Máster en Ingeniería de Sistemas y Control

Proyecto Final de Máster

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

Curso 2018/2019 – Convocatoria de Septiembre

Estudiante / Autor:	D. Tristán Mas Carrascosa
Profesores / Directores:	D. José Sánchez Moreno D. David Moreno Salinas

Máster en Ingeniería de Sistemas y Control

Proyecto Final de Máster

Tipo A: Proyecto específico propuesto por un profesor

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

Curso 2018/2019 – Convocatoria de Septiembre

Estudiante / Autor:	D. Tristán Mas Carrascosa
Profesores / Directores:	D. José Sánchez Moreno D. David Moreno Salinas

HOJA DE CALIFICACIONES

AUTORIZACIÓN

Autorizo 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.

Tristán Mas Carrascosa

RESUMEN

El problema de captura de movimiento puede ser resuelto mediante diferentes enfoques. En este proyecto se utiliza la tecnología de sensores inerciales de bajo costo para la medición de aceleración lineal y velocidad angular de objetos en movimiento en el espacio tridimensional y en base a dichas mediciones – fusionadas a través de un filtro complementario – se calcula su orientación, parametrizada en forma de cuaterniones unitarios.

La recolección de los datos de los sensores y su transmisión al servidor se encomienda a un microcontrolador que ejecuta un programa escrito en lenguaje Arduino (adaptación de C++).

El código del programa que maneja los datos de los sensores, calcula la orientación de los objetos y genera la representación gráfica está escrito en lenguaje Python.

Para la representación gráfica tridimensional del movimiento se recurre al módulo VPhython, generando una ventana gráfica en un navegador web en la que se muestra el modelo que representa a los objetos en movimiento.

PALABRAS CLAVE

Captura de movimiento, sensor inercial, acelerómetro, giroscopio, filtro complementario, parametrización de orientación, cuaternión, representación tridimensional, Python, VPython, Arduino

ABSTRACT

There are different approaches to solve the motion capture problem. This project utilizes low-cost inertial sensors technology to measure the linear acceleration and the angular rate of objects moving in tridimensional space and, based on that data (fused in a complementary filter), the orientation of the object is calculated in form of unit quaternion.

The data collection from the sensors and the transmission to the server are entrusted to a microcontroller that runs a code written in Arduino language (C++ adaptation).

The software that manage the data provided by the sensors, computes the object's orientation and generates the graphic representation is written in Python language.

In order to obtain the tridimensional graphic representation of the movement, the VPython module has been used. Therefore, a graphic window is generated in a web browser, showing the model representing the objects in motion.

KEY WORDS

Motion capture, inertial sensor, accelerometer, gyroscope, complementary filter, orientation parametrization, quaternion, tridimensional representation, Python, VPython, Arduino.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 Motivación

El presente proyecto ha sido realizado por D. Tristán Mas Carrascosa con el fin de dar cumplimiento a los requisitos exigidos por la Universidad Nacional de Educación a Distancia (UNED) y la Universidad Complutense de Madrid (UCM) para recibir la titulación de Máster en Ingeniería de Sistemas y Control.

Este Proyecto Final de Máster (PFM) está calificado como de Tipo A, esto es, se trata de un proyecto específico propuesto por un profesor. La propuesta del PFM ofertado el profesor D. José Sánchez Moreno indica:

> *Título: Sistema de captura de movimientos basado en sistemas electrónicos de bajo coste.*
> *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, los cuales estarán repartidos por diversas partes del cuerpo para poder capturar los movimientos y posteriormente reproducirlos en un computador.*
> *Para la consecución del proyecto el alumno deberá adquirir conocimientos sobre programación con el IDE de Arduino así como conceptos básicos de electrónica y sensores, para realizar el montaje del sistema y la adquisición de datos, y conocimientos sobre filtrado de señales, ya que estos datos deberán ser filtrados para poder reconstruir en un computador los movimientos capturados mediante el adecuado procesado de las señales y datos recogidos.*

1.2 Objetivos

Este proyecto tiene los siguientes objetivos:

- Estudiar el problema de la captura del movimiento
- Construir un sistema físico para la captura del movimiento basado en sensores inerciales
- Procesar los datos entregados por sensores inerciales
- Obtener una parametrización de la orientación en base a datos de sensores inerciales
- Reproducir gráficamente los movimientos de objetos reales

1.3 Alcance

Dentro del proyecto se contempla el siguiente alcance:

- Hardware
    - Construcción del sistema físico en base a módulos electrónicos comerciales (sensores y microcontrolador)
- Software
    - Programación a nivel microcontrolador para la captura de las señales de los sensores
    - Programación a nivel microcontrolador, computador o servidor para el procesado de señales y datos capturados
    - Programación a nivel computador o servidor para la reproducción gráfica de los movimientos capturados

No se contempla dentro del alcance del proyecto el diseño de módulos físicos partiendo de componentes electrónicos.

1.4 Relación con otros trabajos

Este proyecto queda encuadrado dentro de los Proyectos Final de Máster que desarrollan una temática propuesta por un profesor. En este sentido, otros alumnos – como D. Rubén Imbers – han tomado anteriormente el problema de la captura de movimientos basados en sistemas electrónicos de bajo coste.

El presente proyecto, en comparación con proyectos anteriores, aborda el mismo problema con otros enfoques, resultando un trabajo diferente y complementario.
Los principales aportes frente a lo anteriormente desarrollado en otros trabajos son:

- Se utilizan sensores inerciales de seis grados de libertad (combinación de acelerómetros y giroscopios) en lugar de sensores de tres grados de libertad (acelerómetros), lo que permite superar ciertas restricciones técnicas.
- Como consecuencia de la utilización de datos de aceleración lineal y velocidad de giro, es necesario implementar un algoritmo – en este caso un filtro complementario – que fusione dichos datos para generar la estimación de la orientación.
- Se calcula la orientación basada en la parametrización con cuaterniones unitarios en lugar de matrices de rotación, lo que resulta computacionalmente más eficiente.
- Se comunica sensores y microcontrolador mediante bus en lugar de la incorporación de multiplexores al sistema.
- Se utiliza el lenguaje de programación Python para la implementación de las funcionalidades de generación de la orientación y generación de la imagen.

Para la elaboración de este proyecto se han estudiado los siguientes trabajos (más detalles en apartado de bibliografía), que se consideran documentos básicos para establecer la base teórica sobre la que construir el sistema de captura del movimiento:

- Using Inertial Sensors for Position and Orientation Estimation

- Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs

La primera publicación es un manual que desarrolla un modelo probabilístico para el cálculo de la estimación de la orientación, comparando los resultados de aplicación de diferentes técnicas (optimización, filtro extendido de Kalman, filtro complementario) de fusión de datos de aceleración lineal y velocidad angular. De este trabajo se han considerado los contenidos teóricos sobre sistemas de coordenadas, parametrización de la orientación y modelo de medición de los sensores.

La segunda publicación de las anteriormente mencionadas desarrolla un algoritmo para determinar la orientación a partir de la resolución algebraica de sistemas, tomando la parametrización con cuaterniones unitarios y fusionando los datos de aceleración lineal y velocidad angular por medio de un filtro complementario. De este trabajo se ha tomado el algoritmo de determinación de la orientación implementado en este proyecto.

2 ESTADO DEL ARTE

Se ha analizado el estado del arte en relación con los siguientes aspectos clave para el desarrollo del proyecto:

- Captura del movimiento
- Parametrización de la orientación
- Sensores
- Controladores
- Comunicaciones
- Software
- Representación gráfica

2.1 Captura del movimiento

Desde los años 70 se vienen desarrollando diferentes técnicas de captura del movimiento (conocidas bajo la denominación "mocap") en el ámbito de la biomecánica con diferentes aplicaciones (medicina, deporte, ocio, militares). La bibliografía indica que los métodos utilizados para la captura del movimiento se pueden clasificar en dos grandes familias:

- Métodos ópticos
- Métodos físicos:
    - Métodos mecánicos
    - Métodos magnéticos
    - Métodos inerciales

Los métodos ópticos se basan en el uso de sistemas de visión artificial que identifican puntos característicos de objetos en movimiento y parametrizan su posición. De esta manera, conocidas las posiciones de sus puntos característicos, se puede modelar gráficamente el movimiento de los objetos observados.

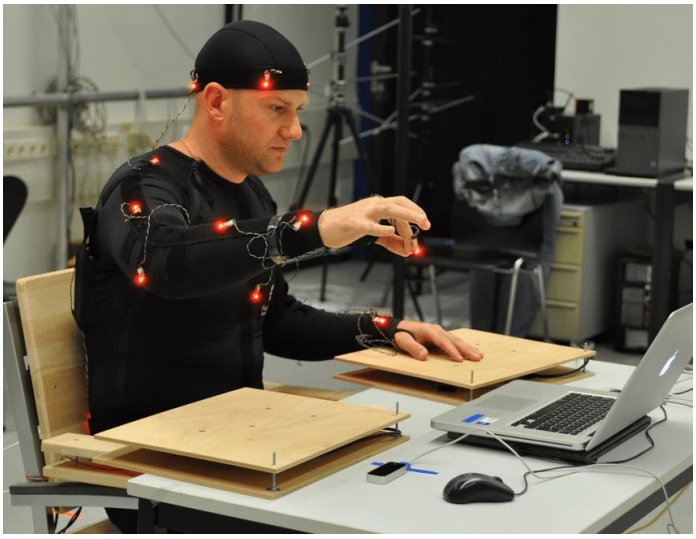

Ilustración 1. Sistema óptico de captura del movimiento

(fuente: Myroslav Bachynskyi)

Los métodos físicos se pueden subclasificar en métodos mecánicos, magnéticos e inerciales.

Los métodos mecánicos toman información – medida por sensores (potenciómetros, encoders) – del movimiento relativo entre los elementos de un exoesqueleto montado sobre un cuerpo articulado en movimiento. Este tipo de soluciones pueden ser muy precisas y de costo contenido.

Ilustración 2. Sistema mecánico de captura de movimiento

(fuente: Metamotion)

Los métodos magnéticos se basan en la utilización de sensores de flujo magnético para la determinación de la posición y la orientación. Este tipo de sensores son sensibles a la presencia de alteraciones del flujo por la presencia de equipos electromagnéticos en su entorno.

Finalmente, entre los métodos físicos destacan los métodos inerciales. Éstos utilizan las mediciones de sensores de aceleración lineal y velocidad angular, de las que se infieren las posiciones y orientaciones de los objetos analizados.

2.2 Parametrización de la orientación

Dado un objeto en un espacio tridimensional, se puede determinar su orientación indicando una serie de parámetros. Esto es lo que se conoce como parametrización de la orientación.

Las cuatro maneras más utilizadas para parametrizar la orientación de un objeto son:

- Matrices de rotación
- Vectores de rotación

- Ángulos de Euler
- Cuaterniones unitarios

Es posible la conversión entre cada uno de los sistemas de parametrización de la orientación. Por ejemplo, es inmediata la obtención de la matriz de rotación conocidos los términos del cuaternión unitario que parametriza una orientación dada.

Las diferencias entre los sistemas de parametrización se centran en el número de parámetros que manejan y la unicidad de la orientación parametrizada, lo que tiene relación con la carga computacional del cálculo de la orientación y la problemática de orientaciones singulares que no permiten ser parametrizadas de forma inequívoca.

Tabla 1. Características de sistemas de parametrización de la orientación

Sistema de parametrización	**Número de parámetros**	**Orientación única**
Matrices de rotación	9	Si
Vectores de rotación	3	No (Wrapping)
Ángulos de Euler	3	No (Wrapping y Gimbal lock)
Cuaterniones unitarios	4	No (q = -q)

2.3 Sensores

En este apartado, el estudio se ha centrado en sensores inerciales de bajo costo basados en tecnología MEMS (Micro Electro Mechanical Systems), que realmente

son sensores integrados (Giroscopios / Acelerómetros / Magnetómetros) de 6 o 9 grados de libertad.

Se han descartado los sensores basados únicamente en acelerómetros – 3 grados de libertad – dado que generan indeterminaciones en determinadas circunstancias (giro sobre el eje de la gravedad, "gimbal lock"), lo que limita su campo de aplicación.

El mercado ofrece sensores integrados por combinación de:

- Giroscopio de 3 ejes y acelerómetro de 3 ejes
- Acelerómetro de 3 ejes y magnetómetro de 3 ejes
- Giroscopio de 3 ejes, acelerómetro de 3 ejes y magnetómetro de 3 ejes

Algunas de las alternativas de módulos electrónicos que se encuentran en el mercado (incluyendo coste estimativo) son:

Tabla 2. Alternativas comerciales de sensores (módulos) de bajo costo

Fabricante	Modelo	Sensores	Coste
Sparkfun	IMU MPU 9250	MPU 6500 + AK8963	US$ 15
Pololu	MinIMU-9	LSM6DS33 + LIS3MDL	US$ 15
DF Robot	6DOF Sensor	MPU 6050	US$ 10
DF Robot	BMI 160	BMI 160	US$ 10
Grove	9DoF	MPU 9250	US$ 15
Sparkfun	LSM303C	LSM303C	US$ 15
SparkFun	9DoF Sensor Stick	LSM9DS1	US$ 15

Se han tabulado las principales características de los sensores que se utilizan en los módulos identificados anteriormente

Tabla 3. Características de sensores con 9 grados de libertad

IMU	MPU 9250	LSM 9DS1
Fabricante	Inven Sense	ST Micro.
Rango Giroscopio	±250/±2000 °/seg	±245/±2000 °/seg
Rango Acelerómetro	±2 / ±16 g	±2 / ±16 g
Rango Magnetómetro	±4800 µT	±400 / ±1600 µT
Buffer	512 byte FIFO	198 byte FIFO

Tabla 4. Características de sensores con 6 grados de libertad

IMU	MPU6050	LSM303C	LSM 6DS3	BMI 160
Fabricante	Inven Sense	ST Micro.	ST Micro.	Bosch
Rango Giroscopio	±250/±2000 °/seg		±125/±2000 °/seg	±125/±2000 °/seg
Rango Acelerómetro	±2 / ±16 g	±2 / ±8 g	±2 / ±16 g	±2 / ±16 g
Rango Magnetómetro		±1600 µT		
Buffer	1024 byte FIFO	192 byte FIFO	8192 byte	1024 byte FIFO

2.4 Controladores

Se ha investigado sobre los controladores de bajo costo disponibles en el mercado y que permiten desempeñar las siguientes funciones:

- Comunicación con los sensores (maestro – esclavo)
- Captura de las mediciones entregadas por los sensores
- Comunicación con el servidor
- Entrega al servidor de las mediciones capturadas a los sensores
- Alimentación (gestión de la energía) a los sensores
- Verificación de conexión y buen funcionamiento de los sensores
- Interfaz con el usuario

En las siguientes tablas se muestran las alternativas de controladores que se encuentran en el mercado (incluyendo costo estimativo y principales características técnicas), en primer lugar, sin disponibilidad de comunicación WiFi y a continuación con dicha comunicación:

Tabla 5. Características de controladores sin comunicación WiFi

Controlador	Uno	Curiosity HPC
Fabricante	Arduino	Microchip
Procesador	ATmega328P (8 bits)	PIC16F18875 (8 bits)
E/S digital	14	36
Memoria	32 KB	14 KB
Comunicación	SPI / I2C	SPI / I2C
Alimentación	5V (3,3V circuito)	2,3V – 5,5V
Cargador batería	No	No
Costo estimativo	20 US$	30 US$

Tabla 6. Características de controladores con comunicación Wifi

Controlador	MKR 1000	NodeMCU v2	ESP8266 Thing Development Board	ESP32 Thing
Fabricante	Arduino	Amica	Sparkfun	Sparkfun
Procesador	SAMD21 Cortex-M0 (32 bits)	Tensilica L106 (32 bits)	Tensilica L106 (32 bits)	Tensilica LX6
E/S digital	8	9	9	18
Memoria	256 KB Flash 32 KB SRAM	4 MB Flash 128 KB SRAM	4 MB Flash 128 KB SRAM	4 MB Flash 520 KB SRAM
Comunicación	SPI / I2C	SPI / I2C	SPI / I2C	SPI / I2C
WiFi	Integrado	Integrado	Integrado	Integrado

Alimentación	5V (3,3V circuito)	5V (3,3V circuito)	5V (3,3V circuito)	5V (3,3V circuito)
Cargador batería	Integrado	No	No	Integrado
Costo estimativo	US$ 35	US$ 10	US$ 15	US$ 20

2.5 Comunicaciones

Se ha dividido el estudio del arte de las comunicaciones entre:

- Comunicaciones sensor – controlador
- Comunicaciones controlador – servidor

2.5.1 Comunicaciones sensor – controlador

Las alternativas más habituales para comunicar los sensores con el microcontrolador son:

- I2C
- SPI

El protocolo "Inter-integrated Circuit" (I2C) fue desarrollado por Philips en los años 80 y en 1992 se publicó su especificación.
Se trata de un bus que permite que varios dispositivos "esclavos" se comuniquen con uno (o varios) dispositivos "maestros".
I2C se basa en comunicación serie síncrona y permite comunicación half-duplex (comunicación entre dos dispositivos no simultánea en ambos sentidos): utiliza dos líneas de señal para el intercambio de información, una de reloj (SCL) y otra de datos (SDA). La selección del dispositivo "esclavo" con el que comunicar se realiza mediante el uso de una dirección en la trama de datos, lo que permite a cada dispositivo (con una dirección determinada) captar la comunicación y reconocer si le corresponde.

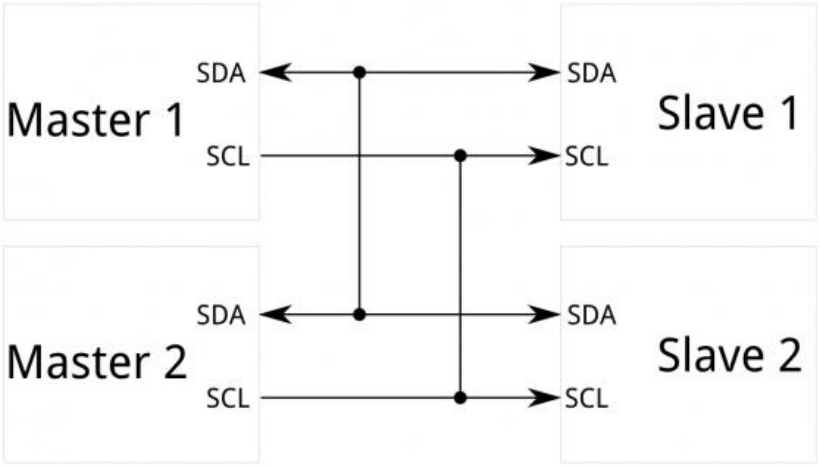

Gráfico 1. Bus I2C

El interfaz "Serial Peripheral Interface" (SPI) es de uso habitual para la comunicación entre microcontroladores y dispositivos periféricos como sensores o tarjetas de memoria.

SPI fue desarrollado por Motorola en los años 80 bajo un esquema "maestro" – "esclavo" con un solo dispositivo "maestro" y capacidad para soportar varios dispositivos "esclavos".

SPI se basa en comunicación serie síncrona y permite comunicación full-duplex (comunicación entre dos dispositivos simultánea en ambos sentidos): utiliza líneas segregadas de señal de reloj (SCLK) y datos (MOSI, MISO), además de una línea específica de selección de dispositivo (SS). Es por esto que también se conoce como bus serie de 4 hilos.

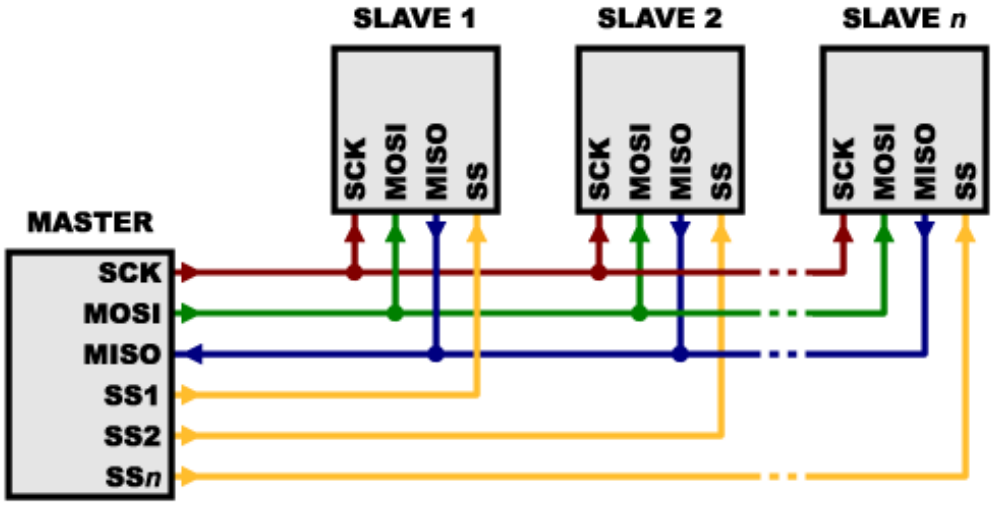

Gráfico 2. Bus SPI

La principal desventaja del bus I2C es que los sensores de un mismo tipo tienen predefinida una sola dirección (2 a lo más), por lo que se complica comunicar con este protocolo un conjunto amplio de sensores del mismo tipo (por ejemplo 6 sensores inerciales) Para solventar esta barrera, se puede optar por implementar un cambio de dirección por hardware en cada sensor o asignar secuencialmente mediante una señal la dirección "principal" a los diferentes sensores (asignando al resto de sensores la segunda dirección) e ir cambiando sucesivamente las asignaciones.

La principal desventaja del protocolo SPI es que se necesita cablear la señal de dirección, por lo que, al manejar un conjunto amplio de sensores, se precisa disponer del mismo número de señales en el controlador.

2.5.2 Comunicaciones controlador – servidor

Los controladores de bajo coste disponibles en el mercado se comunican de manera sencilla con un computador por medio de puerto serie bajo protocolo USB o por comunicación WiFi.

Por medio de comunicación WiFi es posible enlazar el controlador con un servicio en "la nube" a través de un punto de acceso a internet, de manera que el sistema entraría en la categoría de "Internet of Things" (IoT) y no precisaría de un computador local al que enviar la información para ser procesada.

2.6 Software

Se ha dividido el estudio del arte del software entre:

- Software a nivel controlador
- Software a nivel servidor

2.6.1 Software a nivel controlador

Entre las opciones para programar controladores de bajo coste, son muy populares las siguientes:

- Arduino
- Python (por medio de MicroPython o Zerynth)

Los controladores Arduino se programan desde un software específico, el entorno de desarrollo Arduino (Arduino IDE) que soporta código en lenguaje C/C++ con ciertos requisitos en la estructura del programa.
Desde el Arduino IDE se carga el programa a la memoria del controlador mediante el puerto serie.
Arduino IDE es de código abierto y está basado en Java, que a su vez está licenciado como "GNU General Public License".

MicroPython es una implementación del lenguaje de programación Python 3, más liviana, que está optimizada para su uso en microcontroladores y entornos con restricciones.
MicroPython ha sido desarrollado por Damien P. George y Paul Sokolovsky, mientras que su licencia es de tipo MIT license.

Zerynth es un entorno de programación y una máquina virtual que permite programar microcontroladores con el lenguaje Python o una hibridación de Python y C, con orientación a conectar dispositivos "en la nube". Ofrece compatibilidad con una amplia variedad de microcontroladores e integración con diferentes proveedores de infraestructura "en la nube".
El licenciamiento de Zerynth es privado y se centra en la máquina virtual (Zerynth Virtual Machine) que se carga en cada microcontrolador, disponiendo de dos versiones ("starter" y "premium"), estando sujeta su adquisición a un programa de precios.

2.6.2 Software a nivel servidor

Para la implementación de algoritmos a nivel servidor se dispone de multitud de lenguajes de programación, todos ellos con activas comunidades de desarrolladores, entre los que destacan:

- Python
- Java
- C++

Python es un lenguaje de programación multiplataforma, interpretado y de tipado dinámico. Este software fue desarrollado a finales de los 80 por Guido van Rossum en el CWI (Centrum Wiskunde & Informatica) de los Países Bajos. La primera versión de Python fue publicada a principios de los 90 y la última versión estable es Python 3.7.4 (julio 2019)
Python es administrado por la "Python Software Foundation" y cuenta con una numerosa y activa comunidad de desarrolladores. La licencia que rige sobre Python es de código abierto y compatible con "GNU General Public License".

Java es una máquina virtual ("Java Virtual Machine") y un lenguaje de programación multiplataforma, interpretado (compilado y ejecutado sobre JVM), de tipado estático. Java fue desarrollado por James Gosling en Sun Microsystems a principios de los 90. En 1996 se publica Java por primera vez y en la actualidad la última versión disponible es Java SE 12.
Con la adquisición de Sun Microsystems en 2010, Java pasó a ser propiedad de Oracle.
En 2007 Sun Microsystems licenció Java bajo "GNU General Public License".

C++ es un lenguaje de programación compilado de tipado estático que fue desarrollado por Bjarne Stroustrup como una extensión del lenguaje C a finales de los 70 en Bell Laboratories. La primera versión comercial de C++ se publica en 1985. Desde 1998 el lenguaje queda estandarizado por la Organización Internacional de Estandarización como ISO/IEC 14882:1998 y la última versión disponible es ISO/IEC 14882:2017 (conocida como C++17).

2.7 Representación gráfica

La representación gráfica en 3D exige la utilización de herramientas de programación específicas, siendo algunas de ellas las siguientes:

- VPython 7 / GlowScript
- Processing

VPython 7 es una librería del lenguaje de programación Python que facilita la creación de escenas en tres dimensiones, abriendo una ventana del navegador web en el que se visualizan las figuras. Existe una alternativa para programar escenas 3D sin necesidad de tener instalado Python en el equipo, a través de la aplicación Glowscript.

VPython ha sido desarrollado por David Scherer y Bruce Sherwood, estando protegido por Copyright, especificando el uso gratuito mediante la correspondiente mención.

Processing es un entorno y lenguaje de programación basado en Java y orientado a las artes visuales. El desarrollo de Processing comenzó en el 2001 por parte de Ben Fry y Casey Reas cuando eran estudiantes del MIT Media Lab.
Processing es gratuito, de código abierto y sujeto a licencia "GNU General Public License".

3 DISEÑO BÁSICO DEL SISTEMA

El objeto del proyecto es disponer de un sistema que represente gráficamente el movimiento de un objeto. Para conseguir esto se debe dar una serie de conversiones de manera consecutiva:

- Conversión de un evento físico (orientación o movimiento) a señales eléctricas
- Conversión de señales eléctricas a datos numéricos (valores de aceleración, velocidad angular)
- Conversión de valores numéricos a parametrización de la orientación (vector)
- Conversión de la parametrización de la orientación a una imagen

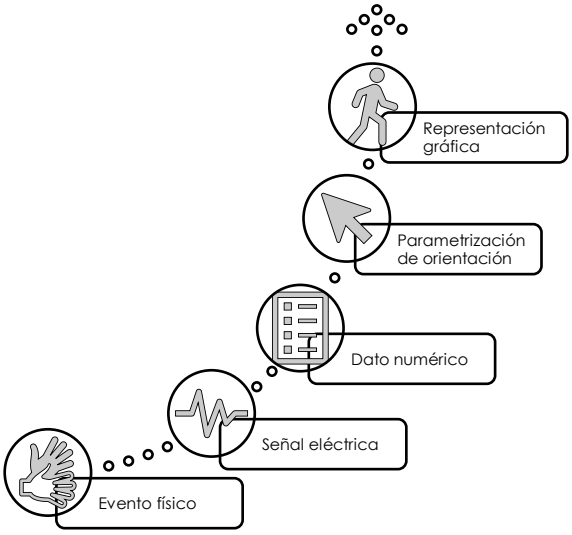

Gráfico 3. Conversiones en el proceso de captura del movimiento

Así, el sistema requiere los siguientes subsistemas:

- Capturador de movimiento
- Generador de orientación
- Generador de imagen

3.1 Subsistema capturador de movimiento

El subsistema capturador de movimiento consiste en un conjunto de sensores inerciales – que combinan un acelerómetro y un giroscopio – vinculados a objetos, que entregan el valor de la aceleración y la velocidad de giro en los tres ejes del espacio a un microcontrolador. El microcontrolador asume la comunicación con los sensores mediante I2C y con el subsistema generador de orientación mediante un puerto serie. La programación del microcontrolador se realiza en lenguaje Arduino.

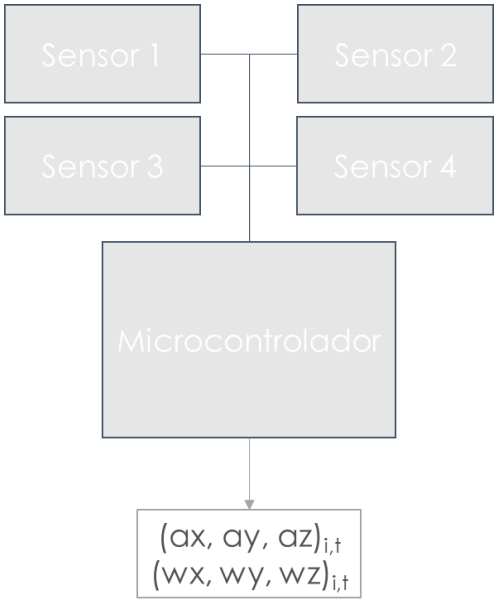

Gráfico 4. Subsistema capturador de movimiento - Esquema básico

Para cada instante de tiempo, el microcontrolador entrega a través del puerto serie 3 valores de aceleración y 3 valores de velocidad angular de cada sensor.

Los principales atributos de este subsistema son:

- Seis grados de libertad: son aquellos sensores que entregan información de aceleración y velocidad angular en los tres ejes del espacio. Se consiguen

ventajas significativas frente a sistemas de tres grados de libertad. En el apartado 2.3. Sensores se indican más detalles

- Comunicación I2C: este protocolo de comunicación permite integrar en un mismo bus varios sensores con una señal de datos y otra de reloj. Existe una limitación al uso de más de dos sensores idénticos en el mismo bus, ya que cada tipo de sensor tiene una dirección I2C predeterminada y la posibilidad de activar una segunda dirección I2C mediante la modificación del bit menos significativo (accesible en un pin del módulo). Para poder utilizar un tercer sensor en el mismo bus I2C se recurre a un sistema de turnos, en el que el microcontrolador asigna la dirección I2C "secundaria" a los sensores que deben permanecer sin comunicación, reservando la dirección I2C "principal" al sensor con el que se quiere comunicar. Esto exige cablear el pin del bit menos significativo de cada sensor con una salida digital en el controlador, resultando una suerte de solución mixta I2C/SPI. En el apartado 2.5.1. Comunicaciones sensor – controlador se indican más detalles.

3.2 Subsistema generador de orientación

El subsistema generador de orientación es un algoritmo escrito en lenguaje Python, instalado en un computador que toma desde el puerto serie los datos de los sensores (recolectados y enviados por el microcontrolador) y genera una estimación de la orientación en forma de cuaternión para cada uno de los objetos vinculados a cada sensor.

El fundamento teórico utilizado para la estimación de la orientación está descrito en el artículo "Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs" de Roberto G. Valenti, Ivan Dryanovski y Jizhong Xiao. En el apartado 4. Fundamentos teóricos, se pueden encontrar más detalles sobre el algoritmo implementado.

Las principales características del algoritmo son:

- Utilización de cuaterniones para la estimación de la orientación basada en datos de aceleración y velocidad angular. Así se evitan las indeterminaciones que implica el uso de ángulos de Euler o la carga computacional del uso de matrices.
- Enfoque determinista (frente al enfoque de optimización)
- Utilización de un filtro complementario para la fusión de datos de aceleración y velocidad angular. Este tipo de filtro es conceptual y computacionalmente más sencillo que otras alternativas habituales para tratar esta problemática, como puede ser un filtro de Kalman

3.3 Subsistema generador de imagen

El subsistema generador de imagen es un algoritmo escrito en VPython (más información en el apartado 2.7. Representación gráfica) que genera una visualización en tres dimensiones de los objetos vinculados a los sensores, en base a las estimaciones de orientación entregadas por el subsistema generador de orientación.

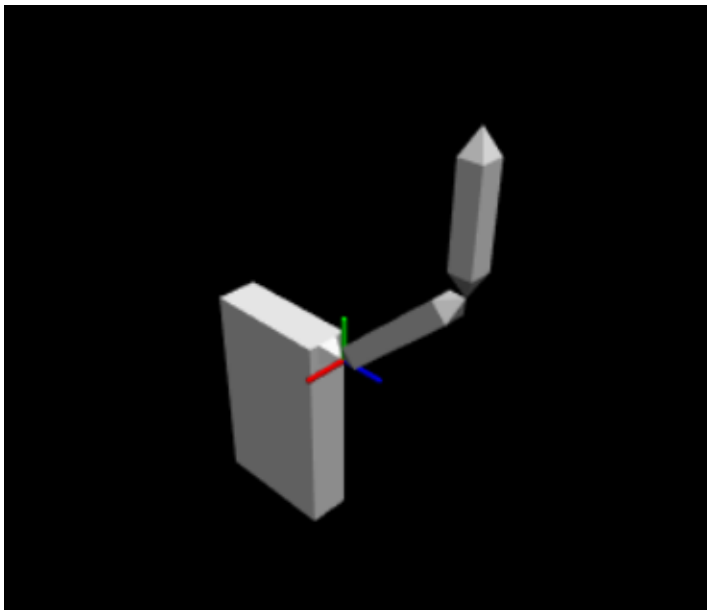

Gráfico 5. Muestra de salida gráfica con VPython

A efectos demostrativos, se modela la representación gráfica de un torso y un brazo izquierdo con elementos prismáticos. El sistema de referencia está centrado en el hombro del modelo y los vectores toman coordenadas cartesianas.
La orientación de cada uno de los elementos viene determinada por un vector alineado al eje longitudinal del prisma y un segundo vector, perpendicular al primero, que determina el giro sobre dicho eje longitudinal.

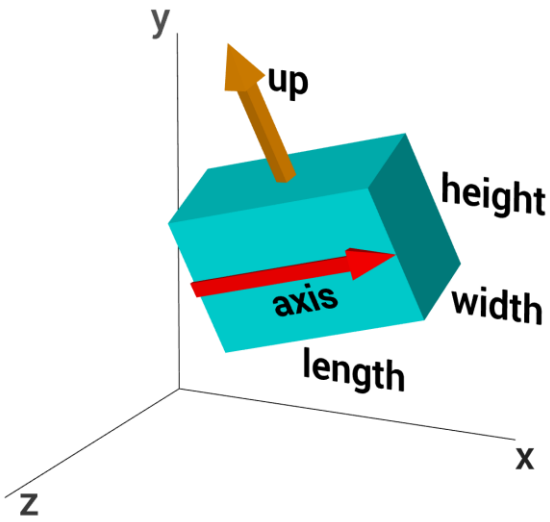

Gráfico 6. Elemento "box" de VPython

La posición de cada uno de los elementos viene dada por la restricción que implica el concepto de articulación como punto común a dos elementos. Así el extremo distal de un elemento coincide con el proximal del siguiente.

4 FUNDAMENTOS TEÓRICOS

4.1 Sistemas de referencia

Para afrontar la determinación de la orientación de un objeto en un espacio tridimensional es necesario establecer los sistemas de referencia. Tomando el caso más amplio, las caracterizaciones más habituales son:

- Sistema de referencia del objeto
- Sistema de referencia de navegación
- Sistema de referencia de la Tierra
- Sistema de referencia inercial

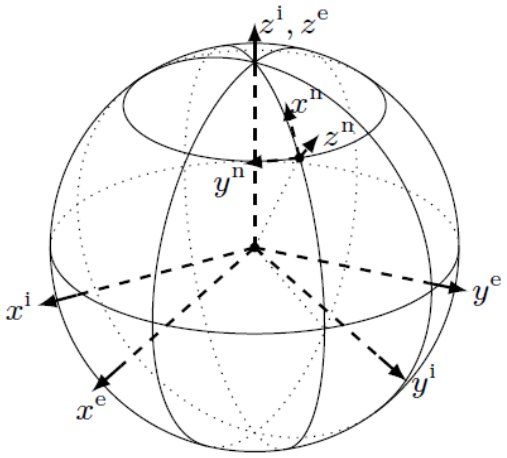

Gráfico 7. Sistemas de referencia de navegación (n), de la Tierra (e) e inercial (i)

El sistema de referencia del objeto ("body frame") es el que tiene su origen en el centro del sensor y está alineado con su encapsulado. En base a este sistema de referencia se entregan los resultados de aceleración, velocidad angular o campo magnético de los sensores.

El sistema de referencia de navegación ("navigation frame") es un sistema de coordenadas local respecto al cual se da el movimiento. En muchas aplicaciones, especialmente aquellas con desplazamientos limitados como el caso que ocupa este proyecto, este sistema de referencia se considera fijo respecto al de la Tierra. Sin embargo, aquellos casos en los que se dan grandes desplazamientos exigen que el sistema de referencia de navegación se mueva por la superficie terrestre.

El sistema de referencia de la Tierra ("earth frame") gira con el movimiento de rotación terrestre, tiene su origen en el centro de la Tierra y los ejes fijados por posiciones terrestres.

El sistema de referencia inercial ("inertial frame") es un sistema fijo. Los sensores miden aceleración y velocidad angular respecto a este sistema, que tiene su origen en el centro de la Tierra y sus ejes alineados con posiciones de estrellas.

Se puede decir que el sistema del objeto se mueve sobre el sistema de navegación, que a su vez se desplaza sobre el sistema de la Tierra y éste sobre el sistema inercial.

4.2 Sensores inerciales

Los sensores inerciales que se consideran en este proyecto constan de un acelerómetro y un giroscopio de tres ejes cada uno.
Al experimentar las magnitudes físicas (velocidad angular y fuerza específica) los sensores generan a una señal de voltaje que es convertida a un valor de medición, basado en un proceso de calibración realizado en fábrica.
En todo caso, los sensores deben ser objeto de un proceso de corrección de errores al ser incorporados al sistema.

4.2.1 Magnitudes medidas por giroscopio y acelerómetro

El giroscopio mide la velocidad angular del sistema de referencia del objeto respecto al sistema de referencia inercial (fijo) en términos del sistema de referencia del objeto (w_{ib}^{b}). Esto se puede expresar a través de la siguiente ecuación:

$$w_{ib}^{b} = R^{bn}(w_{ie}^{n} + w_{en}^{n}) + w_{nb}^{b}$$

Donde:

- w_{nb}^{b} es la velocidad angular del sistema de referencia del objeto respecto al sistema de referencia de navegación, en términos del sistema de referencia del objeto
- w_{en}^{n} es la velocidad angular del sistema de referencia de navegación respecto al sistema de referencia de la Tierra, en términos del sistema de referencia de navegación
- w_{ie}^{n} es la velocidad angular del sistema de referencia de la Tierra respecto al sistema de referencia inercial, en términos del sistema de referencia de navegación
- R^{bn} es la matriz de rotación del sistema de referencia de navegación al sistema de referencia del objeto

Por tanto, en la medición del giroscopio se incluye la velocidad angular del sistema de coordenadas de la Tierra respecto al inercial ("earth rate"), definido por la velocidad de rotación de la Tierra, que tiene un valor aproximado de $7{,}29 \cdot 10^{-5}$ rad/s.

En el caso de que el sistema de navegación no sea estático respecto a la Tierra, también se estaría midiendo la correspondiente velocidad angular ("transport rate").

El acelerómetro mide la fuerza específica en el sistema de referencia del objeto (f^{b}).

$$f^b = R^{bn}(a_{ii}^n - g^n)$$

Donde:

- R^{bn} es la matriz de rotación del sistema de referencia de navegación al sistema de referencia del objeto
- a_{ii}^n es la aceleración lineal del sensor en términos del sistema de coordenadas de navegación, que contiene la aceleración centrífuga y la aceleración de Coriolis
- g^n es el vector de gravedad

Cabe mencionar que la aceleración centrífuga depende de la velocidad de rotación de la Tierra y la ubicación del objeto sobre la superficie terrestre, siendo su magnitud menor que $3{,}39 \cdot 10^{-2}$ m/s^2. La aceleración centrífuga suele considerarse como absorbida por la gravedad.

La aceleración de Coriolis depende de la velocidad de rotación de la Tierra y de la velocidad del objeto, tomando valores pequeños en los casos de movimientos humanos. Por ejemplo, una persona caminando a 5 km/h experimenta una aceleración de Coriolis de $2{,}03 \cdot 10^{-4}$ m/s^2.

4.2.2 Error en los sensores inerciales

Si bien los giroscopios y los acelerómetros miden velocidad angular y fuerza específica, el valor que entregan no es una representación exacta de la realidad. Esto se debe a los siguientes efectos:

- Ruido
- Offset
- Deriva ("Time-varying bias")

La señal que entregan los sensores está afectada por el ruido. Lo que se obtiene al analizar las diferentes señales de los sensores es una distribución pseudo-gaussiana. Tomando un sensor en posición estacionaria se observa que el valor que entrega durante un periodo de 1000 muestras no es constante:

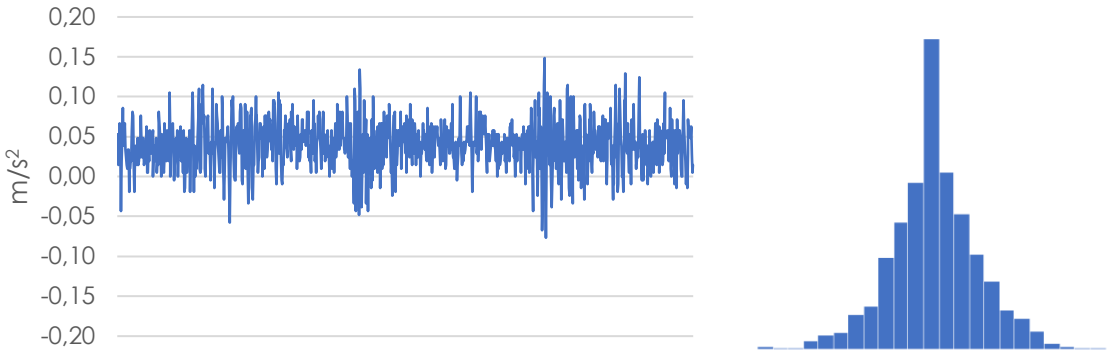

Gráfico 8. Aceleración y su distribución en el eje x

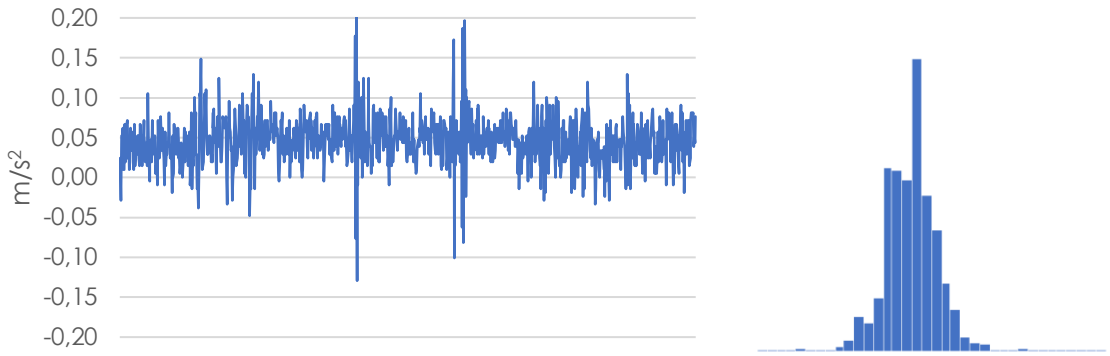

Gráfico 9. Aceleración y su distribución en el eje y

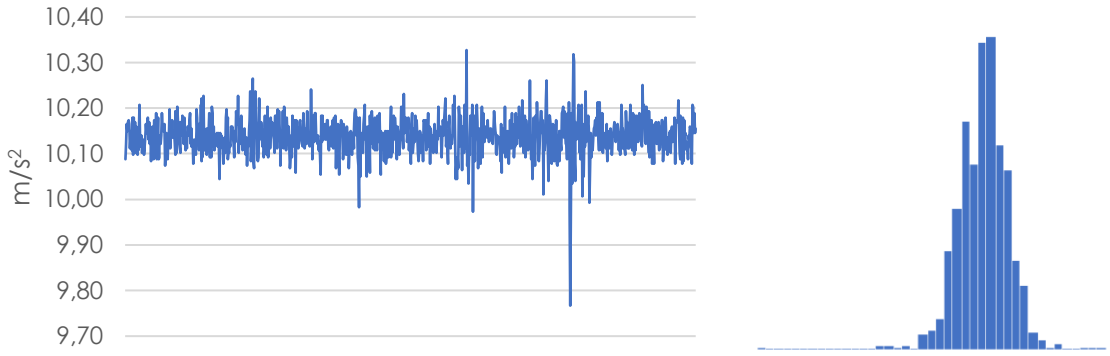

Gráfico 10. Aceleración y su distribución en el eje z

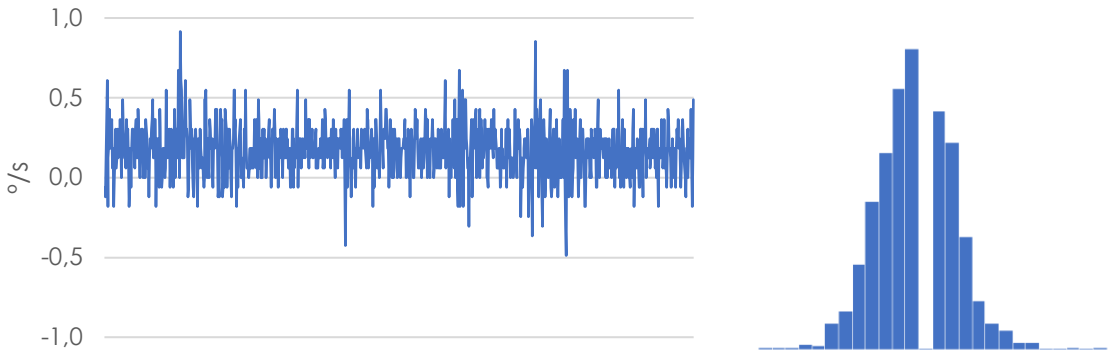

Gráfico 11. Velocidad angular y su distribución en el eje x

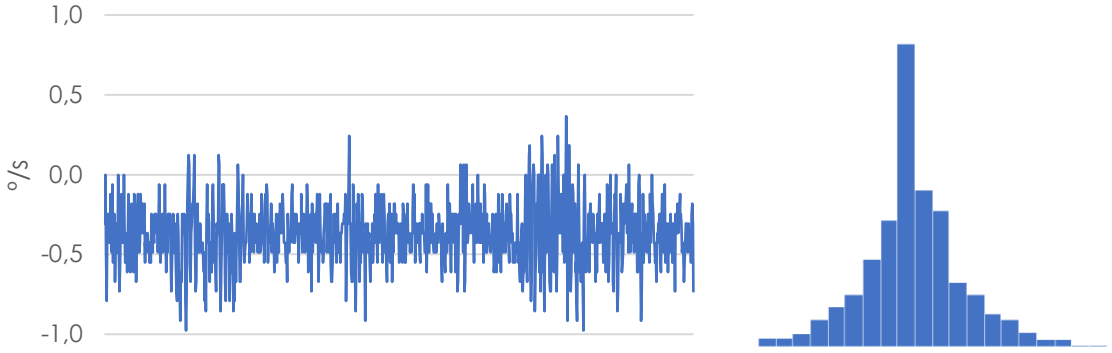

Gráfico 12. Velocidad angular y su distribución en el eje y

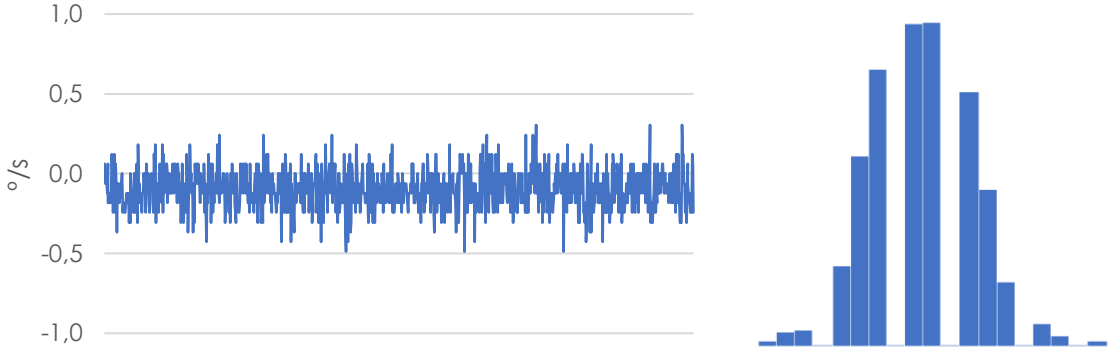

Gráfico 13. Velocidad angular y su distribución en el eje z

En los gráficos anteriores se evidencia que las señales medidas tiene ruido y que no están centradas (presentan un "offset") en el valor que se espera para un sensor estacionario y alineado con los ejes:

Tabla 7. Ejemplo de offset en las mediciones de los sensores

	ax m/s^2	**ay** m/s^2	**az** m/s^2	**gx** °/s	**gy** °/s	**gz** °/s
Promedio	0,0401	0,0458	10,1388	0,1716	-0,3696	-0,0971
Referencia	0	0	9,81	0	0	0
Offset	-0,0401	-0,0458	-0,3288	-0,1716	0,3696	0,0971

Finalmente, se ha de mencionar el efecto de deriva que se da en los giroscopios, cuya medición va variando lentamente con el tiempo aun estando en posición estacionaria.

4.3 Parametrización de la orientación

Dado un objeto en un espacio tridimensional, se puede determinar su orientación indicando una serie de parámetros. Esto es lo que se conoce como parametrización de la orientación.

Las cuatro maneras más utilizadas para parametrizar la orientación de un objeto son:

- Matriz de rotación
- Vector de rotación
- Ángulos de Euler
- Cuaterniones unitarios

4.3.1 Matriz de rotación

Dados dos sistemas de coordenadas "a" y "b", un vector "x" respecto a un sistema de coordenadas "a" se relaciona a través de la correspondiente matriz de rotación "R" con el mismo vector respecto al sistema de coordenadas "b". Esto es:

$$x^a = R^{ab}x^b$$

Las matrices de rotación, "R", tienen las siguientes propiedades:

$$RR^T = R^TR = I_3$$

$$\det R = 1$$

De modo que la matriz de rotación de "a" a "b" es la transpuesta de la matriz de rotación de "b" a "a":

$$R^{ba} = \left(R^{ab}\right)^T$$

$$x^b = \left(R^{ab}\right)^T x^a = R^{ba}x^a$$

La matriz de rotación es una descripción única de la orientación y, en el caso del espacio tridimensional, tiene 9 componentes.

4.3.2 Vector de rotación

De acuerdo con el teorema de rotación de Euler, cualquier movimiento de un sólido rígido que mantenga un punto constante, también debe dejar constante un eje completo. Así, se puede expresar la rotación entre dos sistemas de coordenadas en términos de un ángulo "α" y un vector unitario "n", sobre el que se produce la rotación.

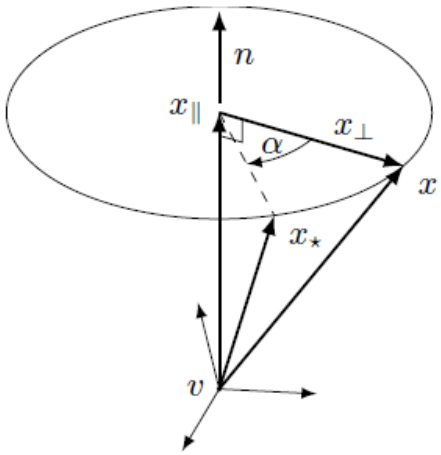

Gráfico 14. Parametrización por vector de rotación

Así, de acuerdo con el gráfico anterior, se puede expresar x_*^v en términos de x^v, n^v y α:

$$x_*^v = x^v \cos\alpha + n^v(x^v \cdot n^v)(1 - \cos\alpha) - (n^v \times x^v)\sin\alpha$$

El vector de rotación permite parametrizar la orientación con tres parámetros, pero no ofrece una solución única, ya que cualquier ángulo al que se le sume 2π resulta ser la misma orientación (fenómeno de envoltura o "wrapping").

La matriz de rotación entre los sistemas de coordenadas "u" y "v" puede ser expresada en términos de n^v y α:

$$R^{uv} = I_3 - \sin\alpha\,[n^v \times] + (1 - \cos\alpha)[n^v \times]^2$$

4.3.3 Ángulos de Euler

Una rotación puede ser definida como rotaciones consecutivas en torno a tres ejes, lo que se conoce como ángulos de Euler.

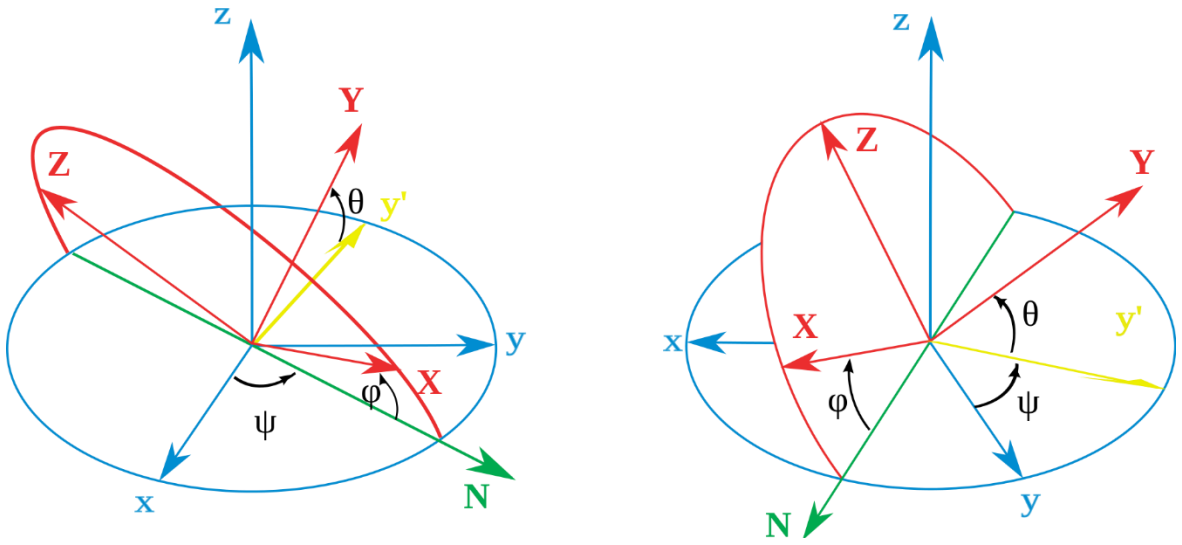

Gráfico 15. Representación de los ángulos de Euler (secuencia zxy)

La representación por ángulos de Euler utiliza 3 parámetros, pero no es única por dos efectos:

- Envoltura o "wrapping": por ejemplo, la rotación (0, 0, 0) es igual que (0, 0, 2π)
- Bloqueo de cardán o "gimbal lock": para el caso de θ=π/2, cualquier giro (φ, ψ) que tenga el mismo valor de φ - ψ, representa la misma orientación.

Para cada secuencia, existe una matriz de rotación que se puede expresar en términos de los ángulos de Euler.

4.3.4 Cuaterniones unitarios

Cualquier orientación en el espacio tridimensional de un sistema de referencia "a" respecto a otro "b" puede ser representada por un cuaternión unitario descrito de la siguiente manera:

$$q_{ab} = [q_0, q_1, q_2, q_3]^T = \left[\cos\frac{\alpha}{2}, e_x \sin\frac{\alpha}{2}, e_y \sin\frac{\alpha}{2}, e_z \sin\frac{\alpha}{2}\right]^T$$

Donde α es el ángulo de rotación y e es el vector unitario que representa al eje de rotación.

El conjugado del cuaternión es igual a la inversa del cuaternión y describe la rotación inversa.

$$q_{ba} = q_{ab}^{*} = [q_0, -q_1, -q_2, -q_3]^T$$

Los cuaterniones se utilizan para obtener la rotación de un vector:

$$^{b}_{q}v = q_{ab} \otimes {}^{a}_{q}v \otimes q_{ab}^{*}$$

Donde ⊗ indica el producto de cuaterniones y $^{a}_{q}v$ es un vector en términos del sistema de referencia "a" y expresado como cuaternión: $[0\ v_x\ v_y\ v_z]^T$.

Se puede obtener una matriz de rotación en términos de cuaternión unitario:

$$R_{ab} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

Por lo que:

$$^{b}_{q}v = R_{ab}\,{}^{a}_{q}v$$

La representación en forma de cuaternión utiliza 4 parámetros y no es única, ya que tanto q como -q representan la misma orientación.

4.4 Álgebra de cuaterniones

Los cuaterniones fueron inventados por William Rowan Hamilton en 1843. Se trata de una extensión de los números complejos, con dos dimensiones imaginarias adicionales.

Un cuaternión está compuesto por cuatro números escalares, que representan una dimensión real y tres dimensiones imaginarias. Cada dimensión imaginaria tiene por valor unitario $\sqrt{-1}$ – tomando la notación i, j y k – mutuamente perpendiculares.

Así un cuaternión, q, se representa de la siguiente manera:

$$q = a + ib + jc + kd$$

Otra representación habitual de los cuaterniones es la de un par escalar-vector:

$$q = s + v$$

Donde s es un escalar y v un vector tridimensional.

Los cuaterniones se pueden sumar y multiplicar de forma similar al álgebra escalar, sin embargo, entre cuaterniones no se cumple la propiedad conmutativa del producto.

Tabla 8. Leyes del álgebra de cuaterniones

	Operación	
	Suma	**Producto**
Elemento neutro	0	1
Propiedad conmutativa	Si	No
Propiedad asociativa	Si	Si
Propiedad distributiva	-	Si
Operación inversa	Si	Si

La suma de cuaterniones se realiza sumando los términos reales y los imaginarios de forma independiente. La resta es la operación inversa de la suma y se considera como la suma de elementos negativos.

$$q_A + q_B = (a + ib + jc + kd) + (e + if + jg + kh)$$
$$= (a + e) + i(b + f) + j(c + g) + k(d + h)$$

El producto de cuaterniones debe considerar las reglas que aplican al producto de los términos imaginarios, tal y como se aprecia en la siguiente tabla:

Tabla 9. Producto de cuaterniones

a⊗b	**b_1**	**b_i**	**b_j**	**b_k**
a_1	1	i	j	k
a_i	i	-1	k	-j
a_j	j	-k	-1	I
a_k	k	j	-i	-1

Generalizando el producto de dos cuaterniones, se tiene:

$$q_A \otimes q_B = (a + ib + jc + kd) + (e + if + jg + kh)$$
$$= (a * e - b * f - c * g - d * h) + i(b * e + a * f + c * h - d * g)$$
$$+ j(a * g - b * h + c * e + d * f) + k(a * h + b * g - c * f + d * e)$$

Para la división se considera la operación de producto por el inverso. En el caso de cuaterniones unitarios, el inverso es igual al conjugado:

$$q^{-1} = conj(q) = a - ib - jc - kd$$

Generalizando la situación a cuaterniones no unitarios:

$$q^{-1} = \frac{conj(q)}{|q^2|} = \frac{a - ib - jc - kd}{a^2 + b^2 + c^2 + d^2}$$

La norma de un cuaternión se obtiene de la siguiente manera:

$$\|q\| = \sqrt{q \otimes conj(q)} = \sqrt{a^2 + b^2 + c^2 + d^2}$$

4.5 Filtro complementario para combinación de señales

4.5.1 Antecedentes: determinación de la orientación

En 1965, la matemática Grace Whaba propuso el conocido como "problema de Whaba", que busca encontrar una matriz de rotación entre dos sistemas de coordenadas dado un conjunto de observaciones. La solución a este problema matemático estaba destinada a la determinación de la actitud de satélites.

Se han planteado multitud de soluciones algorítmicas al problema de Whaba, los cuales se pueden clasificar en dos categorías:

- Deterministas (mediante resolución de ecuaciones)

- De optimización (mediante minimización de funciones)

Para obtener una mejor estimación de la orientación, se han de combinar los datos procedentes del giroscopio (velocidad angular) y del acelerómetro (aceleración lineal). Para lograr esta fusión de datos, las técnicas más habituales se basan en:

- Filtro Extendido de Kalman (enfoque probabilístico)
- Filtro Complementario (mediante análisis en el dominio de frecuencia)

El filtro complementario se comporta como filtro paso-alto para los datos del giroscopio (afectados por ruido de baja frecuencia) y como filtro paso-bajo para los datos del acelerómetro (afectados por ruido de alta frecuencia).
El filtro complementario es más sencillo y menos exigente a nivel computacional que el filtro extendido de Kalman.

La mayoría de los algoritmos recientes para la fusión de datos de sensores inerciales toman la estimación de la orientación en forma de cuaterniones, por sus ventajas frente al uso de otras herramientas matemáticas como matrices de rotación, que requieren más carga computacional, o ángulos de Euler, que sufren de configuraciones singulares en las que no se alcanza solución única.

4.5.2 Descripción del algoritmo implementado

En este proyecto se implementa un algoritmo desarrollado por Roberto G. Valenti, Ivan Dryanovski y Jizhong Xiaoen en el artículo "Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs".
Este algoritmo es de tipo determinista y toma como datos de entrada los valores de aceleración lineal y velocidad angular. Los datos de giroscopio y acelerómetro se combinan mediante un filtro complementario y se toma la orientación en forma de cuaterniones.

El algoritmo de combinación de datos consta de dos pasos:

- Predicción (se obtiene el cuaternión de orientación basado únicamente en los valores de velocidad angular, q_ω)
- Corrección (se obtiene la corrección, escalada y normalizada, basada en datos de acelerómetro, $\widehat{\Delta q_{acc}}$)

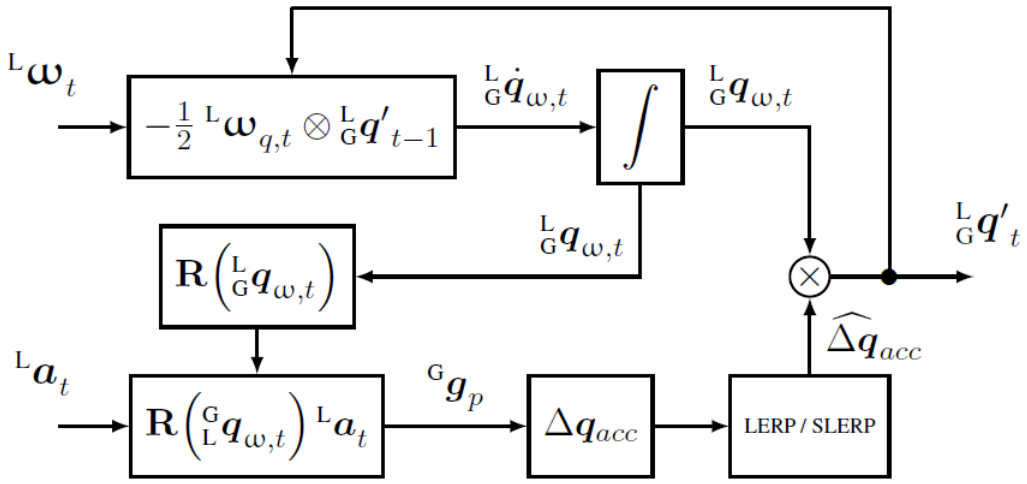

Gráfico 16. Diagrama de bloques del filtro complementario implementado

Predicción

En el paso de predicción, el vector de velocidad angular, medido por el giroscopio, se usa para calcular una primera estimación de la orientación en forma de cuaternión. Conocidas las condiciones iniciales, en primer lugar, se calcula la derivada del cuaternión, que indica la tasa de cambio de la orientación:

$$ {}^{L}_{G}\dot{\boldsymbol{q}}_{\omega,t_k} = -\frac{1}{2}\,{}^{L}\boldsymbol{\omega}_{q,t_k} \otimes {}^{L}_{G}\boldsymbol{q}_{t_{k-1}} $$

La orientación del sistema de coordenadas global respecto al local en el instante t_k se calcula por integración numérica, considerando como periodo de muestreo $\Delta t = t_k - t_{k-1}$:

$$ {}^{L}_{G}\boldsymbol{q}_{\omega,t_k} = {}^{L}_{G}\boldsymbol{q}_{t_{k-1}} + {}^{L}_{G}\dot{\boldsymbol{q}}_{\omega,t_k}\Delta t $$

Corrección

Se toma el resultado de la predicción (q_ω, del paso anterior) y los valores de aceleración lineal entregados por el acelerómetro (a), para obtener la predicción del vector de la gravedad (g_p):

$$\mathbf{R}\left({}^{\mathrm{G}}_{\mathrm{L}}\boldsymbol{q}_{\omega}\right){}^{\mathrm{L}}\boldsymbol{a} = {}^{\mathrm{G}}\boldsymbol{g}_{p}$$

La predicción del vector de la gravedad ($[g_x\ g_y\ g_z]^T$) y el vector de la gravedad real ($[0\ 0\ 1]^T$) tendrán una pequeña diferencia, por lo que se obtiene el delta-cuaternión (Δq_{acc}), que define el giro entre ambos vectores, resolviendo el sistema de ecuaciones correspondiente. Finalmente:

$$\Delta\boldsymbol{q}_{acc} = \begin{bmatrix} \sqrt{\frac{g_z+1}{2}} & -\frac{g_y}{\sqrt{2(g_z+1)}} & \frac{g_x}{\sqrt{2(g_z+1)}} & 0 \end{bmatrix}^{\mathrm{T}}$$

Dado que el delta-cuaternión se ve afectado por el ruido de alta frecuencia del acelerómetro, éste se debe escalar mediante interpolación con el cuaternión unitario ($q_I = [1\ 0\ 0\ 0]^T$), para posteriormente normalizarlo.

Se toman dos métodos de interpolación en función del valor del primer término del delta-cuaternión ($\Delta q0_{acc} = \varepsilon$):

- LERP ("Linear Interpolation") para $\varepsilon > 0{,}9$

$$\overline{\Delta\boldsymbol{q}}_{acc} = (1-\alpha)\boldsymbol{q}_I + \alpha\Delta\boldsymbol{q}_{acc}$$

$$\widehat{\Delta\boldsymbol{q}}_{acc} = \frac{\overline{\Delta\boldsymbol{q}}_{acc}}{\|\overline{\Delta\boldsymbol{q}}_{acc}\|}$$

- SLERP ("Spherical Linear Interpolation") para $\varepsilon \leq 0{,}9$

$$\widehat{\Delta\boldsymbol{q}}_{acc} = \frac{\sin([1-\alpha]\Omega)}{\sin\Omega}\boldsymbol{q}_I + \frac{\sin(\alpha\Omega)}{\sin\Omega}\Delta\boldsymbol{q}_{acc}$$

Considerando que:

- La ganancia, α, es un valor entre 0 y 1 que caracteriza la frecuencia de corte del filtro
- Cos $\Omega = \varepsilon$

Resultado final

La orientación (q) que se obtiene como resultado del algoritmo es el producto de los cuaterniones de predicción basada en datos de giroscopio (q_ω) y corrección basada en datos de acelerómetro ($\widehat{\Delta q_{acc}}$):

$$^{L}_{G}\boldsymbol{q}' = {}^{L}_{G}\boldsymbol{q}_{\omega} \otimes \widehat{\Delta \boldsymbol{q}}_{acc}$$

5 DESARROLLO DEL SISTEMA FÍSICO (HARDWARE)

5.1 Dispositivos

Para la construcción física del sistema de captura de movimiento se ha utilizado una placa Arduino UNO y tres sensores inerciales basados en MPU 6050.

Los dos criterios que justifican el uso de dichos dispositivos son:

- Cumplimiento de especificaciones básicas
- Disponibilidad
- Costo

Tanto la placa Arduino UNO como los sensores inerciales MPU 6050 están disponibles sin problemas en los comercios minoristas especializados en electrónica en Santiago (Chile). En concreto, los sensores han sido adquiridos por un valor unitario de 3.690 pesos chilenos (unos 4,8 €) y la placa compatible Arduino UNO R3 por 4.290 pesos (unos 5,6 €).

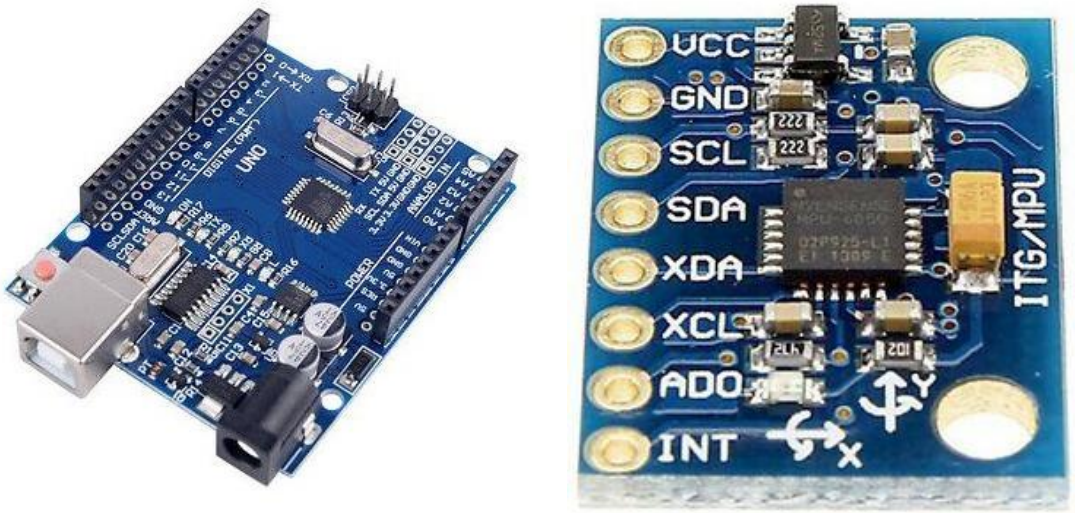

Ilustración 3. Placas del microcontrolador y sensor inercial

5.1.1 Microcontrolador

Tabla 10. Especificaciones del microcontrolador

Fabricante	Arduino
Modelo	UNO R3
Procesador	ATmega328P (8 bits)
Tensión nominal	5 V
Alimentación	7-12 V
E/S digital	14 (6 PWM)
Entradas analógicas	6
Corriente pin E/S	20 mA
Velocidad	16 MHz
Memoria	32 KB
Comunicación	SPI / I2C
Largo	68,6 mm
Ancho	53,4 mm
Peso	25 g

5.1.2 Sensores inerciales

Tabla 11. Especificaciones de los sensores inerciales

Fabricante	Inven Sense
Modelo	MPU 6050
Rango Giroscopio	±250/±2000 °/seg
Conversor AD Giroscopio	16 bits
Rango Acelerómetro	±2 / ±16 g
Conversor AD Acelerómetro	16 bits
Buffer	1024 byte FIFO

Alimentación	2,37-3,46 V
Corriente nominal	3,9 mA
Largo	21,2 mm
Ancho	16,4 mm
Peso	2 g

5.2 Esquema de conexiones

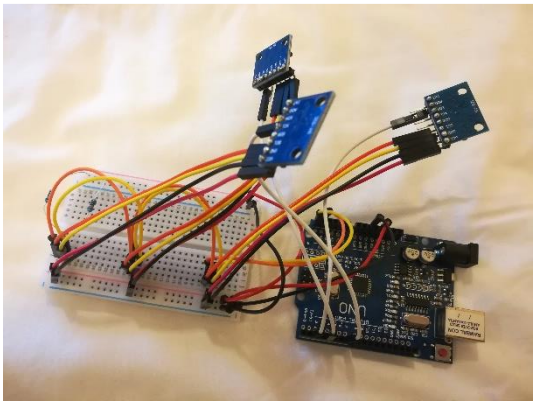

Ilustración 4. Montaje funcional del sistema

Tabla 12. Correspondencia de conexión

Arduino UNO	Sensor 1	Sensor 2	Sensor 3
3.3V	VCC	VCC	VCC
GND	GND	GND	GND
A5	SCL	SCL	SCL
A4	SCA	SCA	SCA
2	ADO	-	-
4	-	ADO	-
7	-	-	ADO

6 DESARROLLO DEL SISTEMA LÓGICO (SOFTWARE)

6.1 Código del subsistema capturador de movimiento

El programa que ejecuta el microcontrolador para capturar los datos que entregan los sensores viene determinado por el siguiente diagrama de flujo:

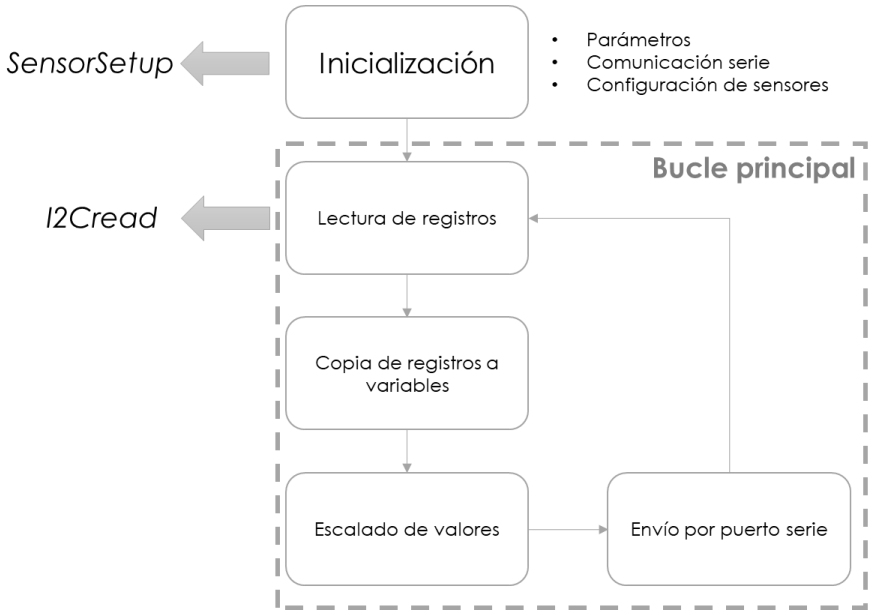

Gráfico 17. Flujo básico del subsistema capturador de movimiento

Se utiliza la librería "Wire" del entorno Ardunio para la comunicación vía I2C.

La inicialización afecta a los parámetros del puerto serie (velocidad de la comunicación) y de configuración de los sensores (fondo de escala del acelerómetro y del giroscopio)

Se han escrito dos funciones sencillas (I2Cread y SensorSetup) para realizar la lectura y escritura de los registros de los sensores por I2C.

Dentro del bucle principal se lleva a cabo una lectura consecutiva de los registros que contienen los datos de aceleración lineal y velocidad angular de los sensores que forman parte del sistema. Los valores crudos de los registros se almacenan en variables intermedias, completando 14 bytes por cada sensor, 12 bytes útiles, que corresponden a 2 bytes por cada uno de los 6 grados de libertad (aceleración y velocidad angular en cada uno de los tres ejes), más 2 bytes del valor de temperatura que no se utiliza en este caso.

Se dispone así de una copia de los valores correspondientes a las magnitudes entregadas por los sensores en variables independientes (6 por sensor).

En relación al fondo de escala seleccionado durante la inicialización, se realiza un escalado de los valores contenidos en las variables para disponer de las mediciones en las unidades convenientes (m/s^2 y °/s).

Finalmente, se envían los datos a través del puerto serie, insertando los correspondientes caracteres de separación de datos y final de la comunicación.

6.2 Código del subsistema generador de orientación

El programa que maneja el subsistema generador de orientación (también integra la inicialización y actualización de la representación gráfica) sigue un flujo que se muestra en el siguiente gráfico.

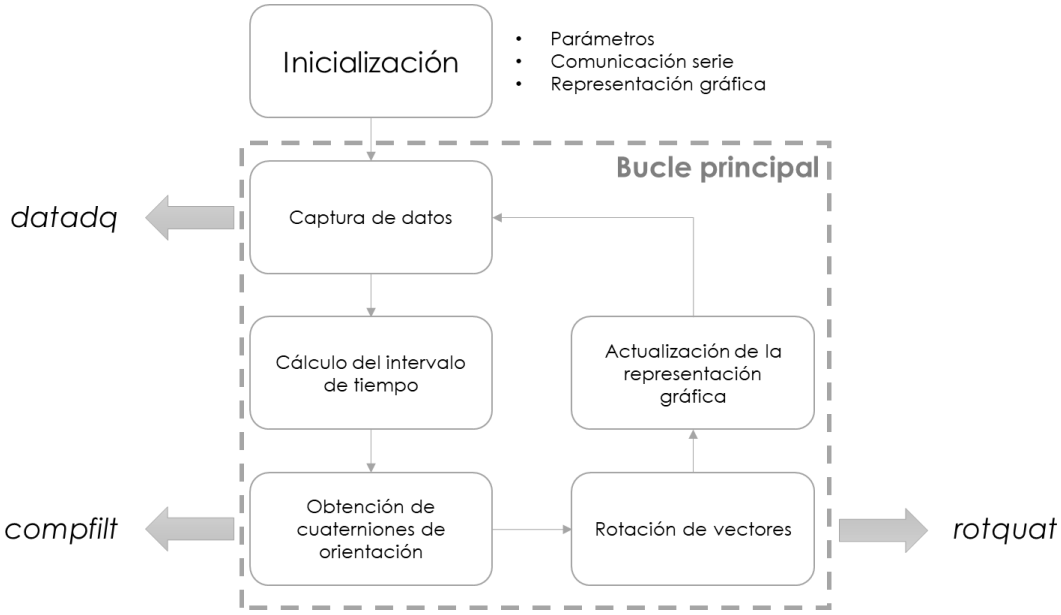

Gráfico 18. Flujo básico del subsistema generador de orientación

El código del programa utiliza una serie de módulos disponibles para Python que permiten incorporar funcionalidades más allá de las que contiene por defecto el lenguaje. Son las siguientes:

- Serial, para la comunicación serie con el microcontrolador
- Time, para manejo de las etiquetas de tiempo
- Math, para funciones matemáticas básicas
- Numpy, para manejo de vectores y matrices
- Vpython, para la representación gráfica (más detalles en 6.3)
- Quaternion, para las operaciones algebraicas con cuaterniones

La inicialización del programa afecta a los siguientes parámetros:

- Definición de los ejes x, y, z del sistema de coordenadas (ini_x, ini_y, ini_z)
- Etiqueta de origen del tiempo (t0)
- Factor de conversión de grados a radianes (k)

Como parte de la inicialización, se abre la comunicación serie y se activa la representación gráfica.

Se han escrito las siguientes funciones:

- Captura de datos del puerto serie (datadq)
- Obtención de matriz de rotación y conversión a cuaternión unitario (rotmtrx)
- Obtención de cuaterniones de orientación (compfilt)
- Construcción de la Direct Cosine Matrix (dcm)
- Rotación de vectores usando cuaterniones (rotquat)

La nomenclatura de los parámetros que se manejan en las funciones queda especificada en los siguientes apartados.

Dentro del bucle principal se suceden los siguientes pasos en forma de ciclo continuo:

- Captura de datos (llamada a la función datadq) de los sensores que se envían por el puerto serie
- Cálculo del periodo de muestreo
- Obtención de cuaterniones de orientación (llamada a la función compfilt) en base a los valores de aceleración y velocidad angular capturados, y a la orientación previa
- Rotación de vectores (llamada a la función rotquat) que representan los ejes del sistema de coordenadas de cada sensor
- Actualización de la representación gráfica, reflejando la orientación de cada uno de los objetos a los que están vinculados los sensores

Cabe destacar que el algoritmo de determinación del cuaternión de orientación para un instante t toma como valores de entrada:

- las aceleraciones y velocidades angulares en el instante t, medidos por los sensores
- el periodo de muestreo, calculado por diferencia entre la etiqueta de tiempo en el instante t y el t-1 (en primera iteración t0, origen del tiempo determinado en la inicialización del programa)

- el cuaternión de orientación del instante t-1, por lo que se ha de predefinir dicho cuaternión, lo que no es inmediato

El modo elegido para la obtención del cuaternión de orientación correspondiente al origen del tiempo es mediante la obtención de la matriz de rotación a partir de los valores de aceleración en t0 y su conversión a cuaternión unitario. Este cálculo se ha implementado en la función rotmtrx, a la que solo se hace una llamada al inicio del bucle.

6.2.1 Función para la captura de datos del puerto serie (datadq)

Se utiliza la función datadq para la captura de los datos de los sensores que se reciben a través del puerto serie.

$$wx, wy, wz, ax, ay, az, t1 = datadq()$$

Tabla 13. Datos de entrada de la función datadq

Datos de entrada	
-	La función no requiere datos de entrada

Tabla 14. Cálculos intermedios en la función datadq

Cálculos intermedios	
rawstring	Cadena de caracteres recibida a través del puerto serie
IMUdata	Estructura de datos que contiene los valores entregados por los sensores

Tabla 15. Resultado de la función datadq

Resultado	
wx, wy, wz	Valores de velocidad angular (en rad/s) en los tres ejes del espacio, capturados del giroscopio, en el instante t

ax, ay, az	Valores de aceleración (adimensionales, vector normalizado) en los tres ejes del espacio, capturados por el acelerómetro en el instante t
t1	Etiqueta de tiempo correspondiente a la captura de datos

6.2.2 Función para obtención de matriz de rotación y conversión a cuaternión unitario (rotmtrx)

Esta función implementa dos cálculos:

- Matriz de rotación en base a las aceleraciones medidas
- Conversión de la matriz de rotación a cuaternión unitario

$$q = rotmtrx(ax, ay, az)$$

Tabla 16. Datos de entrada de la función rotmtrx

Datos de entrada	
ax, ay, az	Valores de aceleración (adimensionales, vector normalizado) en los tres ejes del espacio, capturados por el acelerómetro en el instante t

Tabla 17. Cálculos intermedios en la función rotmtrx

Cálculos intermedios	
r11, r12, r13 r21, r22, r23 r31, r32, r33	Elementos de la matriz de rotación (rot)
q0, q1, q2, q3	Elementos del cuaternión de orientación equivalente a la matriz de rotación

Tabla 18. Resultado de la función rotmtrx

Resultado

q	Cuaternión de orientación equivalente a la matriz de rotación

6.2.3 Función para la obtención de los cuaterniones de orientación (compfilt)

Esta función implementa el algoritmo de filtro complementario para la obtención del cuaternion de orientación que define el giro que experimenta el sensor, en base a los valores medidos de velocidad angular y aceleración.

$$q = compfilt(wx, wy, wz, ax, ay, az, qp, deltat)$$

Tabla 19. Datos de entrada de la función compfilt

Datos de entrada	
wx, wy, wz	Valores de velocidad angular (en rad/s) en los tres ejes del espacio, capturados del giroscopio, en el instante t
ax, ay, az	Valores de aceleración (adimensionales, vector normalizado) en los tres ejes del espacio, capturados por el acelerómetro en el instante t
qp	Cuaternión de orientación previa (en el instante t-1). Se toma el criterio de orientación inversa: sistema global relativo al local
deltat	Diferencia entre t y t-1

Tabla 20. Cálculos intermedios en la función compfilt

Cálculos intermedios	
qw	Predicción: resultado parcial del cuaternión de orientación basado únicamente en los valores de velocidad angular
derqw	Derivada del cuaternión de orientación
dqacc	Corrección basada en datos de acelerómetro
dqacce	Corrección escalada
dqaccn	Corrección normalizada

gp	Predicción del vector de la gravedad
om	Ángulo Ω
alfa	Ganancia

Tabla 21. Resultado de la función compfilt

Resultado	
q	Cuaternión de orientación (en el instante t). Se toma el criterio de orientación inversa: sistema global relativo al local

6.2.4 Función para la construcción de la Direct Cosine Matrix (dcm)

A través de esta función se construye la matriz DCM (Direct Cosine Matrix), conocido un cuaternión de orientación. Esta matriz es necesaria para obtener la predicción del vector de gravedad (gp), como parte del algoritmo implementado en la función compfilt.

$$r = dcm(qor)$$

Tabla 22. Datos de entrada de la función dcm

Datos de entrada	
qor	Cuaternión de orientación

Tabla 23. Cálculos intermedios en la función dcm

Cálculos intermedios	
r11, r12, r13 r21, r22, r23 r31, r32, r33	Elementos de la matriz 3x3

Tabla 24. Resultado de la función dcm

Resultado	
r	Matriz DCM

6.2.5 Función para la rotación de vectores usando cuaterniones (rotquat)

La rotación de un vector en el espacio tridimensional, expresado en forma de cuaternión (con su primer término nulo), se realiza a través de esta función, previa determinación del cuaternión de orientación correspondiente.

$$fin\_quat = rotquat(ini\_quat, q)$$

Tabla 25. Datos de entrada de la función rotquat

Datos de entrada	
ini_quat	Vector tridimensional en forma de cuaternión
q	Cauternión de orientación

Tabla 26. Cálculos intermedios en la función rotquat

Cálculos intermedios	
qc	Conjugado del cuaternión de orientación

Tabla 27. Resultado de la función rotquat

Resultado	
fin_quat	Vector tridimensional, girado, en forma de cuaternión

6.3 Código del subsistema generador de imagen

La parte del código que maneja la representación gráfica del movimiento capturado se puede dividir en dos partes:

- Inicialización del sistema de representación gráfica
- Actualización de los valores de la representación gráfica

Se utiliza el módulo VPython (Visual Python 7), que permite abrir una ventana del navegador de internet en la que se muestran elementos tridimensionales.

6.3.1 Inicialización del sistema de representación gráfica

El código correspondiente a la inicialización del sistema de representación gráfica se ubica fuera del bucle principal del programa, de manera que se ejecuta una sola vez.

En esta parte se definen los elementos básicos que constituyen la representación gráfica:

- Ejes, para facilitar la visualización
- Perspectiva, mediante el posicionamiento del punto de vista respecto al sistema de coordenadas
- Figuras que representan los objetos cuyo movimiento se está capturando
- Relación entre las figuras, restricciones
- Elementos auxiliares

Se han tomado prismas (elementos tipo "box" en VPython) para representar las partes del cuerpo humano como torso, brazo o antebrazo.
La orientación de estas figuras viene determinada por dos vectores: un vector ("axis") alineado al eje longitudinal del prisma y un segundo vector ("up"), perpendicular al primero, que determina el giro sobre dicho eje longitudinal.

Finalmente, como resultado de la ejecución de este fragmento de código, se abre una ventana del navegador, se sitúa el punto de vista y se generan las figuras en una posición y orientación inicial predefinidas.

Así, ya queda todo dispuesto para que se visualice el movimiento mediante la actualización de los vectores que determinan la orientación de las figuras.

6.3.2 Actualización de los valores de la representación gráfica

Durante la ejecución del bucle principal, para cada instante de tiempo, se obtiene el cuaternión de orientación correspondiente a los valores entregados por los sensores. Con esta información se puede calcular el giro del sistema de referencia local (vinculado al sensor) y actualizar los valores de los vectores que determinan la orientación del modelo gráfico correspondiente.

La actualización del valor de los vectores "axis" y "up" que determinan la orientación de los elementos del tipo "box" se realiza de forma directa, mediante el operador "=".
Sin embargo, la representación de los ejes de los sistemas de coordenadas se realizaría mediante elementos del tipo "curve", lo que exigiría el uso del procedimiento "modify" para la actualización de sus valores.

Finalmente, hay que tener en consideración que la dimensión longitudinal de las figuras tipo "box" está relacionada con el valor del vector "axis", por lo que podría verse afectada a lo largo del bucle. Es por esto que se toma un vector "axis" unitario y se aseguran las proporciones de los objetos mediante la multiplicación por constantes.

7 REALIZACIÓN DE PRUEBAS

7.1 Definición de las condiciones de las pruebas

7.1.1 Calibración

Para cada uno de los sensores que forman parte del sistema se realiza una captura en condiciones estáticas (sensor inmóvil) de 1.000 muestras que contienen los valores de aceleración y velocidad angular en los tres ejes y permite determinar el valor promedio de la medición y con éste estimar la desviación respecto al valor esperado.
Estos valores de offset se toman como valores de entrada en el software para corregir los datos entregados por los sensores.

7.1.2 Observación del efecto deriva

Para el conjunto del sistema se realiza una monitorización de los valores de orientación calculados, con los sensores en condiciones estáticas (sensor inmóvil), durante 5 minutos. Así se busca determinar si el error (deriva) implícito en los datos entregados por los sensores afecta al resultado del cálculo de la orientación.

7.1.3 Puesta en marcha

El sistema se ha de poner en marcha desde una posición concreta de los sensores. Esto se debe a que la inicialización de la determinación de la orientación se basa únicamente en los datos de aceleración y la estimación que se obtiene presenta indeterminaciones para ciertas posiciones.
Se establece como posición de inicio del sistema una alineación horizontal de los sensores.

7.1.4 Secuencia de movimientos prefijados

Para la verificación de los resultados del sistema se establece una secuencia de movimientos prefijados partiendo de la posición de inicio.

La definición de las posiciones se realiza indicando la alineación de los ejes "axis" y "up" (que definen la orientación de las figuras de la representación gráfica) respecto al sistema de coordenadas global (X, Y, Z)

Tabla 28. Posiciones de la secuencia de verificación

Posición	**("axis","up")**		
	Sensor 1	**Sensor 2**	**Sensor 3**
"Inicio"	X,Z	Y,Z	Y,Z
"Ángulo"	X,Z	Y,Z	Z,-Y
"Arriba"	X,Z	Z,-Y	Z,-Y
"Abajo"	X,Z	-Z,Y	-Z,Y
"Frente"	X,Z	X,Z	X,Z

7.2 Resultados de las pruebas

7.2.1 Valores de offset de los sensores

Tabla 29. Resultados de calibración

Offset	**Sensor 1**	**Sensor 2**	**Sensor 3**
ax (m/s^2)	-7,855	-3,340	0,185
ay (m/s^2)	0.060	0,020	-0,540
az (m/s^2)	0,415	-0,525	-0,625

wx (°/s)	4,1002	-0,3051	8,2278
wy (°/s)	0,5588	1,1273	-5,5049
wz (°/s)	0,1942	-0,1592	0,3980

7.2.2 Efecto del error de deriva

Se observa una notable desviación de la orientación de las figuras en el resultado gráfico obtenido durante la monitorización en condiciones estáticas. Esto se debe a que la orientación calculada por el algoritmo se ve afectada por el error de deriva que introducen los sensores.

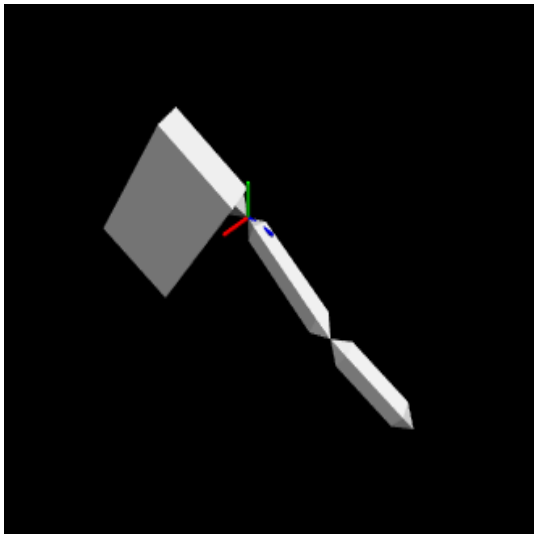

Ilustración 5. Salida gráfica inicial en condiciones estáticas

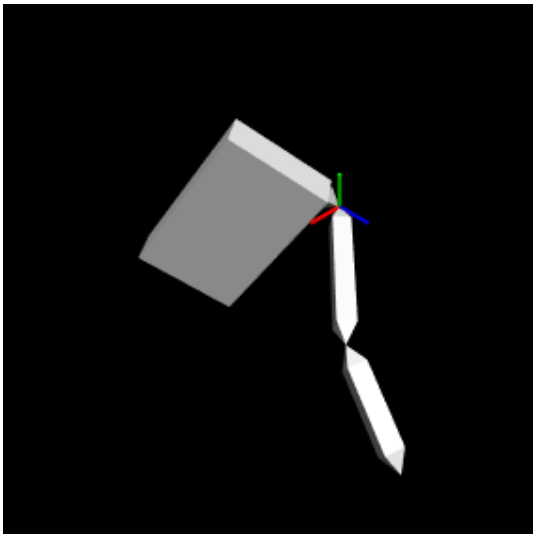

Ilustración 6. Salida gráfica final en condiciones estáticas

La minimización de este efecto puede ser abordada mediante diferentes estrategias (ajuste de los parámetros variables del algoritmo de cálculo de la orientación, incremento de 6 a 9 grados de libertad mediante la incorporación de magnetómetros al sistema) que quedan fuera del alcance de este proyecto.

7.2.3 Salidas gráficas de los movimientos prefijados

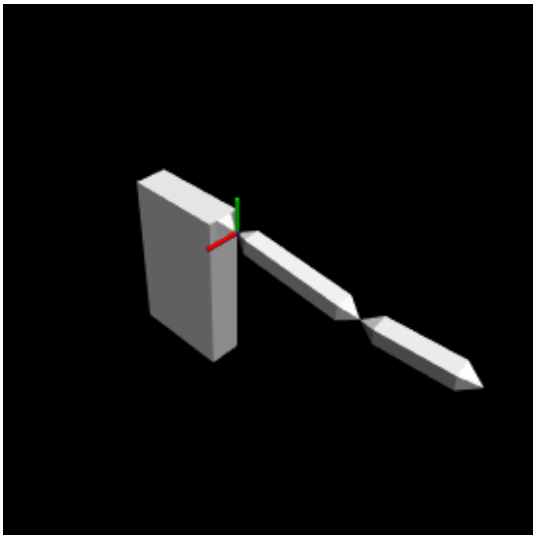

Ilustración 7. Salida gráfica de la posición de inicio

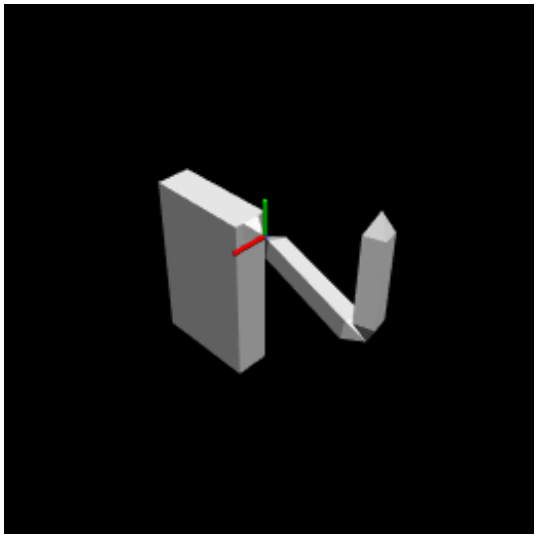

Ilustración 8. Salida gráfica de la posición "ángulo"

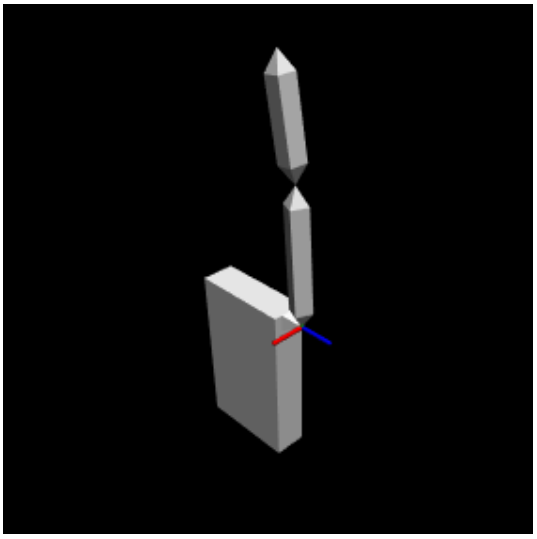

Ilustración 9. Salida gráfica de la posición "arriba"

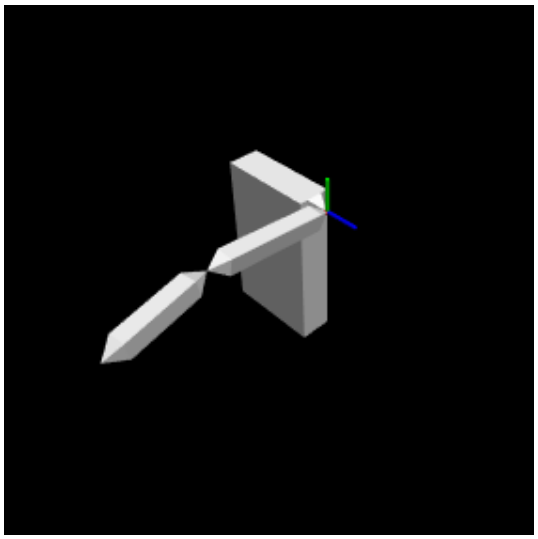

Ilustración 10. Salida gráfica de la posición "frente"

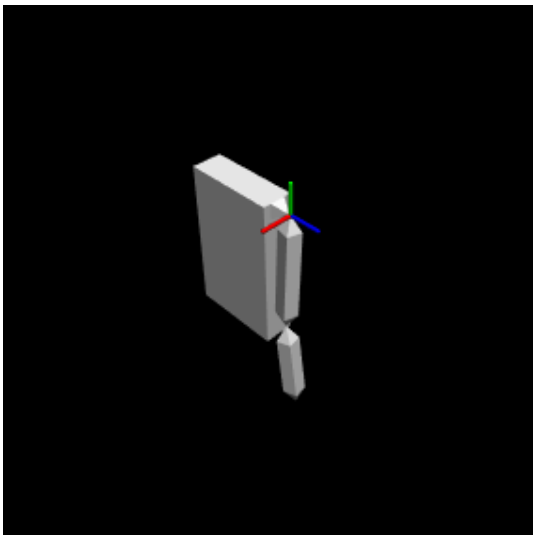

Ilustración 11. Salida gráfica de la posición "abajo"

8 CONCLUSIONES

8.1 Logros alcanzados

El presente proyecto de sistema de captura de movimientos basado en sistemas electrónicos de bajo coste ha permitido dar cumplimiento a los objetivos fijados:

- Se han descrito los fundamentos teóricos del problema de la captura del movimiento.
- Utilizando sensores inerciales de bojo coste de 6 grados de libertad, se ha construido un sistema físico que captura el movimiento de objetos reales.
- Se ha desarrollado un programa, que ejecutado por un microprocesador de bajo coste, permite procesar los datos entregados por los sensores inerciales.
- Se ha implementado un algoritmo que permite obtener una parametrización de la orientación en base a datos de sensores inerciales.
- Se ha desarrollado una visualización 3D que reproduce los movimientos de objetos reales.

8.2 Líneas de trabajo futuro

Algunos aspectos sobre los que se podría avanzar a partir del sistema desarrollado en este proyecto son:

- Comunicación WiFi (microprocesador – servidor o sensores – servidor) que permita al sistema integrarse como IoT.
- Integración de información de magnetómetro (noveno grado de libertad), previsto en el algoritmo utilizado para implementar el filtro complementario.
- Utilizar ganancia adaptativa en lugar de ganancia fija en la implementación del filtro complementario.

BIBLIOGRAFÍA

Básica:

Manon Kok, Jeroen D. Hol and Thomas B. Schön (2017), "Using Inertial Sensors for Position and Orientation Estimation", Foundations and Trends in Signal Processing: Vol. 11: No. 1-2, pp 1-153. http://dx.doi.org/10.1561/2000000094

Roberto G. Valenti, Ivan Dryanovski and Jizhong Xiao (2015), "Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs", Sensors 2015, 15, 19302-19330

Complementaria:

M. Furniss (1999), "Motion capture", MIT Communications Forum

Grace Whaba (1965), "A Least Square Estimate of Spacecraft Attitude", Soc. Ind. Appl. Math. (SIAM) Rev.1965, 7, 409–1965

Rubén Imbers (2018), "Sistema de captura de movimientos basado en sistemas electrónicos de bajo coste", Proyecto Final de Máster

Consultas a webs

Arduino Language Reference
https://www.arduino.cc/reference/en/

Python Tutorial:
https://docs.python.org/3/tutorial/index.html

VPython Language Reference:
https://www.glowscript.org/docs/VPythonDocs/index.html

Datasheets / Especificaciones / Mapas de registros

Arduino Uno Rev3:
https://store.arduino.cc/usa/arduino-uno-rev3

Sensor inercial (MPU 6050):
https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

LISTADO DE SIGLAS

Listado siglas, abreviaturas y acrónimos utilizados en el documento junto con sus respectivos significados.

UNED	Universidad Nacional de Educación a Distancia
UCM	Universidad Complutense de Madrid
PFM	Proyecto Final de Máster
IDE	Integrated Development Environment
MEMS	Micro Electro Mechanical Systems
IMU	Inertial Measurement Unit
E/S	Entradas / Salidas
I2C	Inter-integrated Circuit
SPI	Serial Peripheral Interface
IoT	Internet of Things
MIT	Massachusetts Institute of Technology
3D	3 Dimensiones
LERP	Linear Interpolation
SLERP	Spherical Linear Interpolation

ANEXOS

Anexo 1 Código controlador

```
// MPU6050TMC v6.1
// Captura de datos (aceleración y velocidad angular en los tres ejes) de tres sensores MPU 6050 en bus I2C
// Tristán Mas Carrascosa
// 29 de agosto de 2019
#include<Wire.h>
const int MPU_addr=0x68;  // Dirección I2C del sensor MPU 6050
// Offset de calibración para cada variable
const float Offax1=-7.855;
const float Offay1=0.060;
const float Offaz1=0.415;
const float Offax2=-3.340;
const float Offay2=0.020;
const float Offaz2=-0.525;
const float Offax3=0.185;
const float Offay3=-0.540;
const float Offaz3=-0.625;
const float Offgx1=4.1002;
const float Offgy1=0.5588;
const float Offgz1=0.1942;
const float Offgx2=-0.3051;
const float Offgy2=1.1273;
const float Offgz2=-0.1592;
const float Offgx3=8.2278;
const float Offgy3=-5.5049;
```

```
const float Offgz3=0.3980;
bool statePin2,statePin4,statePin7;
byte error,who,pwr,gconf,aconf;
int16_t Tmp;
int16_t AcX1,AcY1,AcZ1,GyX1,GyY1,GyZ1;
int16_t AcX2,AcY2,AcZ2,GyX2,GyY2,GyZ2;
int16_t AcX3,AcY3,AcZ3,GyX3,GyY3,GyZ3;
//
// Función de lectura de datos de un sensor
void ReadI2C(int16_t &AcX, int16_t &AcY, int16_t &AcZ, int16_t &GyX, int16_t &GyY,
int16_t &GyZ)
{
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);  // request a total of 14 registers
  AcX=Wire.read()<<8|Wire.read();        //  0x3B  (ACCEL_XOUT_H)  &  0x3C
(ACCEL_XOUT_L)
  AcY=Wire.read()<<8|Wire.read();        //  0x3D  (ACCEL_YOUT_H)  &  0x3E
(ACCEL_YOUT_L)
  AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
}
//
// Función de configuración de un sensor
void SensorSetup()
{
```

```
  // Prueba de comunicación
  Wire.beginTransmission(MPU_addr);
  error=Wire.endTransmission();
  if(error==0){
//    Serial.print("COMUNICACIÓN EXITOSA CON EL SENSOR");
//    Serial.println("");
  }
  else{
//    Serial.print("ERROR DE COMUNICACIÓN CON EL SENSOR");
//    Serial.println("");
  }
  // Reseteo del sensor
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // Registro 0x6B: PWR_MGMT_1
  Wire.write(0x40);  // bit7=1: reseteo
  Wire.endTransmission(false);
  delay(100);
  // Desactivación de la función "sleep"
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0x00);  // bit6=0: función sleep desactivada
  Wire.endTransmission(false);
  delay(100);
  // Configuración del fondo de escala del giroscopio
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x1B);  // Registro GYRO_CONFIG
  Wire.write(0x18);  // bit4=1, bit3=1: fondo de escala 2000 dps
  Wire.endTransmission(false);
  delay(100);
  // Configuración del fondo de escala del acelerómetro
```

```
Wire.beginTransmission(MPU_addr);
Wire.write(0x1C);  // Registro ACCEL_CONFIG
Wire.write(0x18);  // bit4=1, bit3=1: fondo de escala 16g
Wire.endTransmission(false);
delay(100);
// Verificación de la dirección I2C del sensor
Wire.beginTransmission(MPU_addr);
Wire.write(0x75);  // Registro 0x75: WHO_AM_I
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr,1,true);
who=Wire.read();
// Verificación de la desactivación de la función "sleep"
Wire.beginTransmission(MPU_addr);
Wire.write(0x6B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr,1,true);
pwr=Wire.read();
// Verificación de la configuración del giroscopio
Wire.beginTransmission(MPU_addr);
Wire.write(0x1B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr,1,true);
gconf=Wire.read();
// Verificación de la configuración del acelerómetro
Wire.beginTransmission(MPU_addr);
Wire.write(0x1C);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr,1,true);
aconf=Wire.read();
// Salida de pantalla de las respuestas de verificación
```

```
//  Serial.print("Who: ");
//  Serial.print(who);
//  Serial.print(" Pwr: ");
//  Serial.print(pwr);
//  Serial.print(" GyroConf: ");
//  Serial.print(gconf);
//  Serial.print(" AccelConf: ");
//  Serial.print(aconf);
//  Serial.println("");
}
// Función para observar el estado de los pines de asignación secuencial
void pinState()
{
  statePin2=digitalRead(2);
  statePin4=digitalRead(4);
  statePin7=digitalRead(7);
//  Serial.print("Pin2/Pin4/Pin7: ");
//  Serial.println("");
//  Serial.print(statePin2);
//  Serial.print("\t");
//  Serial.print(statePin4);
//  Serial.print("\t");
//  Serial.print(statePin7);
//  Serial.println("");
}
//
void setup(){
// Inicialización del puerto serie
  Serial.begin(115200);
//
```

```
// Inicialización del bus I2C
  Wire.begin();
//
// Configuración de los pines de asignación secuencial
  pinMode(2,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(7,OUTPUT);
  digitalWrite(2,HIGH);  // Inicialización con valor 1 en los 3 pines
  digitalWrite(4,HIGH);
  digitalWrite(7,HIGH);
//  Serial.print("*** Inicialización de pines ***");
//  Serial.println("");
  pinState();
//
// Configuración consecutiva de inicio de los módulos
  digitalWrite(2,LOW);
  delay(500);
//  Serial.print("*** Sensor 1 ***");
//  Serial.println("");
  pinState();
  SensorSetup();
  digitalWrite(2,HIGH);
  digitalWrite(4,LOW);
  delay(500);
//  Serial.print("*** Sensor 2 ***");
//  Serial.println("");
  pinState();
  SensorSetup();
  digitalWrite(4,HIGH);
  digitalWrite(7,LOW);
```

```
  delay(500);
//  Serial.print("*** Sensor 3 ***");
//  Serial.println("");
  pinState();
  SensorSetup();
//  Serial.print("*** FIN DE LA CONFIGURACIÓN ***");
//  Serial.println("");
  digitalWrite(7,HIGH);
  delay(500);
}
void loop(){
  digitalWrite(2,LOW);
  ReadI2C(AcX1,AcY1,AcZ1,GyX1,GyY1,GyZ1);
//
  digitalWrite(2,HIGH);
  digitalWrite(4,LOW);
  ReadI2C(AcX2,AcY2,AcZ2,GyX2,GyY2,GyZ2);
//
  digitalWrite(4,HIGH);
  digitalWrite(7,LOW);
  ReadI2C(AcX3,AcY3,AcZ3,GyX3,GyY3,GyZ3);
//
  digitalWrite(7,HIGH);
//
// ESCALAR LOS VALORES "CRUDOS" REGISTRADOS Y APLICAR OFFSET
  float ax1_ms2 = AcX1 *(9.81*16.0/32768.0)+Offax1;
  float ay1_ms2 = AcY1 *(9.81*16.0/32768.0)+Offay1;
  float az1_ms2 = AcZ1 *(9.81*16.0/32768.0)+Offaz1;
  float gx1_dps = GyX1 *(2000.0/32768.0)+Offgx1;
  float gy1_dps = GyY1 *(2000.0/32768.0)+Offgy1;
```

```
  float gz1_dps = GyZ1 *(2000.0/32768.0)+Offgz1;
  float ax2_ms2 = AcX2 *(9.81*16.0/32768.0)+Offax2;
  float ay2_ms2 = AcY2 *(9.81*16.0/32768.0)+Offay2;
  float az2_ms2 = AcZ2 *(9.81*16.0/32768.0)+Offaz2;
  float gx2_dps = GyX2 *(2000.0/32768.0)+Offgx2;
  float gy2_dps = GyY2 *(2000.0/32768.0)+Offgy2;
  float gz2_dps = GyZ2 *(2000.0/32768.0)+Offgz2;
  float ax3_ms2 = AcX3 *(9.81*16.0/32768.0)+Offax3;
  float ay3_ms2 = AcY3 *(9.81*16.0/32768.0)+Offay3;
  float az3_ms2 = AcZ3 *(9.81*16.0/32768.0)+Offaz3;
  float gx3_dps = GyX3 *(2000.0/32768.0)+Offgx3;
  float gy3_dps = GyY3 *(2000.0/32768.0)+Offgy3;
  float gz3_dps = GyZ3 *(2000.0/32768.0)+Offgz3;

// --- Mostrar valores ---
  // Acelerometro 1
  Serial.print(ax1_ms2, DEC);
  Serial.print("\t");
  Serial.print(ay1_ms2, DEC);
  Serial.print("\t");
  Serial.print(az1_ms2, DEC);
  Serial.print("\t");
  // Giroscopio 1
  Serial.print(gx1_dps, DEC);
  Serial.print("\t");
  Serial.print(gy1_dps, DEC);
  Serial.print("\t");
  Serial.print(gz1_dps, DEC);
  Serial.print("\t");
  // Acelerometro 2
```

```
  Serial.print(ax2_ms2, DEC);
  Serial.print("\t");
  Serial.print(ay2_ms2, DEC);
  Serial.print("\t");
  Serial.print(az2_ms2, DEC);
  Serial.print("\t");
  // Giroscopio 2
  Serial.print(gx2_dps, DEC);
  Serial.print("\t");
  Serial.print(gy2_dps, DEC);
  Serial.print("\t");
  Serial.print(gz2_dps, DEC);
  Serial.print("\t");
  // Acelerometro 3
  Serial.print(ax3_ms2, DEC);
  Serial.print("\t");
  Serial.print(ay3_ms2, DEC);
  Serial.print("\t");
  Serial.print(az3_ms2, DEC);
  Serial.print("\t");
  // Giroscopio 3
  Serial.print(gx3_dps, DEC);
  Serial.print("\t");
  Serial.print(gy3_dps, DEC);
  Serial.print("\t");
  Serial.print(gz3_dps, DEC);
  Serial.print("\t");
//
// Fin medicion
  Serial.println("");
```

```
//
  delay(100);
}
```

Anexo 2 Código servidor y representación gráfica

```
# ModeloBrazo v2
# Captura de datos de puerto serie, determinación de orientación y generación de salida gráfica
# Tristán Mas Carrascosa
# 11 septiembre 2019
#
import serial, time, math
import numpy as np
import vpython as vp
import quaternion
#
# Inicio de la comunicación serie
#
arduino = serial.Serial('COM4', 115200)
#
# Definición de constantes
#   ini_x, ini_y -> Ejes x e y del sistema de coordenadas
#   t0 -> Origen del tiempo
#   k -> Conversión de grados a radianes
#
ini_x=[[1],
       [0],
       [0]]
ini_y=[[0],
       [1],
       [0]]
ini_z=[[0],
       [0],
       [1]]
```

```
t0=time.time()
k=2*np.pi/360
#
# FUNCIÓN de captura de datos del puerto serie
#
def datadq(t0):
    rawString = arduino.readline() #Captura datos del puerto serie
    t1=time.time()
    deltat=t1-t0
    t0=t1
    IMUdata = rawString.split()
    ax1=(float(IMUdata[0]))#dato en m/s2
    ay1=(float(IMUdata[1]))#dato en m/s2
    az1=(float(IMUdata[2]))#dato en m/s2
    wx1=k*(float(IMUdata[3]))#dato en rad/seg
    wy1=k*(float(IMUdata[4]))#dato en rad/seg
    wz1=k*(float(IMUdata[5]))#dato en rad/seg
    ax2=(float(IMUdata[6]))#dato en m/s2
    ay2=(float(IMUdata[7]))#dato en m/s2
    az2=(float(IMUdata[8]))#dato en m/s2
    wx2=k*(float(IMUdata[9]))#dato en rad/seg
    wy2=k*(float(IMUdata[10]))#dato en rad/seg
    wz2=k*(float(IMUdata[11]))#dato en rad/seg
    ax3=(float(IMUdata[12]))#dato en m/s2
    ay3=(float(IMUdata[13]))#dato en m/s2
    az3=(float(IMUdata[14]))#dato en m/s2
    wx3=k*(float(IMUdata[15]))#dato en rad/seg
    wy3=k*(float(IMUdata[16]))#dato en rad/seg
    wz3=k*(float(IMUdata[17]))#dato en rad/seg
```

```
    return
ax1,ay1,az1,wx1,wy1,wz1,ax2,ay2,az2,wx2,wy2,wz2,ax3,ay3,az3,wx3,wy3,wz3,t1
#
# FUNCIÓN de rotación del sistema de coordenadas usando matriz de rotación
(secuencia XYZ)
#
def rotmtrx(ax,ay,az):
    roll=-math.atan2(ay,az)
    pitch=-math.atan2(-ax,math.sqrt(ay*ay+az*az))
    r11=math.cos(pitch)
    r12=0
    r13=-math.sin(pitch)
    r21=math.sin(pitch)*math.sin(roll)
    r22=math.cos(roll)
    r23=math.cos(pitch)*math.sin(roll)
    r31=math.cos(roll)*math.sin(pitch)
    r32=-math.sin(roll)
    r33=math.cos(pitch)*math.cos(roll)
    rot=np.array([[r11,r12,r13],
                  [r21,r22,r23],
                  [r31,r32,r33]])
    fin_x=rot@ini_x
    fin_y=rot@ini_y
    fin_z=rot@ini_z
    q0=math.sqrt(1+r11+r22+r33)/2
    q1=(r32-r23)/(4*q0)
    q2=(r13-r31)/(4*q0)
    q3=(r21-r12)/(4*q0)
    qp=np.quaternion(q0,q1,q2,q3)
    qp=qp/math.sqrt(q0*q0+q1*q1+q2*q2+q3*q3) # normalización de qp
```

```
    return fin_x,fin_y,fin_z,qp
#
# FUNCIÓN de rotación usando cuaterniones
#
def rotquat(ini_quat,q):
    qc=np.quaternion(q.w,-q.x,-q.y,-q.z) #conjugado de q
    fin_quat=qc*ini_quat*q
    return fin_quat
##
##FUNCIÓN de filtro complementario
##Datos de entrada:
##  Observaciones giroscopio (wx,wy,wz) en rad/s
##  Observaciones acelerómetro (ax,ay,az) NORMALIZADA
##  Orientación INVERSA previa, en t-1, en forma de cuaternión (qp)
##  Delta t, definido fuera de la función (deltat)
##Cálculos intermedios:
## Predicción (qw)
## Derivada de qw (derqw)
## Corrección basada en datos de acelerómetro (dqacc)
##  escalado (dqacce)
##  normalizado (dqaccn)
## Predicción de la gravedad (gp)
## Omega (om)
## Ganancia (alfa)
##Resultado:
## orientación INVERSA (en t) en forma de cuaternión (q)
##
def compfilt(wx,wy,wz,ax,ay,az,qp):
    ## Paso 1: predicción
    derqw=-1/2*np.quaternion(0,-wx,-wy,wz)*qp
```

```
    qw=qp+derqw*deltat
    ## Paso 2: Corrección basada en datos del acelerómetro
    qwc=np.quaternion(qw.w,-qw.x,-qw.y,-qw.z) #conjugado de qw
    gp=dcm(qwc)@[[ax],[ay],[az]]
    gx=gp[0][0]
    gy=gp[1][0]
    gz=gp[2][0]
    normagp=math.sqrt(gx*gx+gy*gy+gz*gz)
    gx=gx/normagp ##normalización de gp
    gy=gy/normagp
    gz=gz/normagp
    alfa=0.8 ## valor entre 0 y 1
    dqacc=np.quaternion(math.sqrt((gz+1)/2),-
gy/math.sqrt(2*(gz+1)),gx/math.sqrt(2*(gz+1)),0)
    om=np.arccos(dqacc.w)
    dqacce=(1-alfa)*np.quaternion(1,0,0,0)+alfa*dqacc
    ## verificar norma cuaternion ## si dqacc.w > 0,9
    if dqacc.w>0.9:

dqaccn=dqacce/math.sqrt(dqacce.w*dqacce.w+dqacce.x*dqacce.x+dqacce.
y*dqacce.y+dqacce.z*dqacce.z)
    else:
        dqaccn=math.sin((1-
alfa)*om)/math.sin(om)*np.quaternion(1,0,0,0)+math.sin(alfa*om)/math.sin(alfa)*d
qacc
    q=qw*dqaccn
    q=q/math.sqrt(q.w*q.w+q.x*q.x+q.y*q.y+q.z*q.z) #normalización de q
    return q
##
```

```
##FUNCIÓN de construcción de la Direct Cosine Matrix (DCM) en términos de cuaternión de orientación
##Datos de entrada:
## Cuaternion de orientación (qor)
##Resultado:
## Direct Cosine Matrix
def dcm(qor):
    r11=qor.w*qor.w+qor.x*qor.x-qor.y*qor.y-qor.z*qor.z
    r12=2*(qor.x*qor.y-qor.w*qor.z)
    r13=2*(qor.x*qor.z+qor.w*qor.y)
    r21=2*(qor.x*qor.y+qor.w*qor.z)
    r22=qor.w*qor.w-qor.x*qor.x+qor.y*qor.y-qor.z*qor.z
    r23=2*(qor.y*qor.z-qor.w*qor.x)
    r31=2*(qor.x*qor.z-qor.w*qor.y)
    r32=2*(qor.y*qor.z+qor.w*qor.x)
    r33=qor.w*qor.w-qor.x*qor.x-qor.y*qor.y+qor.z*qor.z
    r=np.array([[r11,r12,r13],
                [r21,r22,r23],
                [r31,r32,r33]])
    return r
#
# --- BUCLE PRINCIPAL ---
#
ax1,ay1,az1,wx1,wy1,wz1,ax2,ay2,az2,wx2,wy2,wz2,ax3,ay3,az3,wx3,wy3,wz3,t1=datadq(t0)
fin_x1,fin_y1,fin_z1,qp1=rotmtrx(ax1,ay1,az1)
fin_x2,fin_y2,fin_z2,qp2=rotmtrx(ax2,ay2,az2)
fin_x3,fin_y3,fin_z3,qp3=rotmtrx(ax3,ay3,az3)
ini_x_quat1=np.quaternion(0,1,0,0)
ini_y_quat1=np.quaternion(0,0,1,0)
```

```
ini_z_quat1=np.quaternion(0,0,0,1)
ini_x_quat2=np.quaternion(0,1,0,0)
ini_y_quat2=np.quaternion(0,0,1,0)
ini_z_quat2=np.quaternion(0,0,0,1)
ini_x_quat3=np.quaternion(0,1,0,0)
ini_y_quat3=np.quaternion(0,0,1,0)
ini_z_quat3=np.quaternion(0,0,0,1)
#
# Representación gráfica
# refx, refy, refz -> Sistema de coordenadas
#
refx=vp.curve(pos=[vp.vector(0,0,0),vp.vector(10,0,0)],color=vp.color.red)
refy=vp.curve(pos=[vp.vector(0,0,0),vp.vector(0,10,0)],color=vp.color.blue)
refz=vp.curve(pos=[vp.vector(0,0,0),vp.vector(0,0,10)],color=vp.color.green)
vp.scene.camera.pos=vp.vector(100,100,100)
vp.scene.camera.axis=vp.vector(-100,-100,-100)
vp.scene.up=vp.vector(-1,-1,1)
torso=vp.box(pos=vp.vector(0,-20,-22.5),
        axis=vp.vector(0,1,0),
        up=vp.vector(0,0,1),
        size=vp.vector(30,50,10))
hombro_izq=vp.pyramid(pos=vp.vector(0,-5,0),
            axis=vp.vector(0,1,0),
            size=vp.vector(5,5,5))
p_brz=vp.pyramid(pos=vp.vector(0,0,0),
            axis=vp.vector(1,0,0),
            size=vp.vector(5,5,5))
c_brz=vp.box(pos=vp.vector(-15,0,0),
        axis=vp.vector(1,0,0),
        up=vp.vector(0,0,1),
```

```
        size=vp.vector(30,5,5))
d_brz=vp.pyramid(pos=vp.vector(-30,0,0),
            axis=vp.vector(-1,0,0),
            size=vp.vector(5,5,5))
brazo=vp.compound([p_brz,c_brz,d_brz])
p_nbz=vp.pyramid(pos=vp.vector(0,0,0),
            axis=vp.vector(1,0,0),
            size=vp.vector(5,5,5))
c_nbz=vp.box(pos=vp.vector(-10,0,0),
        axis=vp.vector(1,0,0),
        up=vp.vector(0,0,1),
        size=vp.vector(20,5,5))
d_nbz=vp.pyramid(pos=vp.vector(-20,0,0),
            axis=vp.vector(-1,0,0),
            size=vp.vector(5,5,5))
antebrazo=vp.compound([p_nbz,c_nbz,d_nbz])
# Posición inicial
torso.axis=vp.vector(0,1,0)
torso.up=vp.vector(0,0,1)
torso.pos=-20*torso.axis-22.5*torso.up
hombro_izq.axis=torso.axis
hombro_izq.pos=-5*torso.axis
torso.size=vp.vector(30,50,10)
hombro_izq.size=vp.vector(5,5,5)
brazo.axis=vp.vector(0,1,0)
brazo.up=vp.vector(0,0,1)
brazo.pos=20*brazo.axis
codo=40*brazo.axis
antebrazo.axis=vp.vector(0,1,0)
antebrazo.up=vp.vector(0,0,1)
```

```
antebrazo.pos=codo+15*antebrazo.axis
#
while True:

ax1,ay1,az1,wx1,wy1,wz1,ax2,ay2,az2,wx2,wy2,wz2,ax3,ay3,az3,wx3,wy3,wz3,t1=datadq(t0) # Captura datos (acelerómetro y giroscopio) del puerto serie
   axn1=ax1/math.sqrt(ax1*ax1+ay1*ay1+az1*az1)  # Normalización de la aceleración
   ayn1=ay1/math.sqrt(ax1*ax1+ay1*ay1+az1*az1)
   azn1=az1/math.sqrt(ax1*ax1+ay1*ay1+az1*az1)
   axn2=ax2/math.sqrt(ax2*ax2+ay2*ay2+az2*az2)
   ayn2=ay2/math.sqrt(ax2*ax2+ay2*ay2+az2*az2)
   azn2=az2/math.sqrt(ax2*ax2+ay2*ay2+az2*az2)
   axn3=ax3/math.sqrt(ax3*ax3+ay3*ay3+az3*az3)
   ayn3=ay3/math.sqrt(ax3*ax3+ay3*ay3+az3*az3)
   azn3=az3/math.sqrt(ax3*ax3+ay3*ay3+az3*az3)
   deltat=t1-t0
   t0=t1
   fin_x1,fin_y1,fin_z1,nouso1=rotmtrx(ax1,ay1,az1)  # Rotación usando matriz de rotación
   fin_x2,fin_y2,fin_z2,nouso2=rotmtrx(ax2,ay2,az2)
   fin_x3,fin_y3,fin_z3,nouso3=rotmtrx(ax3,ay3,az3)
   q1=compfilt(wx1,wy1,wz1,axn1,ayn1,azn1,qp1) # Determinación del cuaternión de orientacion
   qp1=q1
   q2=compfilt(wx2,wy2,wz2,axn2,ayn2,azn2,qp2)
   qp2=q2
   q3=compfilt(wx3,wy3,wz3,axn3,ayn3,azn3,qp3)
   qp3=q3
   fin_x_quat1=rotquat(ini_x_quat1,q1)#Rotación usando cuaterniones
```

```
    fin_y_quat1=rotquat(ini_y_quat1,q1)
    fin_z_quat1=rotquat(ini_z_quat1,q1)
    fin_x_quat2=rotquat(ini_x_quat2,q2)
    fin_y_quat2=rotquat(ini_y_quat2,q2)
    fin_z_quat2=rotquat(ini_z_quat2,q2)
    fin_x_quat3=rotquat(ini_x_quat3,q3)
    fin_y_quat3=rotquat(ini_y_quat3,q3)
    fin_z_quat3=rotquat(ini_z_quat3,q3)
#Actualización de los valores de la representación gráfica
    torso.axis=vp.vector(fin_y_quat1.x,fin_y_quat1.y,fin_y_quat1.z)  # El vector "axis" debe ser unitario para que no se pierda la proporción al calcular "pos"
    torso.up=vp.vector(fin_z_quat1.x,fin_z_quat1.y,fin_z_quat1.z)
    torso.pos=-20*torso.axis-22.5*torso.up
    hombro_izq.axis=torso.axis
    hombro_izq.pos=-5*torso.axis
    torso.size=vp.vector(30,50,10)
    hombro_izq.size=vp.vector(5,5,5)
    brazo.axis=vp.vector(fin_y_quat2.x,fin_y_quat2.y,fin_y_quat2.z)
    brazo.up=vp.vector(fin_z_quat2.x,fin_z_quat2.y,fin_z_quat2.z)
    brazo.pos=20*brazo.axis
    codo=40*brazo.axis
    antebrazo.axis=vp.vector(fin_y_quat3.x,fin_y_quat3.y,fin_y_quat3.z)
    antebrazo.up=vp.vector(fin_z_quat3.x,fin_z_quat3.y,fin_z_quat3.z)
    antebrazo.pos=codo+15*antebrazo.axis
    time.sleep(0.1)
```