bytes. Esto se empleará en el lado receptor del mensaje a la hora de identificar tramas válidas de datos.

4.3. Processing

El programa en Processing está encargado de realizar tres funciones: Recibir los datos del Arduino por el puerto serie, computar los datos recibidos y transformarlos en ángulos Pitch y Roll, y graficar en un entorno 3D los datos recogidos para animar un torso.

Processing se ejecuta en un ordenador conectado por cable USB al Arduino. Mantiene activo el puerto serie y cuando llega una nueva trama la analiza y extrae los datos relativos a los ejes de los acelerómetros. Realiza una conversión de las aceleraciones a ángulos de Roll y Pitch para los cuatro sensores a analizar. Tras esto dibuja los brazos y antebrazos en un *canvas* 3D, a lo que añade el dibujo del tronco y una cabeza estáticas, ya que no existen medidas de la orientación en los mismos. Se ha procurado respetar las proporciones anatómicas entre el cuerpo y los brazos.

Mientras tanto, el usuario puede interactuar con el *canvas* 3D, girándolo a voluntad o alrededor de un eje. Si resulta interesante analizar una postura en concreto, es posible pausar el s*treaming* de datos, congelando la animación, para permitir rotar el *canvas* sobre el cuerpo fijo. Estas interacciones se tratan como interrupciones al programa principal.

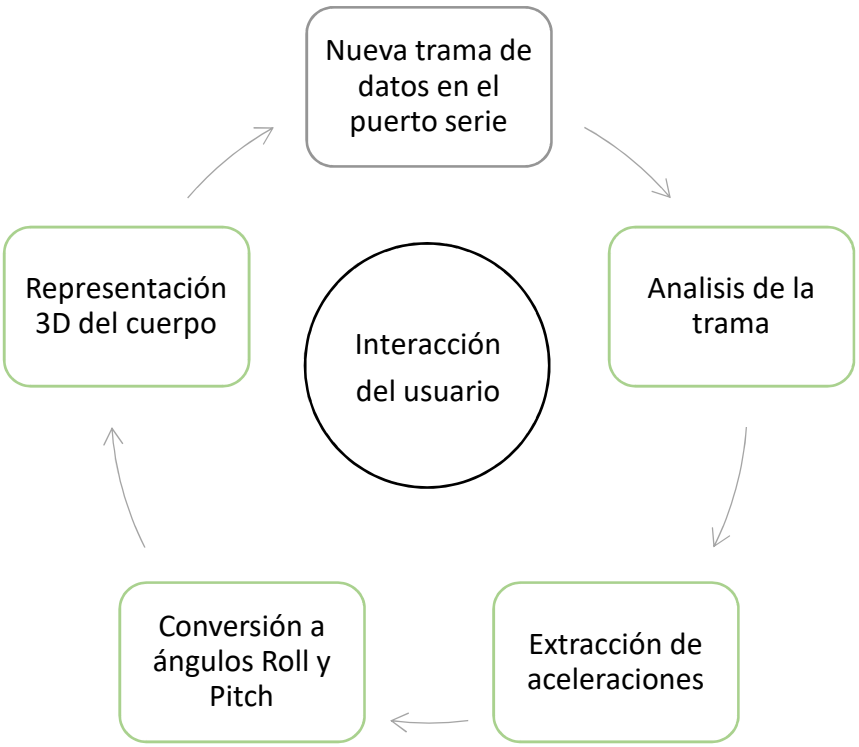


Ilustración 29. Ciclo de ejecución del programa de Processing.

4.3.1. Comunicación serie

Para gestionar la comunicación por el puerto serie se emplea la librería incluida en el IDE de Processing *Serial*. La librería permite realizar lecturas y escrituras del puerto serie, así como gestionar su apertura o cierre. Así mismo, permite diferenciar entre un envío de datos ASCII y un envío de bytes, permitiendo al usuario realizar la codificación de datos que desee a nivel de bit. Existe la posibilidad de asociar una interrupción cuando se detecta una trama específica en el puerto serie.

Para este proyecto se emplean el constructor `Serial(context, port, baudRate)` y las funciones, `available()`, `read()`, `clear()` y `stop()`.

Se genera una función para gestionar la apertura del puerto serie, que realiza, mediante un control `cath & try` 3 conexiones al puerto y a la velocidad configurados. Esta conexión se repetirá si no tiene éxito tras un lapso de 50 ciclos de ejecución. Se establece una tasa de transferencia de 115200 baudios, igual que en el Arduino.

Se genera una función para monitorizar el buffer del puerto serie y el estado de este. Si el puerto está abierto, en cuanto se detecta, mediante la función `available()`, que el tamaño del buffer es mayor al tamaño mínimo de la trama (67) se realiza una lectura de los bytes del buffer. Se convierten en un único *string* que contenga la totalidad de la trama leída y se realiza un truncado del mismo buscando el identificador de inicio de trama "####". Se conservan los datos de la trama y se realiza un nuevo truncado buscando el identificador de fin de trama "%%%%". Ahora que se ha eliminado el encapsulado del mensaje, se realiza un truncado para separar por comas el *string* obtenido. Si existen 12 valores separados por coma, se revisa que tras el identificador de fin de trama se encuentre información sobre los tiempos de ciclo del Arduino y si existen, se da por bueno el mensaje, pasando al procesado de los datos obtenidos. A lo largo del procesado del algoritmo se va revisando la calidad de los datos obtenidos en el paso anterior para descartar la trama si los tamaños no encajan o si no se detectan identificadores de inicio/fin de trama. En tal caso, la trama se descarta y se borra el buffer del puerto con la función `clear()`.

El algoritmo analiza también posibles fallos de comunicación. Si el programa pasa más de 10 ciclos de ejecución sin refrescar los datos (cuya frecuencia de refresco oscila entre 1 y 3 ciclos), marca el puerto como cerrado e intenta realizar una reconexión cada 50 ciclos de ejecución.

4.3.2. Tratamiento de los datos

Para el tratamiento de los datos hay que hacer una conversión de los valores de aceleraciones de los tres ejes a ángulos de giro. Para representar la posición de los sensores (en forma de brazos y antebrazos) es necesario asumir una situación de estabilidad donde no existen aceleraciones lineales, ya que la única referencia existente en el sistema es la gravedad. El emplear filtros para la lectura de las aceleraciones ayuda a eliminar ruido que desestabilizaría el cálculo de posición.

Se define la posición de reposo de los sensores con el eje Z orientado hacia el cielo, paralelo al vector de la gravedad. El cuerpo en reposo tendrá los brazos (y antebrazos) formando 90º con la columna vertebral y perpendiculares al pecho. Este se considera el punto donde tanto el ángulo de Roll como el ángulo de Pitch son iguales a cero. Las medidas analógicas se han tratado de tal forma que indiquen 1g cuando el eje está alineado y apuntando en el mismo sentido que la gravedad, por lo que en la situación de reposo de los sensores la lectura del eje z recibida es de -1g, mientras que las lecturas de los otros dos ejes son iguales a cero.

Para extraer los ángulos Roll y Pitch se emplean las siguientes ecuaciones con las aceleraciones de los ejes X (a_x), Y (a_y) y Z (a_z):

$$Roll\ (\phi) = \tan^{-1} \frac{a_y}{sign(a_z) \cdot \sqrt{{a_z}^2 + \mu \cdot {a_x}^2}}$$

$$Pitch\ (\theta) = \tan^{-1} \frac{-a_x}{\sqrt{{a_y}^2 + {a_z}^2}}$$

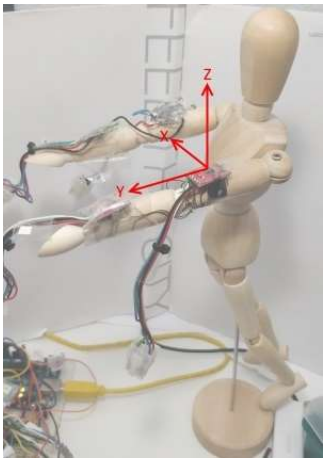


Ilustración 31. Sensor rotado 180º sobre su eje Y de medición desde su posición de reposo.

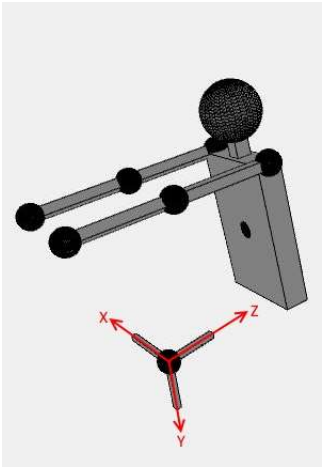


Ilustración 30. Ejes de referencia en Processing.

4.3.3. Representación 3D de los datos

Una vez obtenidos los ángulos de Roll y Pitch se procede a dibujar en el *canvas* 3D el torso humano. Se van a emplear las funciones `pushMatrix()`, `popMatrix()`, `translate()`, `rotateX()`, `rotateY()`, `box()` y `sphere()`. Se emplean las ilustraciones siguientes para explicarlas. (18)

A la hora de dibujar gráficos en Processing hay que indicar las coordenadas de origen de la forma a representar. Para dibujar el cuadrado gris se emplea la instrucción `rect(20, 20, 40, 40)` que dibuja un rectángulo con origen en las coordenadas X=20, Y=20, un ancho de 40 puntos y un largo de 40 puntos. Para dibujar el cuadrado negro emplea la misma instrucción desplazando el punto de origen del dibujo a la nueva posición con la instrucción `rect(20 + 60, 20 + 80, 40, 40)`. Este método obliga a llevar un control sobre las posiciones absolutas de cada punto de origen de dibujo.

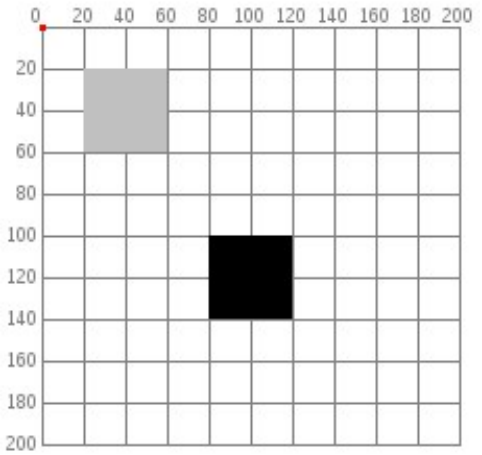


Ilustración 32. Representación de dos cuadrados. Fuente processing.org.

La instrucción `translate()` permite mover el punto de origen de posición (0,0) del *canvas*, En este caso, tras dibujar el primer cuadrado se ejecuta la instrucción `translate(60,80)`, lo que mueve el *canvas* sin mover lo que está previamente dibujado sobre él. Ahora el segundo cuadrado se dibuja con la instrucción `rect(20,20,40,40)`, ya que el nuevo punto escogido es la nueva referencia (0,0) del *canvas*. La translación es una instrucción incremental, por lo que si se ejecutase la instrucción `translate(40, 20)` después de dibujar el segundo cuadrado, se movería el *canvas* al punto 100,100 según el *canvas* original.

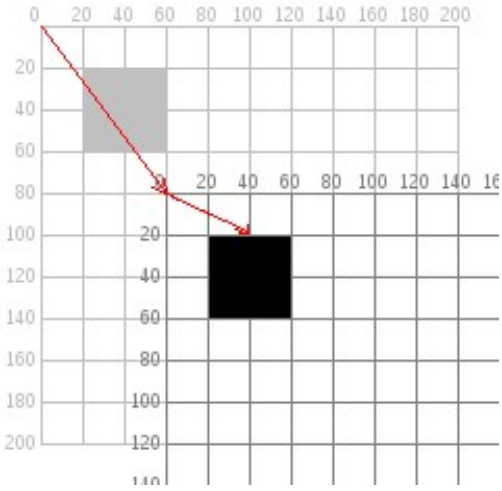


Ilustración 33. Ejemplo de una translación del canvas. Fuente processing.org.

La instrucción `rotate(α)` realiza una rotación de los ejes de coordenadas X e Y en sentido horario. En este caso, tras dibujar el primer cuadrado se ejecuta la instrucción `rotate(radians(45))`, lo que rota los ejes del *canvas* sin alterar lo que está previamente dibujado. Ahora el segundo cuadrado se dibuja con la instrucción `rect(20, 20, 40, 40)`, pues los nuevos ejes están rotados respecto a los originales. En este caso se observa que el segundo cuadrado se sale del *canvas*. Al igual que con la instrucción `translate()`, la instrucción `rotate(α)` es acumulativa.

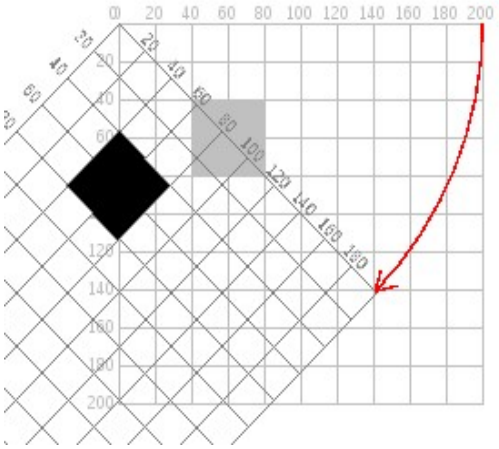


Ilustración 34. Ejemplo de una rotación de ejes. Fuente processing.org.

Ambas instrucciones `translate(X,Y)` y `rotate(α)` pueden combinarse para realizar un desplazamiento y una rotación de los ejes. Se ve cómo en este caso, tras dibujar el primer cuadrado, se ejecuta la instrucción (A) `translate(40, 40)` la instrucción (B) `rotate(radians(45))` y por último se dibuja el segundo cuadrado (C) `rect(0, 0, 40, 40)`. Al ser ambas instrucciones incrementales, el orden de ejecución de las instrucciones será determinante para el resultado.

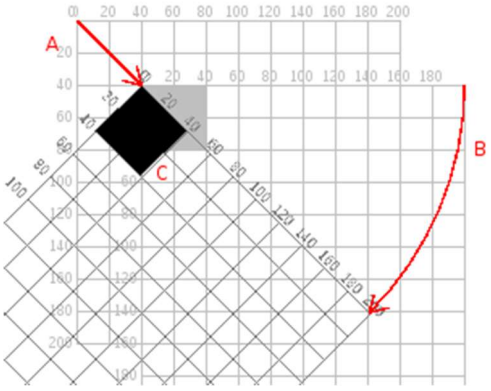


Ilustración 35. Ejemplo de una translación y una rotación de ejes. Fuente processing.org.

El problema asociado a estas operaciones es que modifican de manera indefinida la orientación de los ejes y la posición del *canvas*, información que se guarda en la matriz de transformación. Existe una forma de realizar cambios temporales en el entorno gráfico empleando una pila para las matrices de transformación. Las instrucciones `pushMatrix()` y `popMatrix()` permiten modificar el *canvas* para realizar dibujos y luego restaurarlo a la situación anterior. Processing realiza un reset de la matriz de transformación con cada ejecución del ciclo `draw`().

A la hora de realizar dibujos en 3D en vez de en 2D existen cambios a la hora de realizar las operaciones, pero el concepto general del *canvas* es el mismo.

Las funciones de dibujo bidimensional `rect()` y `ellipse()` se sustituyen por las funciones `box()` y `sphere()`. En lugar de introducir las coordenadas de origen del dibujo, los dibujos se generan sobre su centro de gravedad, indicando la anchura, altura y profundidad en el caso del cubo, y el radio en el caso de la esfera. (19)

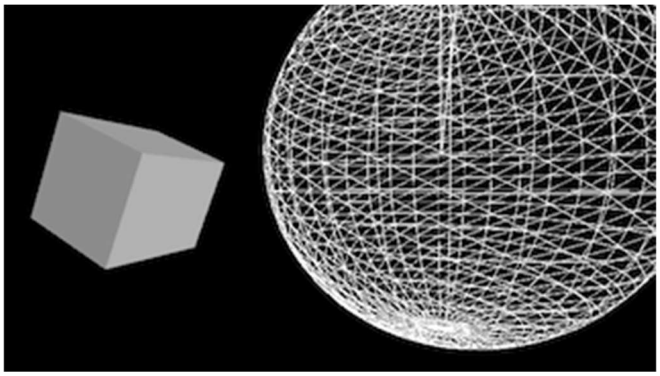


Ilustración 36. Representación de una esfera y un cubo en un canvas 3D. fuente processing.org.

El punto de origen del *canvas* pasa a ser el punto sobre el que se centrará la figura tridimensional a dibujar, por lo que será necesario jugar con la translación del *canvas* y la rotación de los ejes constantemente.

La instrucción `translate()` emplea ahora un tercer parámetro para la tercera dimensión (profundidad). La instrucción `rotate()` sin embargo se convierte en tres instrucciones de rotación independientes, `rotateX(α)`, `rotateY(α)` y `rotateZ(α)`. Estas funciones siguen siendo incrementales en el *canvas* 3D.

Existe una diferencia en el *canvas* 3D que es la existencia de las cámaras. La cámara permite rotar la visualización y realizar zoom, sin alterar con ello los ejes de referencia del *canvas*. Es la herramienta para contemplar las diferentes perspectivas que nos da la tercera dimensión del dibujo. En este proyecto se emplea la librería *peasyCam* para gestionar la cámara con el ratón.

Para la representación gráfica de los sensores hay que tener en cuenta la posición de los ejes de referencia de Processing frente al os ejes de referencia del sensor en situación de reposo. Interesa especialmente que los ejes estén alineados, no tanto el sentido del eje, pues esto afectará sólo al signo del ángulo que hay que rotar. Se realizan los cálculos de ángulos giro para el sistema de referencia del sensor y después se adecúa a los ejes de Processing.

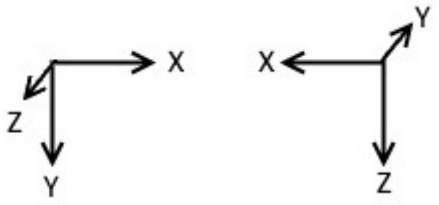


Ilustración 37. Ejes de referencia en Processing (derecha) y en el sensor en posición de reposo (izquierda).

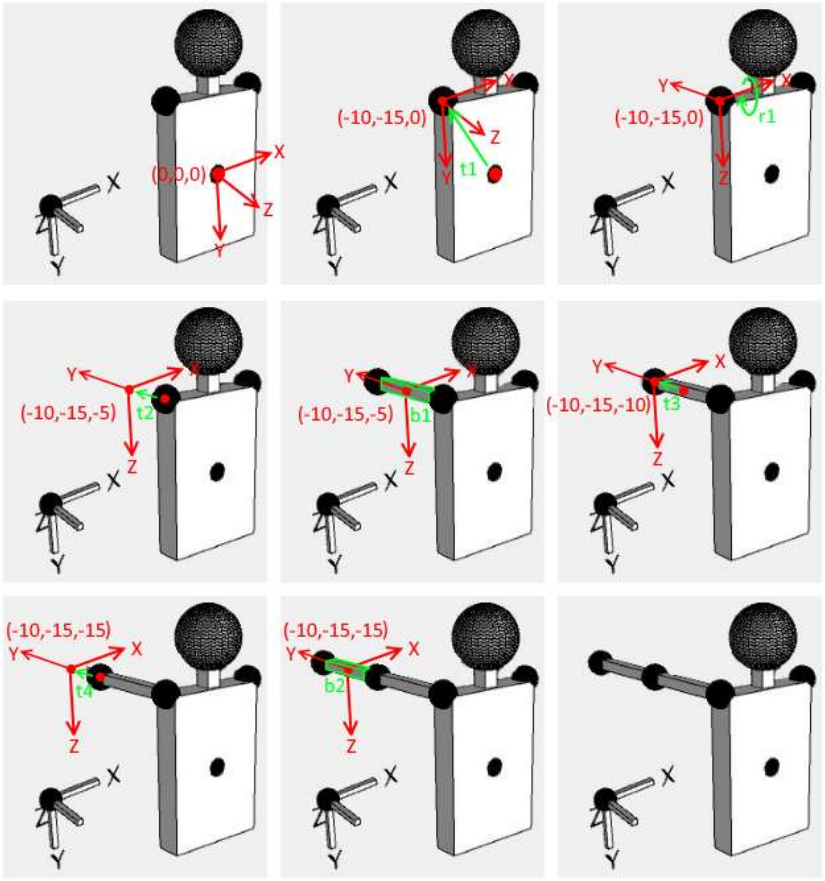


Ilustración 38. Proceso de dibujo del brazo izquierdo en estado de reposo en Processing. El color rojo indica los orígenes de dibujo y sus ejes, y el color verde representa las operaciones realizadas.

La representación simplificada del estado de reposo (Roll = 0º y Pitch = 0º) se genera con un encadenamiento de rotaciones y translaciones. Se emplea la instrucción de `pushMatrix()` para modificar temporalmente la matriz de rotaciones de Processing. La translación *t1* mueve el origen de coordenadas del punto inicial del *canvas* (0,0,0) al hombro (-10,-15,0). Se realiza la rotación *r1* de 90º ($\frac{\pi}{2}$) sobre el eje X que orienta a los ejes Y y Z de igual que en los ejes del sensor en situación de reposo. En este punto se aplicarían los giros sobre el eje X (Roll) y sobre el eje Y (Pitch) con las funciones `rotateX()` y `rotateY()`, pero al ser esta la situación de reposo, los ángulos de giro tienen un valor de cero y se ha omitido la operación. Tras orientar los ejes se avanza la mitad del tamaño del brazo a representar (-10, -15, -5) con la traslación *t2*, ya que las funciones de dibujo 3D realizan un dibujo sobre el punto central del cuerpo. Sobre este punto se dibuja como una caja el brazo izquierdo con b1. Se realiza la translación *t3* de la mitad del tamaño del brazo a dibujar el eje Y para llegar al codo (-10, -15, -10). Aquí se repite el proceso del hombro, reorientando los ejes a las medidas del sensor del antebrazo, desplazando con *t4* el punto de origen al centro del brazo y dibujándolo con *b2*. Con el brazo representado se deshacen los cambios de la matriz de rotación con la función `popMatrix()` y se realiza el mismo proceso con el brazo derecho.

5. Análisis del trabajo realizado

Para validar el proyecto desarrollado se realizan una serie de análisis que se exponen a continuación.

5.1. Temporizador e interrupción hardware

El primer análisis busca obtener la frecuencia de las interrupciones hardware. Se elimina el código del Arduino y se mantiene la llamada únicamente a la interrupción hardware con la señal de flanco del temporizador. Se mantiene el puerto serie constantemente abierto a 115200 baudios para evitar retrasos en la conexión y desconexión de este. Con cada flanco ascendente detectado por el puerto se realiza una identificación con un contador de la señal, se guarda el *timestamp* y se envía por el puerto serie. Se analiza el comportamiento de 4000 muestras.

Resultados	
Media número de ciclos por trama	1,0000
Media entre pulsos (µs)	45946
Desviación estándar entre pulsos (µs)	457

De los resultados obtenidos se deduce que no se detectan fallos de comunicación en un entorno controlado, pues no existe ningún flanco detectado que no haya sido enviado por el puerto serie. Se comprueba que el tiempo medio entre pulsos es de 45,946 ms, con una desviación estándar de 457 µs (1%). El tiempo teórico esperado es de 45,73 ms, por lo que se aceptan como válidos los cálculos del circuito realizados.

5.2. Ejecución de código

El siguiente análisis busca conocer si es necesario realizar ciclos de bajo consumo. Para ello se determina el tiempo de ejecución del código, mantiene el puerto serie constantemente a 115200 baudios abierto a fin de no tener fallos en la apertura de este. Se realiza un *timestamp* al iniciar la ejecución del código con la interrupción hardware. Tras realizar las lecturas analógicas y enviarlas por el puerto serie se envía un nuevo *timestamp* para conocer cuándo ha terminado la ejecución del programa principal. Se estima que la lectura de las señales analógicas será lo que más tiempo consuma.

Resultados	
Media número de ciclos por trama	1,0000
Media entre ciclos start (µs)	46855
Desviación estándar de tiempo de ciclo start	2808
Media entre ciclos end (µs)	46855
Desviación estándar de tiempo de ciclo end	2808
Media de tiempo de ejecución de código (µs)	9139
Desviación estándar de tiempo de ejecución de código	68
Media tiempo de ciclo ejecutando código (%)	19,57%
Media tiempo de espera (%)	80,43%

La media de tiempo para ejecutar el código está en 9,1ms. Frente a la frecuencia de las interrupciones, este supone un 19,5% del tiempo, lo que implica que el 80,5% del tiempo el procesador se encuentra esperando a una nueva interrupción. El consumo del microprocesador en modo Idle (el que se emplea en este proyecto) es 4 veces inferior al consumo activo del microprocesador. Si se emplea un modo más restrictivo, este consumo puede llegar a una milésima parte del consumo activo del procesador.

Symbol	Parameter	Condition		Min.	Typ.(2)	Max.	Units
I_{CC}	Power Supply Current(1)	Active 1MHz, V_{CC} = 2V	T = 85°C		0.3	0.5	mA
		Active 4MHz, V_{CC} = 3V	T = 85°C		1.7	3.5	
		Active 8MHz, V_{CC} = 5V	T = 85°C		5.2	12	
		Idle 1MHz, V_{CC} = 2V	T = 85°C		0.04	0.5	
		Idle 4MHz, V_{CC} = 3V	T = 85°C		0.3	1.5	
		Idle 8MHz, V_{CC} = 5V	T = 85°C		1.2	5.5	
	Power-save mode(3)	32kHz TOSC enabled, V_{CC} = 1.8V	T = 85°C		0.8		µA
		32kHz TOSC enabled, V_{CC} = 3V	T = 85°C		0.9		
	Power-down mode(3)	WDT enabled, V_{CC} = 3V	T = 85°C		4.2	15	
		WDT disabled, V_{CC} = 3V	T = 85°C		0.1	2	

Ilustración 39. Consumos del chip ATmega328P. Fuente datasheet.

5.3. Ejecución de código con gestión del puerto serie

Este análisis pretende medir el impacto de abrir y cerrar el puerto serie antes y después de las comunicaciones. Se realiza el mismo ensayo que en el punto anterior, pero esta vez se gestiona la apertura del puerto después de detectar el flanco y grabar el *timestamp*. Tras realizar el último envío (*el timestamp*) el puerto se cierra.

Resultados	
Media número de ciclos por trama	1,0000
Media entre ciclos start (µs)	46022
Desviación estándar de tiempo de ciclo start	216
Media entre ciclos end (µs)	46022
Desviación estándar de tiempo de ciclo end	217
Media de tiempo de ejecución de código (µs)	9042
Desviación estándar de tiempo de ejecución de código	53
Media tiempo de ciclo ejecutando código (%)	19,65%
Media tiempo de espera (%)	80,35%

Aunque en esta medición el tiempo de ejecución ha caído respecto a la medición anterior, también lo ha hecho la frecuencia de la interrupción. Como sólo se tiene en cuenta el tiempo de apertura del puerto serie y no el cierre, el cambio de tiempo de ejecución se podrá doblar para estimar el incremento computacional. Así pasamos de un 19,57% del tiempo en ejecución a un 19,73% (interpolado ya).

Queda justificado el uso del modo de bajo consumo con estos datos. Como se puede observar en la figura anterior, los modos de bajo consumo *power-save* y *power-down* hacen uso de tensiones reducidas de alimentación del microprocesador ATmega328P, por lo que no sería posible emplearlos en la placa Arduino empleada (que alimenta el microprocesador con 5V y emplea un oscilador externo de 16MHz). Otras placas, como el Arduino mini, emplean un voltaje de 3,3V para alimentar al microprocesador, lo que le permite trabajar con el reloj de 8MHz y reducir su consumo, tanto en modo activo, como en los modos de bajo consumo.

5.4. Temporización de interrupción en bajo consumo

Este análisis busca medir el impacto del modo bajo consumo en el tiempo de respuesta a las interrupciones y a la comunicación serie. Se realiza el mismo análisis que en el punto 5.1 para detectar la frecuencia, sólo que esta vez se habilita el modo bajo consumo entre interrupción e interrupción y se realiza una gestión de apertura/cierre del puerto serie. Como se emplea el modo *idle* como modo de bajo consumo, y este mantiene energizados los temporizadores, se puede seguir empleando la instrucción `micros()` del microprocesador para realizar un *timestamp* con la información interna de Arduino.

Resultados	
Media número de ciclos por trama	1,0000
Media tiempo de ciclo idle (µs)	45535
Desviación estándar idle	463

Se observa que el empleo del modo de bajo consumo no hace que se dejen de detectar interrupciones, ni que se pierdan tramas con la apertura y cierre del puerto.

Se realiza el mismo análisis manteniendo todo el código de Arduino, realizando una medición con el código final desarrollado (cuyo tiempo de ejecución ha sido calculado en los puntos 5.2 y 5.3).

Resultados	
Media número de ciclos por trama	1,0000
Media entre ciclos start (ms)	45075
Desviación estándar de tiempo de ciclo start	2407

Nuevamente los tiempos de trabajo no varían de forma notable, por lo que se da por válido el algoritmo de gestión de energía desarrollado para el proyecto.

5.5. Cálculos de orientación

Este análisis busca validar los cálculos numéricos realizados, así como la correcta interpretación de los ejes de referencia de los sensores frente a la referencia de reposo asumida.

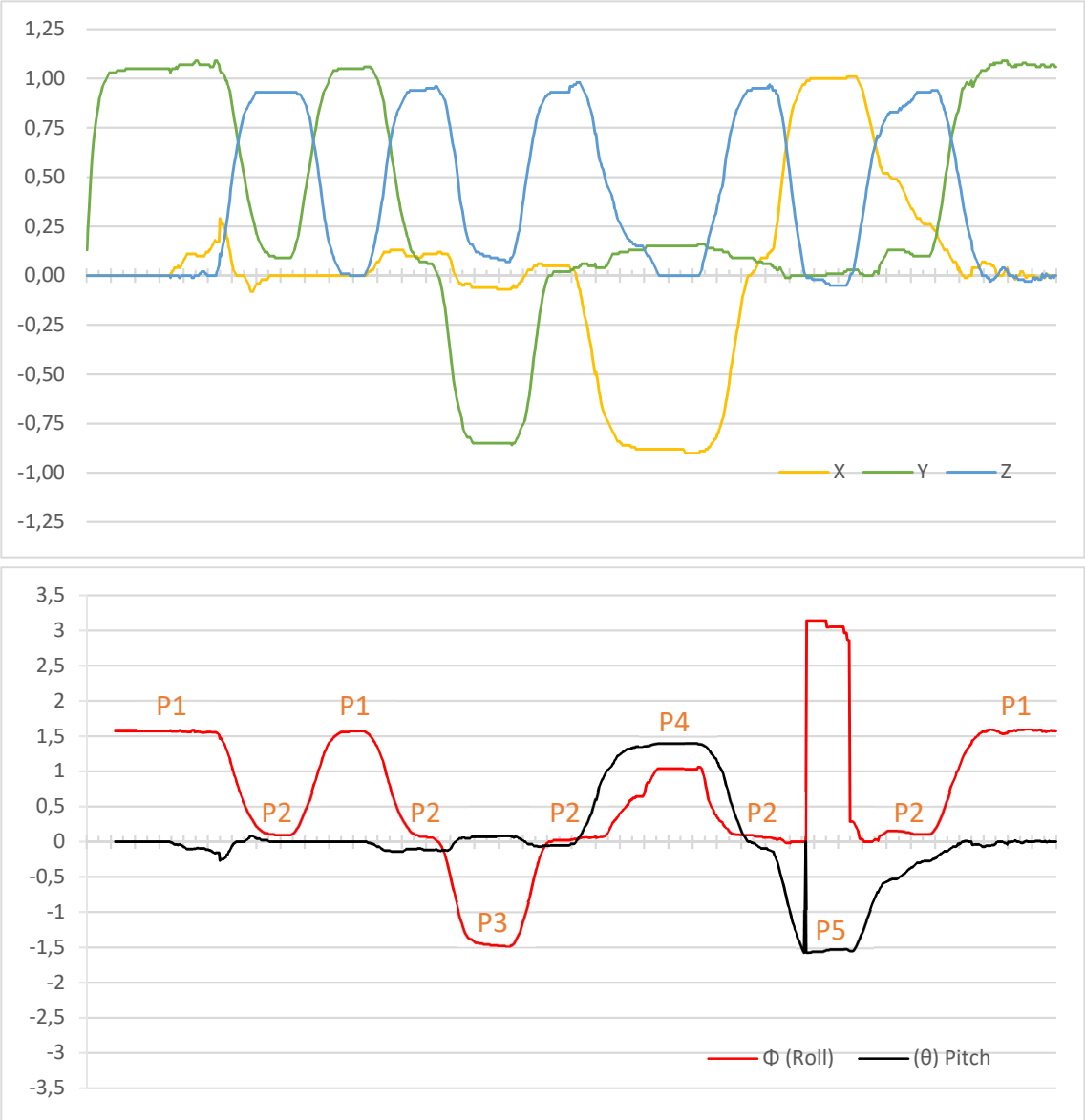


Ilustración 40. Representación de las medidas del acelerómetro 1 leídas (superior) y de los ángulos de Roll y Pitch calculados con las matrices de rotación. Se indica en naranja la posición del maniquí de la figura 41 correspondiente.

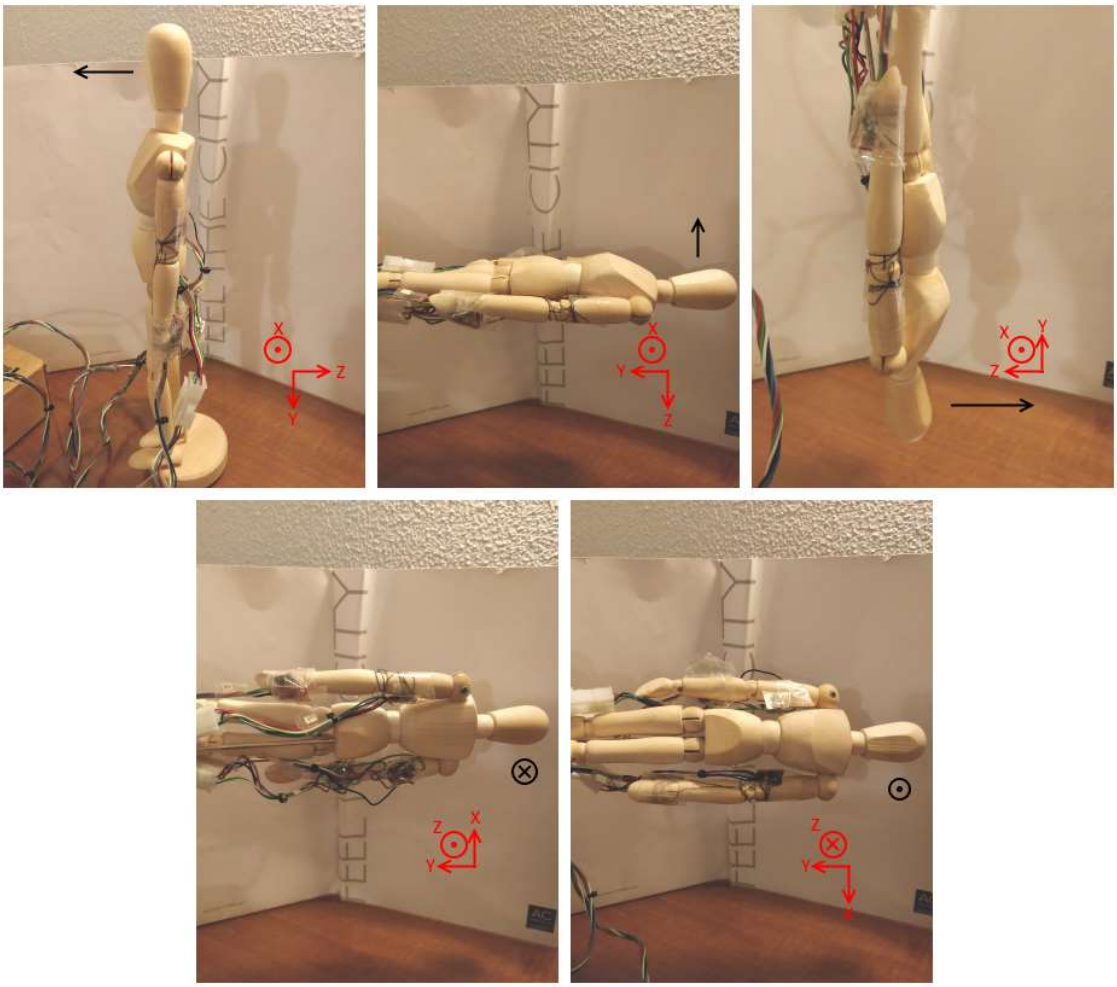


Ilustración 41. Posiciones analizadas en el análisis. De izquierda a derecha y de arriba abajo en el orden de lectura del texto están las posiciones p1, p2, p3, p4 y p5. Se representan en negro la orientación de la parte frontal del maniquí para cada rotación. Se representa en rojo la orientación de los ejes de los sensores para cada rotación.

Se espera al ciclo 30 para asegurar una estabilización de los valores para realizar el cálculo de Roll y Pitch. Se indican en el gráfico de los ángulos Roll y Pitch las posiciones analizadas correspondientes a cada estado de la Ilustración 41.

Para el análisis se pliegan los brazos del maniquí y se posiciona este en vertical como se muestran en la posición *p1*, quedando los ejes del sensor rotados 90º de la posición de reposo del sensor. Se aplica un giro al maniquí hasta la posición *p2*, quedando los ejes de los sensores en la posición de reposo. Se devuelve el maniquí a la posición *p1*, para después volver a rotar a *p2*. Se hace otra rotación en el mismo sentido para llegar a la posición *p3* y después se vuelve a la posición *p2*. Desde ahí se rota sobre otro eje el maniquí hasta la posición *p4*. Después se realiza un giro en sentido contrario, pasando por la posición *p2*, hasta llegar a la posición *p5*. Tras esto se vuelve a la posición *p2* y se termina en la posición *p1*.

Se puede observar que en la posición *p1* el eje Y (el longitudinal del brazo) está apuntando hacia el suelo (+1g) y que el ángulo de Roll calculado es de unos $\frac{\pi}{2}$. En cuanto rotamos el maniquí a la posición *p2* el eje Z pasa a estar orientado hacia el suelo (+1g) y el ángulo de Roll calculado es cero. Si se rota el maniquí a la posición *p3* se puede observar que el eje Y apunta hacia el cielo (-1g) y que el ángulo de Roll es ahora de $-\frac{\pi}{2}$. Se observan rotaciones de Pitch de pequeña magnitud, posiblemente debidas a la alineación del sensor con el brazo y a haber realizado las rotaciones del maniquí manualmente.

Cuando se terminan los movimientos de roll se analizan los movimientos de pitch. Se observa que el cálculo del Pitch es correcto para las posiciones *p4* y *p5*. En la posición *p4* la gravedad en X vale -1g y el valor de pitch es de $\frac{\pi}{2}$. En la posición *p5* se observa que la gravedad en X vale +1g y que el ángulo de pitch calculado es de -$\frac{\pi}{2}$. Sin embargo se puede observar que el cálculo del Roll no es coherente cerca del giro de $\pm\frac{\pi}{2}$. Se aprecia también una discontinuidad en el cálculo del Roll en el paso de la posición *p2* a la *p5*. Si analizamos los valores de las aceleraciones se aprecia que las aceleraciones de X e Y están cerca de cero. Este error es intencionado, pues corresponde a la aproximación empleada para el ángulo de Roll. Sin esta aproximación, el resultado del cálculo de Roll sería una indeterminación del tipo $\frac{0}{0}$ y no tendría solución para ese punto.

5.6. Representación de orientación

Este último análisis busca medir la fidelidad de representación resultante de las matrices de rotación empleadas en el ordenador sobre Processing. Para ello se ha realizado la conexión serie entre el Arduino y el ordenador, dejando que los softwares se ejecuten libremente mientras se posiciona al maniquí en diferentes posturas.

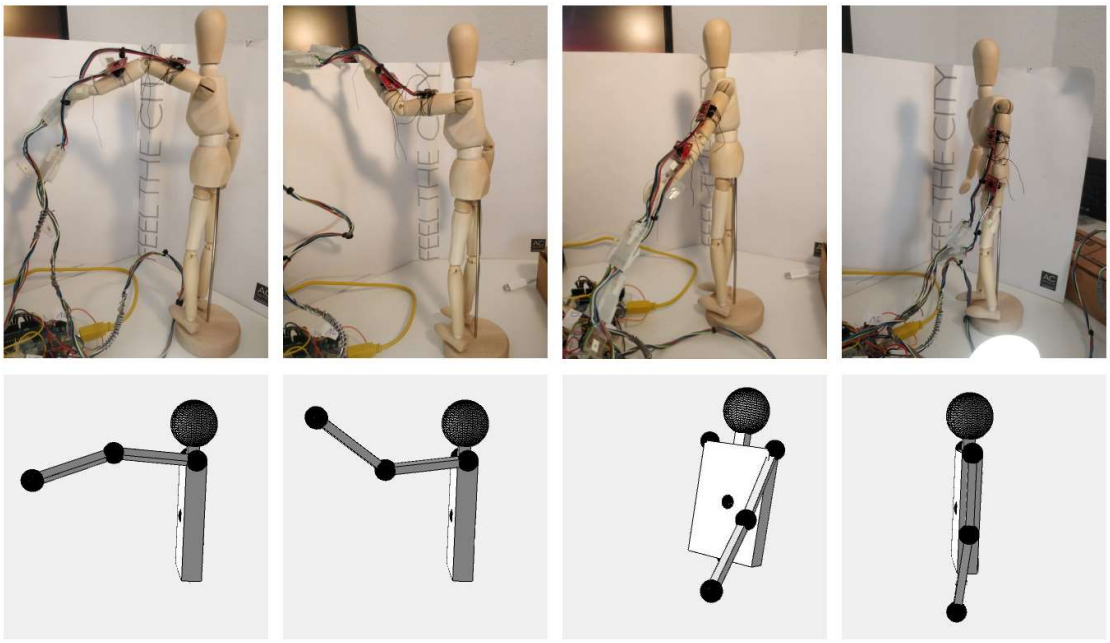


Ilustración 42. Representación del movimiento del brazo izquierdo capturado por dos sensores.

Primero se ha analizado la respuesta del sistema con dos sensores. Se realizan diferentes movimientos del brazo y del antebrazo para comprobar la correcta parametrización del sistema.

Los resultados son satisfactorios para la representación del ángulo de Roll, así como en la mayoría de los ángulos de Pitch. Se comprueba que situando el sensor cerca de la orientación en la que el eje X se encuentra paralelo a la gravedad (y las aceleraciones Y y Z son cercanas a cero) aparecen errores de medida del ángulo de Roll. Este comportamiento se ha previsto en el análisis anterior y se conoce como *gimball lock*, donde la representación de ángulos de Euler pierde un grado de libertad. Una solución a este problema es el empleo de cuaterniones para la definición de la orientación.

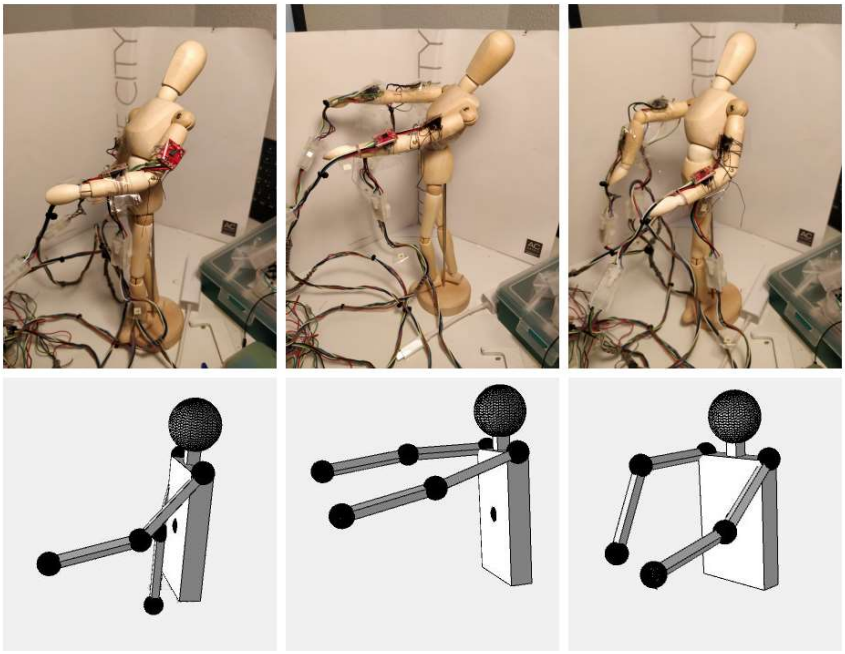


Ilustración 43. Representación del movimiento de los dos brazos capturados por cuatro sensores.

Añadir un segundo brazo, con su juego de dos sensores, devuelve las mismas conclusiones que el ensayo anterior. Ahora es posible graficar la posición de ambos brazos simultáneamente y orientar cada uno de los cuatro sensores en una posición diferentes para ver, en tiempo real, la replicación de la orientación.

Se aprecia que, aunque exista una zona de indeterminación donde la representación no es correcta, al salir de la zona de conflicto la representación se restaura, no existiendo integraciones del error ni fallos permanentes.

6. Conclusiones

Se ha desarrollado un sistema de captura de movimiento empleando acelerómetros como única fuente de información externa. Se consigue capturar y representar los ángulos de Roll (giro sobre el eje X) y Pitch (giro sobre el eje Y). Sin embargo, el material disponible implica limitaciones físicas de reproducción de movimiento, no siendo posible representar el ángulo de Yaw (giro sobre el eje de la gravedad) de los sensores.

El empleo de acelerómetros para el cálculo de orientación implica también la existencia de una zona de indeterminación matemática, conocida como *gimball lock*, ante un ángulo elevado de pitch. Este se da cuando el eje X se encuentra paralelo a la gravedad (independientemente de su sentido). Este error no se acumula en el tiempo.

Estas dos características pueden ser limitantes para algunas aplicaciones, pero la simplicidad matemática y computacional de la captura de movimiento con acelerómetros la convierte en una herramienta atractiva para diversas aplicaciones.

En el caso de la captura de movimiento humano, el uso de acelerómetros se plantea una solución parcial. Al ser los movimientos de las articulaciones humanas circular, perder el ángulo de la orientación del giro Yaw puede suponer una pérdida grande de información si se quiere capturar todo el rango de movimientos. Si los movimientos se limitan a la flexión/extensión del brazo (movimientos hacia adelante y hacia atrás) o movimientos de rotación (giro del antebrazo sobre su eje) el método supone una solución aceptable.

Otras posibles aplicaciones de la captura de movimiento con acelerómetros pueden ser en control de orientación de pantallas en dispositivos móviles, el cálculo de pendientes y desniveles en una ruta, aplicaciones de realidad virtual, niveles digitales…

Si se requiere realizar una captura de movimiento precisa es necesario realizar una integración de más sensores. La solución óptima pasa por emplear un IMU (acelerómetro + giroscopio) que integre un magnetómetro de tres ejes para conseguir un mayor número de grados de libertad. El procesado de estos sensores es más complejo que el del proyecto desarrollado (matrices de cosenos directores o filtros de Kalman como algoritmos de fusión sensorial), tanto conceptual como computacionalmente y requiere de procesadores más potentes (y por tanto más caros) que los empleados en este proyecto. Por esto, para proyectos en entornos controlados a los que no afecten las limitaciones descritas, la solución empleada en este proyecto puede ser más atractiva, sencilla y económica.

7. Bibliografía

1. D., Roetenberg. *Inertial and magnetic sensing of human motion.* [En línea] 2006. https://research.utwente.nl/files/6041330/thesis_Roetenberg.pdf.

2. *Human-Scale Motion Capture with an Accelerometer-Based Gaming Controller.* Purkayastha S.N, Bryne M.D, O'Malley M.K. Vol.25, pp.458-465, s.l. : JRM, 2013, Vol. Journal of Robotics and Mechatronics. doi: 10.20965/jrm.2013.p0458.

3. F., Guillou D. Sensorsmag. *New Manufacturing Methodology Substantially Reduces Smart MEMS Costs.* [En línea] [Citado el: 31 de agosto de 2018.] http://archives.sensorsmag.com/articles/1203/20/main.shtml.

4. J., Bernstein. Sensorsmag. *An overview of MEMS Inertial Sensing Technology.* [En línea] [Citado el: 2 de septiembre de 2018.] https://www.sensorsmag.com/components/overview-mems-inertial-sensing-technology.

5. Xsens. XSens. *Inertial sensors.* [En línea] [Citado el: 2 de septiembre de 2018.] https://www.xsens.com/tags/inertial-sensors/.

6. *Implementing a Sensor Fusion Algorithm for 3D.* Abyarjoo F., Barreto A., Cofino J., Ortega F.R. vol 313, s.l. : Springer, 2015, Vols. Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering. Lecture Notes in Electrical Engineering. doi: 10.1007/978-3-319-06773-5_41.

7. *Sensor Fusion-Based Attitude Estimation Using Low-Cost MEMS-IMU for Mobile Robot Navigation.* pp. 465-468, Chongqing : IEEE , 2011, Vol. Joint International Information Technology and Artificial Intelligence Conference. doi: 10.1109/ITAIC.2011.6030374.

8. E, Weisstein. Wolfram Mathworld. *Euler Angles.* [En línea] [Citado el: 31 de agosto de 2018.] http://mathworld.wolfram.com/EulerAngles.html.

9. Pedley, M. NXP Freescale Semiconductor. *Tilt Sensing Using a Three—Axis Accelerometer.* [En línea] marzo de 2013. http://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf. AB3461.

10. *MOCA: A Low-Power, Low-CostMotion Capture System Based on Integrated Accelerometers.* Farella E., Benini L., Ricc B., Acquaviva A. s.l. : Hindawi, 2007, Vol. Advances in Multimedia. doi:10.1155/2007/82638.

11. *Using Inertial Sensors for Position and Orientation Estimation.* Kok M., Hol J.D., Schön T.B. Vol. 11, pp1-153, s.l. : ARXIV, 2017, Vol. Foundations and Trends in Signal Processing. doi:10.1561/2000000094.

12. Arduino. *What is Arduino?* [En línea] [Citado el: 6 de septiembre de 2018.] https://www.arduino.cc/en/Guide/Introduction.

13. —. *Arduino UNO rev3 Tech specs & documentation.* [En línea] [Citado el: 14 de agosto de 2018.] https://store.arduino.cc/genuino-uno-rev3.

14. Atmel. microchip. *ATmega328/P datasheet.* [En línea] [Citado el: 16 de agosto de 2018.] http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf.

15. Analog Devices. *ADXL335 datasheet rev B, Small, Low Power, 3-Axis ±3 g Accelerometer.* [En línea] [Citado el: 6 de septiembre de 2018.] http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf.

16. Texas Instrument. ti. *CD4051B, CD4052B, CD4053B datasheet rev H, CD405xB CMOS Single 8-Channel Analog Multiplexer/Demultiplexer With Logic-Level Conversion.* [En línea] agosto de 1998. [Citado el: 6 de septiembre de 2018.] http://www.ti.com/lit/ds/symlink/cd4053b-mil.pdf. SCHS047I.

17. Texas Instruments. ti. *NA555, NE555, SA555, SE555 datasheet rev H, xx555 Precision Timers.* [En línea] september de 1973. [Citado el: 6 de septiembre de 2018.] http://www.ti.com/lit/ds/symlink/ne555.pdf. SLFS022I.

18. Processing. Processing 3. *Overview.* [En línea] [Citado el: 6 de septiembre de 2018.] https://processing.org/overview/.

19. J., Feinberg. Peasycam. *Mouse driven camera context for processing.* [En línea] [Citado el: 26 de agosto de 2018.] http://mrfeinberg.com/peasycam/.

20. AVR Libc. AVR Libc. *Power Management and Sleep Modes.* [En línea] [Citado el: 6 de septiembre de 2018.] https://www.nongnu.org/avr-libc/user-manual/group__avr__sleep.html.

21. —. AVR Libc. *Power Reduction Management.* [En línea] [Citado el: 6 de septiembre de 2018.] https://www.nongnu.org/avr-libc/user-manual/group__avr__power.html.

22. Arduino. . *Language reference.* [En línea] [Citado el: 14 de agosto de 2018.] https://www.arduino.cc/reference/en/.

23. Processing. *2d Translations.* [En línea] [Citado el: 9 de septiembre de 2018.] https://www.processing.org/tutorials/transform2d/.

24. —. *P3D.* [En línea] [Citado el: 26 de septiembre de 2018.] https://www.processing.org/tutorials/p3d/.

25. Xsens. Xsens. *An introduction to the begining of motion capture technologies.* [En línea] [Citado el: 2 de septiembre de 2018.] https://www.xsens.com/fascination-motion-capture/.

26. Arduino. Arduino Playground. *Sleep control.* [En línea] [Citado el: 14 de agosto de 2018.] https://playground.arduino.cc/Learning/ArduinoSleepCode.

8. Listado de siglas, abreviaturas y acrónimos

- **ADC:** *Analog to Digital Conerter* – Conversor Analógico a Digital. Módulo electrónico encargado de transformar una señal analógica en una señal digital con una precisión determinada en bits.
- **AHRS:** *Attitude and Heading Reference System* – Sistema de Referencia de Actitud y Rumbo. Sistemas de captura de movimiento de alta precisión enfocados en el área de la navegación aeronáutica. A diferencia de las IMUs, integran un procesado de datos con algoritmos de fusión sensorial.
- **EEPROM:** *Electrically Erasable Programable ROM* - ROM Programable y Borrable Eléctricamente. Memoria no volátil que puede ser reprogramada.
- **DSP:** *Digital Signal Processor* – Procesador Digital de Señales. Microprocesador con una arquitectura especialmente diseñada para el tratamiento digital de señales analógicas.
- **IDE**: *Integrated Development Environment* – Entorno de Desarrollo Integrado. Programa informático que proporciona servicios integrales para el desarrollo de software. Suelen estar compuestos, por lo menos, por un editor de código, un compilador y un depurador.
- **IMU:** *Inertial Measurement Unit* – Unidad de Medida Inercial. Dispositivo que analiza y devuelve información acerca de la aceleración lineal y velocidad angular usando sensores inerciales (acelerómetros y giroscopios).
- **I^2C:** *Inter-Integrated Circuit* – Circuito Inter-Integrado. Bus de comunicaciones serie síncrono desarrollado por Philips para la comunicación entre controladores en circuitos integrados. El I2C es un bus half-duplex, enfocado a aplicaciones de lecturas puntuales de registros.
- **I/O**: *Input/Output* – Entradas/Salidas. En electrónica hace referencia a los pines de chips (o señales de un sistema) que pueden ser controladas o leídas en el ciclo de ejecución. Son la forma de realizar intercambios de información entre el procesador y el mundo exterior.
- **ICSP:** *In Circuit Serial Programming* – Programación Serie En Circuito. Puertos para programación externa del chip integrados en el sistema final para la reprogramación del mismo.
- **ISR:** *Interruption Service Routine* – Rutona de Servicio de Interrupción. Funciones especiales que ejecutan instrucciones ante una interrupción.
- **MEMS:** *Micro-Electro-Mechanical System* – Sistema Micro-Electromecánico. Dispositivos electrónicos del tamaño $1 \div 100\ \mu m$ con partes mecánicas móviles.

- **PCB**: *Printed Circuit Board* – Placa de Circuito Impreso. Soporte físico que conecta eléctricamente y soporta mecánicamente diversos componentes electrónicos mediante buses conductores.
- **PPM**: *Pulse Position Modulation* – Modulación por Posición de Pulsos. Técnica de modulación de señales para la codificación de datos sobre señales de pulsos. Emplea cambios en la posición de pulsos del mismo ancho y amplitud, siendo un 0% una señal que oscila en la misma frecuencia.
- **PWM**: *Pulse Width Modulation* – Modulación por Ancho de Pulsos. Técnica de modulación de señales para la codificación de datos sobre señales de pulsos. Emplea cambios en los tiempos de activación/desactivación de la señal portadora, siendo un 100% de señal la correspondiente a una señal activa durante todo el ciclo.
- **RAM:** *Random Access Memory* – Memoria de Acceso Aleatorio. Almacenamiento volátil de datos de trabajo que permite la lectura y escritura en cualquier posición de esta.
- **RISC:** *Reduced Instruction Set Computing* – Computador con Conjunto de Instrucciones Reducido. Diseño de CPU con instrucciones de tamaño fijo y simples, que permite reducir el número de ciclos de ejecución necesarios por cada instrucción.
- **ROM:** *Read Only Memory* – Memoria de Solo Lectura. Memoria no volátil ideada para retener los datos grabados inicialmente y no permitir su posterior modificación.
- **SPI:** *Serial Peripheral Interface* – Interfaz Serie de Periféricos. Bus serie síncrono desarrollado por Motorola para la comunicación entre controladores en circuitos integrados. El SPI es un bus full-dúplex, enfocado a aplicaciones de *streaming* de datos.
- **SR**: Set-Reset – Activación- Desactivación. Biestable asíncrono con dos entradas de control, una de activación y otra de desactivación.
- **SRAM**: *Static Random Access Memory* – Memoria Estática de Acceso Aleatorio. Memoria RAM no volátil que retiene, mediante biestables, el estado de cada bit de datos.
- **TTL:** *Transistor-Transistor Logic -Lógica* Transistor a Transistor. Tecnología de I/O de dispositivos en la que se emplean transistores bipolares en ambos extremos del intercambio de información. Suele trabajar sobre 5V.
- **UART:** *Universal Asynchronous Receiver-Transmitter* – Transmisor-Receptor Asíncrono Universal. Dispositivo hardware que se encarga de gestionar las trasmisiones asíncronas en las comunicaciones serie.

- **USB**: *Universal Serial Bus* – Bus serie universal. Bus de comunicaciones que establece a través de un estándar industrial las especificaciones de los cables, conectores y protocolos de conexión, comunicación y alimentación entre ordenadores y dispositivos electrónicos de uso general.

Anexos

Anexo I. Código Arduino

```
/*********************************************************************
     Master's thesis - Motion capture using low cost hardware based on
accelerometers and Arduino
     Main program file - Sleep control
     Make the arduino board fall into sleep mode (IDLE) usign external
interrupts on pin 3 to wake it up.
     Code() function will be summoned every interrupt, accesing the
cyclic tab.
     Author: Ruben Imbers
     Date: 09 of June 2018

*********************************************************************/

#include <avr/sleep.h>
#include <avr/power.h>

/*****
     Start of custom parameters
 *****/
const byte filterMode = 2;
const long baudRate = 115200L;
const byte sensorNumber = 4;
const String startFrame = "####";
const String endFrame = "%%%%";
/*****
     End of custom parameters
 *****/

const byte timerPin = 3;
const byte muxCPin = 5;
const byte muxBPin = 6;
const byte muxAPin = 7;
const byte muxXPin = 0;
const byte muxYPin = 1;
const byte muxZPin = 2;
```

```
volatile bool newCycle = false;
volatile unsigned long cycle = 0;

float a;
float aux;
float auxX;
float auxY;
float auxZ;

byte i = 1;
byte j = 1;

unsigned long cycleTimeStamp = 0;

int values [7][3]; // array for analog reading from each sensor

void setup() {
  sleep_enable();
  pinMode(timerPin, INPUT);
  pinMode(muxCPin, OUTPUT);
  pinMode(muxBPin, OUTPUT);
  pinMode(muxAPin, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(timerPin), CycleInterrupt,
RISING);
  noInterrupts();
  analogReference(DEFAULT);
  analogRead(muxXPin);
  analogRead(muxYPin);
  analogRead(muxZPin);
  sleep_disable();
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 4; j++) {
      values[j][i] = 0;
    }
  }
  Serial.begin(baudRate);
  while (!Serial) {
    ;
  }
  interrupts();
```

```
}

void loop() {
  if (newCycle) {
    cycleTimeStamp = micros() ;
    Serial.begin(baudRate);
    Cycle();
    if (Serial) {
      Serial.print(startFrame); //Start of transmission
      for (i = 0; i < sensorNumber; i++) {
        auxX = - values[i][0] / 100.0;
        auxY = - values[i][1] / 100.0;
        auxZ = - values[i][2] / 100.0;
        Serial.print(auxX);
        Serial.print(',');
        Serial.print(auxY);
        Serial.print(',');
        Serial.print(auxZ);
        if (i < sensorNumber - 1) Serial.print(',');
      }
      Serial.print(endFrame); //End of transmission
      Serial.print(",");
      Serial.print(cycle);
      Serial.print(",");
      Serial.println(cycleTimeStamp);
    }
    Serial.end();
  }
  newCycle = false;
  power_adc_disable(); // PROBAR
  interrupts();
  //set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_enable();
  sleep_mode();
}

void CycleInterrupt () {
  noInterrupts();
  sleep_disable();
  power_adc_enable(); // PROBAR
```

```
  cycle += 1;
  newCycle = true;
}
/** filterRead
    Filter the analog reading using a first order filter that follows
the next formula:

      Out = previousValue * a + (1 - a) * newValue

    a gets its vaue from the filter used:

      0 -> a = 0.     No filter is applied
      1 -> a = 0.750. Soft filter with a stablish time (63%) of 4T
      2 -> a = 0.937. Medium filter with a stablish time (63%) of 16T
      3 -> a = 0.992. Hard filter with a stablish time (63%) of 128T
*/
float filterRead (float previousValue, float newValue) {
  switch (filterMode) {
    case 1:
      a = 0.75;
      break;
    case 2:
      a = 0.875;
      break;
    case 3:
      a = 0.992;
      break;
    default:
      a = 0;
  }
  return (a * previousValue + (1 - a) * newValue);
}

/** Cycle
    check the serial link and enable the MUX for reading each sensor
consecutively
    read twice the analog input to eliminate distortions from previous
reads and capacitor discharges
    map the input (0 - 3.3V) to the sensor output scaled by 100 (-300
- 300 cg)
```

```
    filter the output using the selected filter mode
    write the result in the serial bus
*/
void Cycle() {
  for (i = 0; i < sensorNumber; i++) {
    switch (i) {
      case 0:
        digitalWrite(muxAPin, true);
        digitalWrite(muxBPin, false);
        digitalWrite(muxCPin, false);
        break;
      case 1:
        digitalWrite(muxAPin, false);
        digitalWrite(muxBPin, true);
        digitalWrite(muxCPin, false);
        break;
      case 2:
        digitalWrite(muxAPin, true);
        digitalWrite(muxBPin, true);
        digitalWrite(muxCPin, false);
        break;
      case 3:
        digitalWrite(muxAPin, false);
        digitalWrite(muxBPin, false);
        digitalWrite(muxCPin, true);
        break;
      case 4:
        digitalWrite(muxAPin, true);
        digitalWrite(muxBPin, false);
        digitalWrite(muxCPin, true);
        break;
      case 5:
        digitalWrite(muxAPin, false);
        digitalWrite(muxBPin, true);
        digitalWrite(muxCPin, true);
        break;
      case 6:
        digitalWrite(muxAPin, true);
        digitalWrite(muxBPin, true);
        digitalWrite(muxCPin, true);
```

```
        break;
    }
    analogRead(muxXPin);
    aux = map(analogRead(muxXPin), 135, 540, -300, 300);
    values[i][0] = filterRead(values[i][0], aux);
    analogRead(muxYPin);
    aux = map(analogRead(muxYPin), 135, 540, -300, 300);
    values[i][1] = filterRead(values[i][1], aux);
    analogRead(muxZPin);
    aux = map(analogRead(muxZPin), 135, 540, -300, 300);
    values[i][2] = filterRead(values[i][2], aux);
  }
}
```

Anexo II. Código Processing

```
/*****************************************************************************
************************************************
 *  Master's thesis - Motion capture using low cost hardware based on
accelerometers and Arduino
 *  Main program file - Data processing and display
 *  Make the PC stablish a serial link with the arduino board, read
accelerometer data and display it in a 3D canvas.
 *  Author: Ruben Imbers
 *  Date:  2018

*****************************************************************************
************************************************/

import peasy.*;
import processing.serial.*;

/*****
 *   Start of custom parameters
 *****/
String startFrame = "####";
String endFrame = "%%%%";
float armLength = 20;
float armWidth = 2;
int baudRate = 115200;
String comPort = "COM6";
/*****
 *   End of custom parameters
 *****/

PeasyCam cam;

Serial usbPort;
int comWatchdog = 0;
int cont = 0;
boolean newData = false;
boolean connected = false;
boolean connectionEnabled = true;
```

```
String data[];
String info;

float anglesL1[]; // Angles of left arm Roll (X), Pitch(Y)
float anglesL2[]; // Angles of left forearm Roll (X), Pitch(Y)
float anglesR1[]; // Angles of right arm Roll (X), Pitch(Y)
float anglesR2[]; // Angles of right forearm Roll (X), Pitch(Y)

void setup() {
  size(920, 800, P3D);
  background(240);
  frameRate(50);

  cam = new PeasyCam(this, 100);
  cam.setMinimumDistance(150);
  cam.setMaximumDistance(500);
  cam.rotateY(.25);
  cam.rotateX(.25);

  anglesL1 = new float[2];
  anglesL2 = new float[2];
  anglesR1 = new float[2];
  anglesR2 = new float[2];

  anglesL1[0] = 0;
  anglesL1[1] = 0;
  anglesR1[0] = 0;
  anglesR1[1] = 0;
  anglesL2[0] = 0;
  anglesL2[1] = 0;
  anglesR2[0] = 0;
  anglesR2[1] = 0;

  connected = connectSerialPort(comPort, baudRate, 3);
}

void draw () {
  background(240);
  lights();
  if (connected) comWatchdog +=1;
```

```
  cont +=1;
  watchCom();

  drawText();

  drawAxes();
  drawLeftArm(anglesL1[0], anglesL1[1], anglesL2[0], anglesL2[1]);
  drawRightArm(anglesR1[0], anglesR1[1], anglesR2[0], anglesR2[1]);
  drawBody();
}
/**  Functions for serial link communication and data processing
 *       - Stablish serial communication with arduino
 *       - Identify valid frames
 *       - Extract each sensor's acceleration
 *       - Proccess accelerations and turn them into pitch and roll
angles
 */



/**  connectSerialPort
 *   tries to open the serial port portNumber at a rate
 *   of baudRate bauds connectionAttempts times
 *   returns true if succeeds, false if not
 */
boolean connectSerialPort(String portNumber, int baudRate, int
connectionsAttempts) {
  int tries = 0;
  while (tries < connectionsAttempts) {
    try {
      usbPort = new Serial(this, portNumber, baudRate);
    }
    catch (RuntimeException e) {
      System.err.println("Failed to connect to device in attempt " +
str(tries + 1));
      tries++;
      delay(10);
      continue;
    }
    break;
  }
```

```
  if (tries < connectionsAttempts) {
    println("Connected to Arduino");
    comWatchdog = 0;
    cont=0;
    return true;
  } else {
    return false;
  }
}

/**  disconnectSerialPort
 *   tries closing the serial port
 */
boolean disconnectSerialPort() {
  boolean disc = false;
  try {
    usbPort.clear();
    usbPort.stop();
    disc = true;
  }
  catch (RuntimeException e) {
    System.err.println("Failed to disconnect from device");
  }
  if (disc) {
    println("Disconnected from Arduino");
    return false;
  } else {
    return true;
  }
}

/**  watchCom
 *   waits for the serial buffer to have the minimum desired size
 *   and analizes the data searching for 12 comma-separated values
 *   convert ASCII into float values to get 3-axis accelerations
 *   turn this accelerations into Roll and Pitch angles
 */
void watchCom() {
  if (!connected && connectionEnabled && cont%100==0) connected =
connectSerialPort(comPort, baudRate, 3);
```

```
  if (connected && !connectionEnabled) connected =
disconnectSerialPort();

  if (connected && connectionEnabled) {
    if (usbPort.available() >= 67 ) {
      comWatchdog = 0;
      byte serialData[] = usbPort.readBytes();
      String stringSerialData = join(str(char(serialData)), "");
      String startTruncData[] = splitTokens(stringSerialData,
startFrame);
      if (startTruncData[0].length() < stringSerialData.length() &&
startTruncData[0].length() >= 63) {
        String endTruncData[] = splitTokens(startTruncData[0],
endFrame);
        if (endTruncData[0].length() < startTruncData[0].length() &&
endTruncData[0].length() >= 59) {
          String rawData[] = split(endTruncData[0], ",");
          if (rawData.length == 12 && endTruncData.length > 1) {
            newData = true;
            data = rawData;
            info = endTruncData[1];
          } else usbPort.clear();
        } else usbPort.clear();
      } else usbPort.clear();
    } else if (comWatchdog >= 10 && cont > 100) {
      connected = false;
    }
  }

  if (newData) {
    for (byte i = 0; i< 4; i++) {
      float aX = float(data[i * 3]);
      float aY = float(data[i * 3 + 1]);
      float aZ = float(data[i * 3 + 2]);
      float rollX = 0;
      float pitchY = 0;
            try {
        if (aZ >= 0)  rollX = atan2(aY, sqrt(pow(aZ, 2) + 0.01*pow(aX,
2)));
```

```
        else  rollX = atan2(aY, -sqrt(pow(aZ, 2) + 0.01*pow(aX, 2)));
        pitchY = atan2(-aX, sqrt(pow(aY, 2) + pow(aZ, 2)));
      }
      catch (RuntimeException e) {
        System.err.println(e);
      }
      switch(i) {
      case 0:
        anglesL1[0] = rollX;
        anglesL1[1] = pitchY;
        break;
      case 1:
        anglesL2[0] = rollX;
        anglesL2[1] = pitchY;
        break;
      case 2:
        anglesR1[0] = rollX;
        anglesR1[1] = pitchY;
        break;
      case 3:
        anglesR2[0] = rollX;
        anglesR2[1] = pitchY;
        break;
      }
    }
    newData = false;
  }
}
void drawText() {
  textSize(5);
  fill(0);
  textMode(SHAPE);
  if (connectionEnabled) text("Press 'a' to disable serial link", -80,
-80, -30);
  else text("Press 'a' to enable serial link", -80, -80, -30);
  text("Configured serial port is " + comPort + " at " + str(baudRate)
+ " bauds", -80, -75, -30);
  text("X",-23,32,-10);
  text("Y",-36,45,-10);
  text("Z",-41,31,-5);
```

```
  fill(255);
}

void drawBody() {
  box(20, 30, 5);  // Draw torax

  pushMatrix();     //Draw neck and head
  translate(0, -17, 0);
  box(3, 4, 3);
  translate(0, -7, 0);
  sphere(6);
  popMatrix();

  pushMatrix();    //Draw right shoulder sphere
  translate(10, -15, 0);
  sphere(2.5);
  popMatrix();

  pushMatrix();    //Draw right shoulder sphere
  translate(-10, -15, 0);
  sphere(2.5);
  popMatrix();
  sphere(3);
}

void drawAxes() {
  pushMatrix();
  translate(-30, 30, -10);
  box(10, 1, 1);
  translate(-5, 5, 0);
  box(1, 10, 1);
  translate(0, -5, 5);
  box(1, 1, 10);
  translate(0, 0, -5);
  sphere(2);
  popMatrix();
}

void drawRightArm(float aX1, float aY1, float aX2, float aY2) {
  pushMatrix();
```

```
  translate(10, -15, 0); //Potion drawing point in right shoulder
  rotateX(TWO_PI/4);
  rotateX(aX1);
  rotateY(aY1);
  translate(0, -armLength/2, 0);
  box(armWidth, armLength, armWidth);

  pushMatrix();
  translate(0, -armLength/2, 0);
  rotateY(-aY1);
  rotateX(-aX1);
  rotateX(aX2);
  rotateY(aY2);
  translate(0, -armLength/2, 0);
  box(armWidth, armLength, armWidth);
  translate(0, -armLength/2, 0);
  sphere(2.5);
  translate(0, armLength, 0);
  sphere(2.5);
  popMatrix();


  popMatrix();
}

void drawLeftArm(float aX1, float aY1, float aX2, float aY2) {
  pushMatrix();
  translate(-10, -15, 0); //Potion drawing point in right shoulder
  rotateX(TWO_PI/4);
  rotateX(aX1);
  rotateY(aY1);
  translate(0, -armLength/2, 0);
  box(armWidth, armLength, armWidth);

  pushMatrix();
  translate(0, -armLength/2, 0);
  rotateY(-aY1);
  rotateX(-aX1);
  rotateX(aX2);
  rotateY(aY2);
```

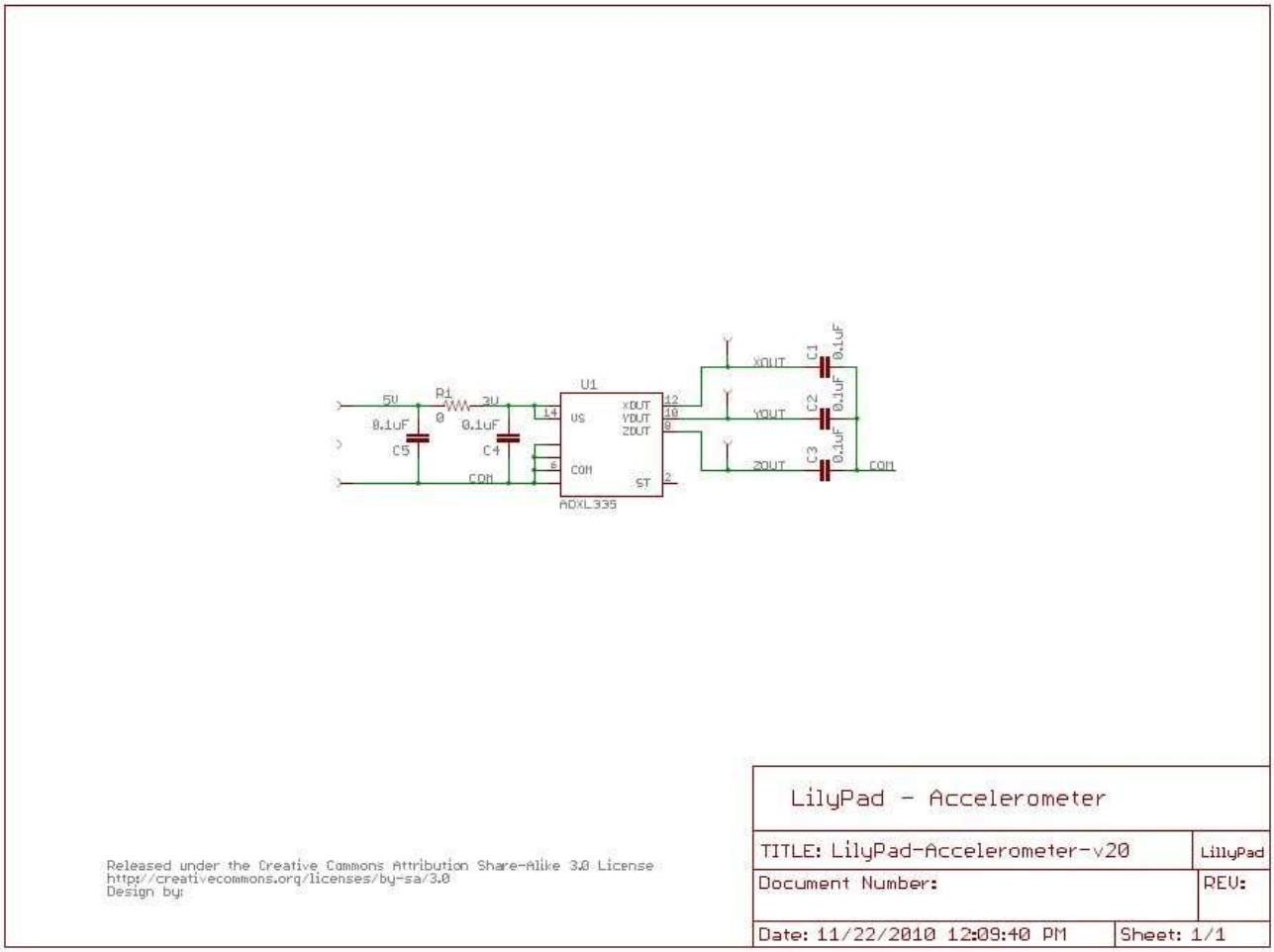
```
  translate(0, -armLength/2, 0);
  box(armWidth, armLength, armWidth);
  translate(0, -armLength/2, 0);
  sphere(2.5);
  translate(0, armLength, 0);
  sphere(2.5);
  popMatrix();

  popMatrix();
}

void keyReleased() {
  if (key == 'a' || key == 'A') {
    connectionEnabled = ! connectionEnabled;
  }
}
```

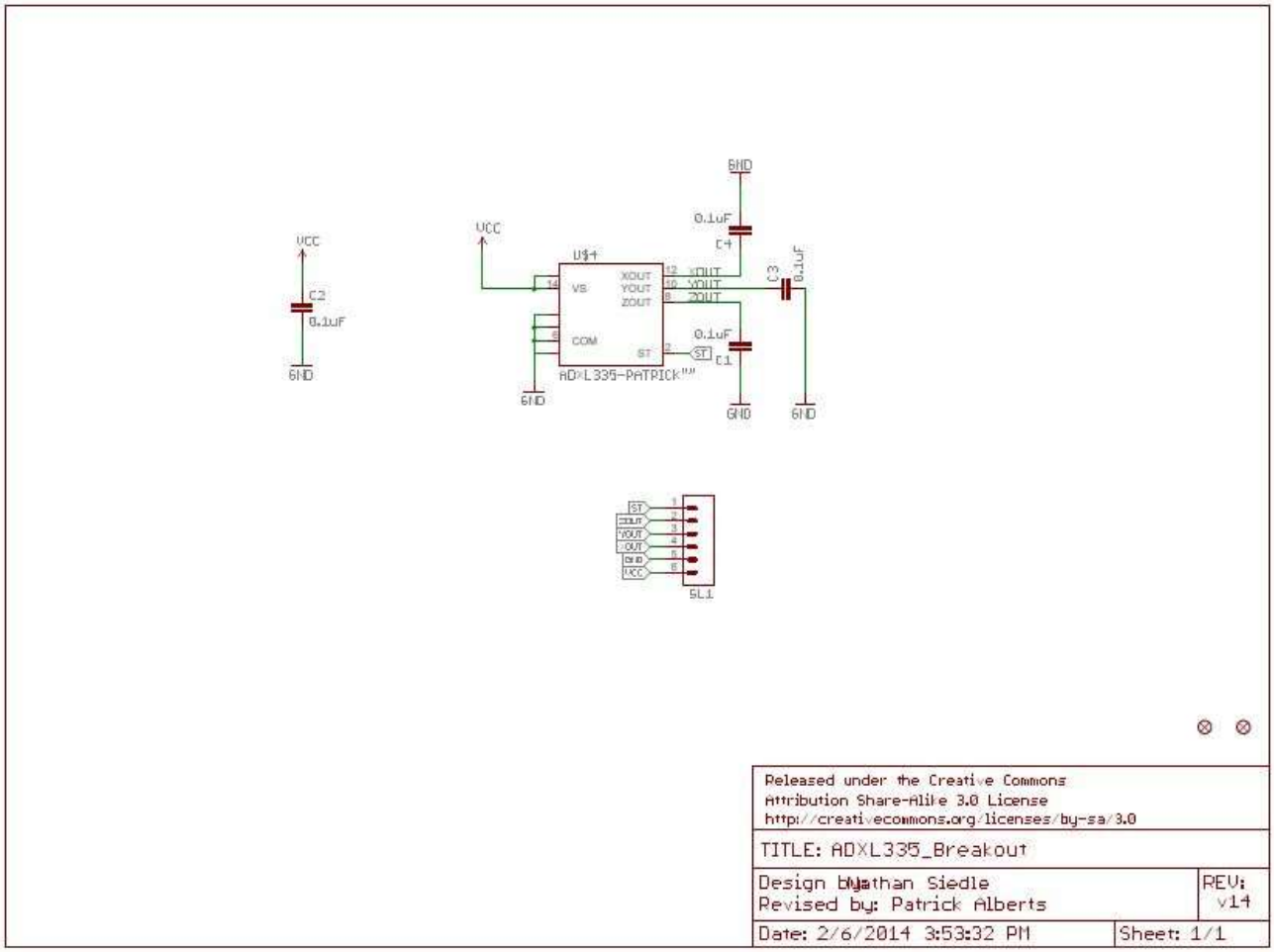
Anexo III. Esquemas eléctricos

Anexo III.I. Placa de prototipo Lilypad para ADXL335



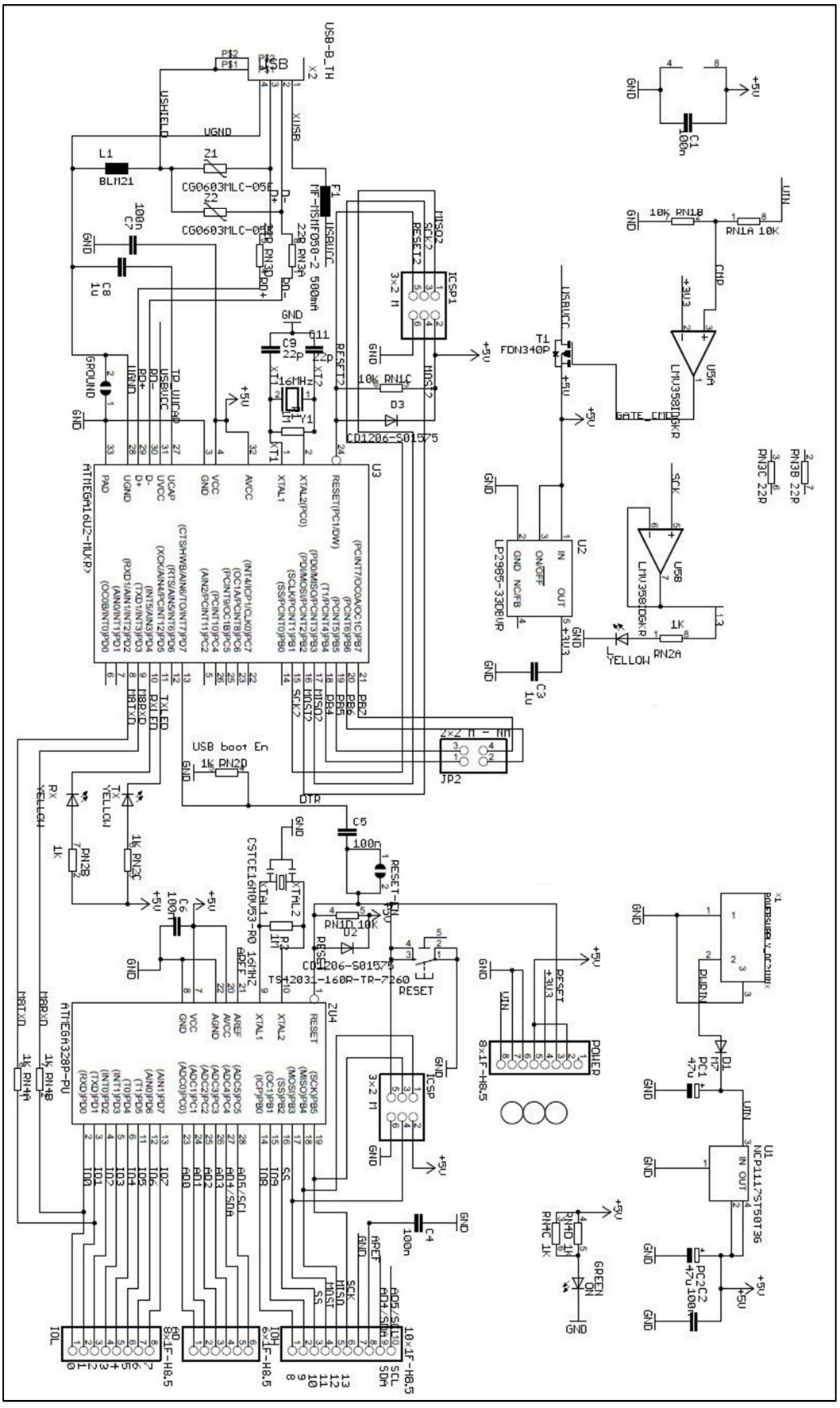
Esquema eléctrico de la placa de prototipo de Lilypad para el acelerómetro ADXL335. Fuente Sparkfun.com.

Anexo III.II. Placa de prototipos Sparkfun para ADXL335



Esquema eléctrico de la placa de prototipo de Sparkfun para el acelerómetro ADXL335. Fuente Sparkfun.com.

Anexo III.III. Placa Arduino



Esquema eléctrico placa Arduino UNO rev3. Fuente Arduino.cc.