UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN

Desarrollo de librerías Python para comunicación bluetooth y análisis de datos de dispositivos vestibles de movimiento. Integración en herramienta de telerehabilitación.

Autor:

D. Daniel Diez Barredo

Tutor:

Dr. D. Mario Martínez Zarzuela

Valladolid, septiembre de 2022

TÍTULO:	**Desarrollo de librerías Python para comunicación bluetooth y análisis de datos de dispositivos vestibles de movimiento. Integración en herramienta de telerehabilitación.**
AUTOR:	**D. Daniel Diez Barredo**
TUTOR:	**Dr. D. Mario Martínez Zarzuela**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Carlos Gómez Peña**
VOCAL:	**Dr. D. David González Ortega**
SECRETARIO:	**Dr. D. Mario Martínez Zarzuela**
SUPLENTE 1:	**Dr. D. Jesús Poza Crespo**
SUPLENTE 2:	**Dr. D. Francisco Javier Díaz Pernas**

FECHA:
CALIFICACIÓN:

Agradecimientos

En primer lugar, agradecer a Mario por la confianza depositada en mi desde el primer día y darme la oportunidad de participar en el proyecto.

Sin olvidarme por supuesto de mis compañeros de laboratorio, especialmente a Javi por hacer más llevaderas las innumerables horas peleando contra los elementos.

Finalmente agradecer a mis padres por el apoyo incondicional.

Resumen

El objetivo de este Trabajo Fin de Grado consiste en la integración de sensores inerciales *Xsens Dot* sobre un proyecto de telerehabilitación, con el propósito de capturar los movimientos del paciente durante la realización de los ejercicios en el domicilio. A lo largo del trabajo se han realizado desarrollos que engloban todas las fases involucradas en el uso de estos sensores.

Para la fase de recolección de datos de los sensores, se ha desarrollado una librería en *Python* para su gestión a través de *bluetooth*, además para mejorar la experiencia de usuario en el uso de sensores se ha implementado una interfaz gráfica con *Pygame*.

En la fase de análisis se ha implementado un *script* en *Python* que se encarga de la reconstrucción del movimiento a través de *OpenSim*. Adicionalmente, se han creado varias herramientas de análisis de los datos recibidos con *PyOpenGL*.

Finalmente, en la fase de visualización se ha desarrollado un pequeño *script* para mostrar al personal clínico información útil sobre el resultado de las capturas y los análisis realizados sobre ellas.

Palabras clave

Telerehabilitación, IMU, Xsens Dot, Python, OpenSim, Pygame, PyOpenGL, bluetooth.

Abstract

The objective of this Master Thesis is the integration of Xsens Dot inertial sensors on a telerehabilitation system, with the purpose of capturing the patient's movements during home exercises. Throughout the work, different Python codes have been developed to encompass all the phases involved in the use of these sensors.

For the sensor data collection phase, a Python library for bluetooth connectivity management has been developed. In addition, with the aim of improving the user experience with inertial sensors, a graphical interface with Pygame has been made available.

In analysis phase we have implemented a Python script responsible for the patient's motion reconstruction using OpenSim. Additionally, we have created tools for the data analysis of the data received with PyOpenGL.

Finally, for visualization phase, a small script has been developed to show to the clinic staff useful information about the result of the movement captures.

Keywords

telerehabilitation, IMU, Xsens Dot, Python, OpenSim, Pygame, PyOpenGL, bluetooth.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 Introducción

1.1 Motivación y objetivos

Con los grandes avances tecnológicos en el campo de la salud en estas últimas décadas, los residentes de los países más industrializados como España ahora tienen una vida mucho más longeva, lo que ha traído consigo profundas transformaciones en la pirámide poblacional, entre ellas un proceso de envejecimiento notable con múltiples condiciones de salud que a menudo requieren de mayores cuidados.

Según la información de la *Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia 2020* elaborada por el *Instituto Nacional de Estadística*, en España hay 4,38 millones de personas que declaran tener alguna discapacidad, esto representa en torno al 9% de la población española [1].

Con la llegada de la crisis sanitaria de *2020* por el *COVID-19*, todavía a día de hoy dos años después continúan los problemas de saturación en el sistema nacional de salud español afectando a múltiples ámbitos del sector sanitario.

Según una nota de prensa publicada por la *SERMEF* (*sociedad española de rehabilitación y medicina física*), el 54% de los pacientes hospitalizados por *COVID-19* necesitan tratamientos de rehabilitación al darles el alta. Estos pacientes presentan principalmente una movilidad limitada por problemas derivados de su mal estado muscular y respiratorio, de todos ellos un 27.4% necesita ayuda para caminar [2].

Después de lo sucedido, las previsiones de la *Organización Mundial de la Salud* de las estimaciones sobre las necesidades en el campo de la rehabilitación, se han quedado cortas y ahora son los servicios de medicina física y rehabilitación los que han de tratar de coordinarse para atender esta nueva demanda sin desatender el resto de enfermedades que siguen requiriendo de estos cuidados.

Además, la propia *SERMEF* ha manifestado que la situación de la pandemia ha modificado significativamente el funcionamiento normal de los servicios sanitarios y ha causado que muchos de los tratamientos de rehabilitación hayan sido suspendidos, trayendo consigo posibles consecuencias irreversibles para la recuperación de muchos de los pacientes. Es por esto por lo que actualmente se necesitan infraestructuras, recursos humanos y se están buscando alternativas válidas para continuar con los tratamientos [3].

En este contexto, se presenta un proyecto de telerehabilitación que a través del uso de la tecnología contribuya a la descongestión del sistema sanitario, haciendo posible la realización de la terapia de rehabilitación de forma remota en el domicilio del propio paciente.

Este sistema no solo va a permitir dar servicio a los pacientes actuales de forma más cómoda para ellos, sino que además permitirá atender de forma simultánea a un mayor número de pacientes mejorando el servicio y disminuyendo los costes en tiempo y recursos del proceso de rehabilitación.

Esta disminución de los costes de servicio, tiene además el potencial de mitigar problemas de acceso de los pacientes con menos recursos y extender el acceso de los especialistas a áreas más rurales.

Los acontecimientos vividos durante la crisis sanitaria deben servir como aliciente para incorporarse a la aceleración en el proceso de digitalización y tratar de desarrollar nuevas herramientas para los tratamientos para seguir dando un buen servicio a los pacientes.

Algunos estudios clínicos que han tratado de comparar la eficacia de tratamientos de telerehabilitación frente a los realizados en clínica, no han encontrado diferencias estadísticas significativas entre el enfoque en el entorno tradicional y el telemático. Especialmente si se procura mantener el interés y la motivación del paciente a través de un enfoque basado en un juego con retos y logros, supervisados y diseñados siempre por un terapeuta [4].

Sobre la base de este proyecto de telerehabilitación descrito con más detalle en la *sección 3.2,* se añadirán una serie de sensores inerciales de bajo coste para recoger los movimientos que realiza el paciente durante sus ejercicios.

Con estos datos el personal clínico podrá controlar de forma remota el estado del paciente, evaluar sus avances y poder asignar ejercicios que se adapten de forma más eficiente a las necesidades específicas del paciente.

Todo esto es posible gracias a la tecnología *MEMS* (*Microelectromechanical systems*), que ha permitido el desarrollo de sensores inerciales más pequeños para la monitorización de la actividad motora y el control de salud. Esta miniaturización y el abaratamiento de los propios circuitos electrónicos ha jugado un papel clave en el aumento del uso actualmente de sistemas vestibles de sensores.

Por tanto, los objetivos principales que se pretenden conseguir en este trabajo son:

- Recoger datos de movimiento a través de sensores inerciales en la casa del paciente durante la realización de sus ejercicios.
- Hacer sencillo y asequible el uso de los sensores para todo tipo de usuarios.
- Analizar los datos recogidos para poder reconstruir el movimiento del paciente.
- Mostrar los datos recogidos por los sensores de forma útil para el personal clínico.

Manteniendo los objetivos del proyecto inicial de ser un sistema simple, de bajo coste y organizado que permita a los pacientes rehabilitarse desde casa manteniendo la sensación de cercanía por parte de los terapeutas en las rehabilitaciones presenciales.

1.2 Recursos hardware y software

Para que fuera posible la realización de este proyecto se ha necesitado el uso de los siguientes elementos:

Hardware

Para la realización de todos los desarrollos descritos a lo largo del presente documento, se ha hecho uso de un ordenador personal con las siguientes características técnicas:

- *Procesador*: Intel Core i7-6500U de 2.50Ghz.
- *Memoria RAM*: 8Gb DDR3.
- *Tarjeta Gráfica*: NVIDIA GeForce 920MX.

Además, para implementar el dispositivo que debe contener el bot de *Telegram* y la recolección de datos a través de sensores se han utilizado los siguientes recursos:

- *RaspberryPi* 4 Modelo B+ con 8 GB de RAM.
- Tarjeta *micro-SD* de 32GB.
- Cable *HDMI* a *micro HDMI*.
- Cinco sensores inerciales *Xsens Dot* con su base de carga.

Software

Los diferentes recursos *software* que se han necesitado a lo largo del proyecto son los siguientes:

- Sistema operativo *Windows 10 pro* para el entorno de trabajo en el ordenador personal.
- Sistema operativo *Raspbian (versión del 2022-04-04*) para la *RaspberryPi*.
- Entorno de desarrollo *SublimeText 3*.
- *Python 3.8.5* con las librerías requeridas en cada parte del proyecto, incluyendo *Pygame* y *PyOpenGL*.
- Instancia en *AWS* (*Amazon Web Services*) con sistema operativo *Ubuntu 18.04* y servidor web *Apache*.
- Servidor web funcionando con *Django*.
- *OpenSim 4.3* para el proceso de cinemática inversa en el procesamiento de los datos.
- *Photosop* para la creación de los diferentes elementos gráficos de la interfaz gráfica del bot.
- Videos y sesiones creados por los terapeutas para los pacientes.

En el *anexo 1* se mostrará un presupuesto aproximado de los costes de este proyecto.

1.3 Estructura de la memoria

En el *primer capítulo* de este documento se realiza una contextualización de la situación sanitaria española en materia de rehabilitación actual y se presenta una propuesta de proyecto que intenta dar una solución o apoyo a esa problemática.

Finalmente se hará un breve resumen de todos los recursos *hardware* y *software* empleados durante la realización del proyecto.

En el *segundo capítulo* se hace un detallado repaso de los pormenores de todas las tecnologías y conceptos involucrados en el proyecto. Desde el funcionamiento interno de los sensores *Xsens Dot* y cómo funciona el protocolo *BLE* para su conexión, pasando por diferentes librerías de *Python* importantes para el desarrollo del proyecto, hasta finalizar dando una breve introducción sobre la herramienta matemática de los cuaterniones y sus operaciones.

En el *tercer capítulo* de este documento hay un repaso de la literatura en busca de diferentes publicaciones que hayan tratado de abordar proyectos similares a los que se desarrollan en este proyecto sobre la telerehabilitación y sus conclusiones. Desde investigaciones con realidad virtual aplicada a la telerehabilitación, sensores vestibles aplicados a la telerehabilitación, hasta otros proyectos que hayan hecho uso de los mismos sensores inerciales *Xsens Dot*.

El punto final de este capítulo es fundamental para la comprensión del resto del documento, ya que se presenta brevemente el proyecto de telerehabilitación base sobre el que se va a partir para los desarrollos realizados en este trabajo.

El *cuarto capítulo* contiene todos los detalles de los desarrollos realizados a lo largo del trabajo en orden cronológico, este detalle es importante ya que muchos de los conocimientos adquiridos durante el desarrollo de un punto se aplican a puntos posteriores.

Finalmente, la última sección de este capítulo trata de recopilar todos los pequeños subproyectos y darles un orden más funcional para dar una visión general de todo el trabajo realizado.

En el *quinto capítulo* de este documento se exponen las conclusiones del trabajo realizado y las líneas futuras con mejoras y evolución que podrá tomar el proyecto en un futuro.

Por último, se incluye un pequeño *anexo* en el que se presenta un presupuesto aproximado de los costes del trabajo realizado.

Capítulo 2 Revisión de las tecnologías

2.1 Sensores Xsens Dot

Los sensores *Xsens Dot* son una de las gamas más económicas de sensores inerciales del fabricante *Xsens*, una de las empresas más conocidas en el mercado de la innovación tecnológica de sensores de movimiento ubicada en los Países Bajos.

Los productos *Xsens* son ampliamente conocidos por su uso en diversas películas y videojuegos para la recreación de movimiento sobre personajes virtuales. Han intervenido en películas tan conocidas como "*Los Vengadores*", "*Thor*", "*Batman Vs Superman*", entre otros.

Los sensores han sido diseñados como plataforma *hardware* para desarrolladores de todo el mundo, para realizar sobre ellos cualquier tipo de proyecto que involucre capturas de movimiento y que requieran el uso de sensores inerciales de pequeño tamaño.

Figura 1 : Sensores Xsens Dot en la base de carga.

Estos sensores incorporan en su interior varios módulos que se encargan de diferentes tareas. Por una parte, incorporan en su interior un acelerómetro capaz de registrar la aceleración angular y un giroscopio de precisión para medir la orientación del sensor utilizando los principios de la conservación del momento angular.

Con estos dos elementos básicos ya se podrían dar una medida bastante precisa de la orientación del sensor, el problema es que conforme pasa el tiempo, la precisión de esta medida se va degradando debido a un pequeño error acumulativo conocido

como *drift.* Este error se produce por pequeñas imprecisiones en el cálculo de los grados en las mediciones del giroscopio.

Para hacer frente a este error se incorpora un algoritmo propietario llamado *Xsens Kalman Filter core (XKFCore)*, una variación del conocido filtro de *Kalman* capaz de deducir y corregir el error de una medida a partir de las medidas anteriores, eliminando ese error acumulativo y mejorando la fiabilidad y precisión de los sensores.

Por otra parte, incorpora un magnetómetro capaz de medir la fuerza y dirección de los campos magnéticos que rodean al sensor. Esta información es utilizada para mejorar la precisión de las mediciones, utilizando como referencia absoluta el norte magnético de la Tierra.

Otra funcionalidad extra que posee este sensor, es la de construir una imagen de los campos magnéticos presentes a su alrededor, pero esta función se encuentra todavía en estado beta y no se encuentra disponible para los desarrolladores.

Como se especifica en el propio manual de usuario hay que tener precaución con el entorno donde se realicen las mediciones, ya que se pueden producir errores significativos si hay demasiados campos magnéticos externos. Estos campos suelen ser inducidos por elementos metálicos de gran tamaño presentes en entornos como fábricas, aunque no es descartable que en menor medida se puedan producir también en el entorno doméstico.

Finalmente, el sensor *Xsens Dot* tiene un procesador interno que se encarga de gestionar el muestreo de todos los módulos que contiene y del proceso de calibración. La arquitectura de procesamiento de los *Xsens Dot* se puede ver en la *Figura 2.*

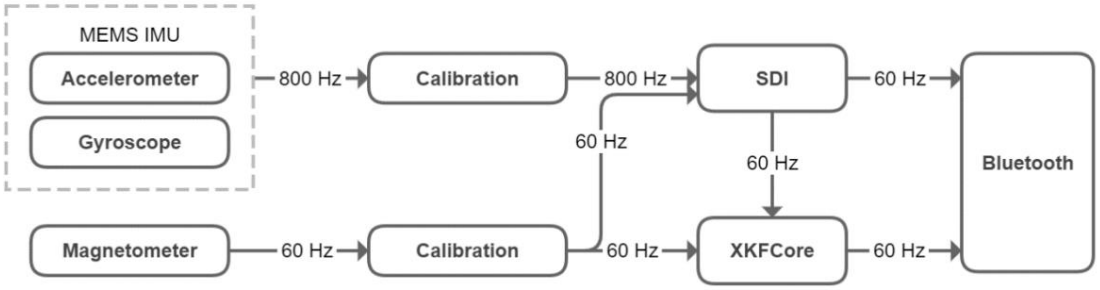

Figura 2 : Arquitectura de procesamiento de sensor Xsens Dot.

Como los datos procesados tendrán que ser transmitidos finalmente a través de un módulo *Bluetooth* y este protocolo no puede soportar la tasa de muestreo de 800 Hz que se reciben del acelerómetro y el giroscopio, se pasa por un módulo intermedio llamado *Strapdown Integration (SDI)* bajando la frecuencia recibida a 60 Hz que es la máxima tasa de transmisión en tiempo real.

Si se necesitara una tasa mayor de muestreo, los propios sensores *Xsens Dot* incorporan la capacidad de almacenar las muestras en la memoria interna que incluye el sensor y luego volcar los datos finalizada la captura [5].

2.2 Bluetooth Low Energy (BLE)

Bluetooth fue desarrollada en el año *1994* por *Bluetooth Special Interest Group* (SIG), una asociación privada de compañías tecnológicas sin ánimo de lucro en *Kirkland* (*Washington*). Esta tecnología perseguía el envío de ficheros de forma inalámbrica entre dispositivos con un consumo reducido.

En el año *2010* se presenta por primera vez la versión 4.0 del estándar *Bluetooth*, con nuevo estándar de funcionamiento conocida como *Bluetooth Low Energy*. Esta estaría pensada para la transmisión de pequeñas cantidades de datos entre dispositivos de bajo consumo como relojes inteligentes, aplicaciones domóticas, etc.

La principal prioridad de *BLE* es minimizar el consumo de la transmisión de datos, transmitiendo pequeñas cantidades sin la necesidad de mantener continuamente la conexión. Esto permite a los dispositivos que sólo necesiten estar activos cuando necesiten recibir o emitir datos, aumentando la autonomía de las baterías.

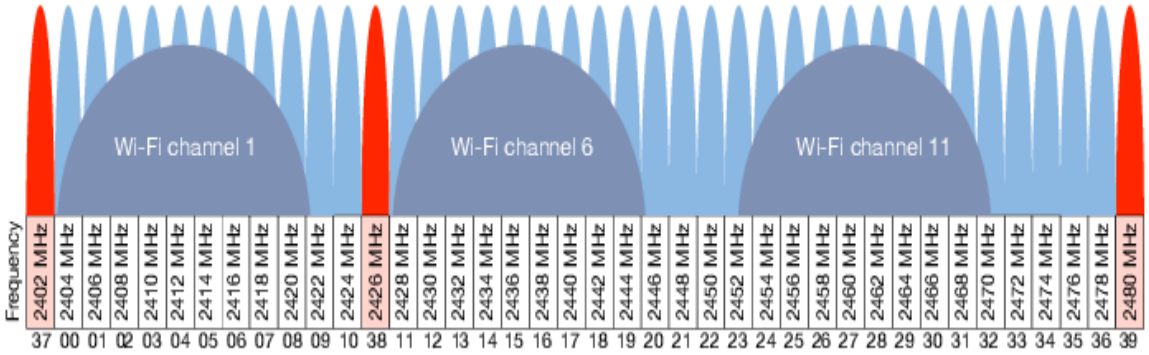

Figura 3 : Canales Bluetooth Low Energy.

Arquitectura

La arquitectura de las redes *BLE* son de tipo estrella, en este modo de conexión existen dos roles definidos:

- ***Periférico o esclavo***: Es el dispositivo que envía paquetes de forma periódica indicando su disponibilidad para la conexión y una vez establecida dicha conexión intercambiará datos con el dispositivo central o maestro.
- ***Central o maestro***: Se encarga de buscar los dispositivos disponibles a través de los paquetes de anuncio de disponibilidad y una vez lograda la conexión comienza el intercambio de datos.

El grupo *SIG* define para *Bluetooth* una pila de componentes dividida en tres partes básicas:

- ***Controlador***: Es el dispositivo físico que se encarga del envío y recepción de las señales de radio interpretando cada paquete de información. Se distinguen en este componente tres capas:
    - Capa física: Usa la banda de 2.4GHz con 40 canales con separación de 2MHz y modulación *GFSK* (*Gaussian Frequency Shift Keying*) a 1Mbps.
    - Capa de enlace: Encargada de la estructura de las tramas y el control de los diferentes estados de conexión:
        - *Advertising*: Envía paquetes de aviso de disponibilidad a los canales reservados para esta función específica.
        - *Standby*: Modo de espera, sin transmitir ni recibir.
        - *Scanning*: Escucha paquetes de disponibilidad en los canales de *advertising*.
        - *Initiating*: Inicia la conexión con el dispositivo.
    - Interfaz de controlador del host: Esta capa define la conexión entre *host* y *controller* mediante comandos *HCI*.
- ***Host***: *software* que hace de capa intermedia entre la *API* final a la que puede acceder el desarrollador y la interfaz del controlador del host con comandos *HCI*. Además, se encarga de funciones de seguridad y gestión de los dispositivos conectados.
- ***Aplicaciones***: aplicación final desarrollada que utiliza la *API*.

En la *Figura 3* se puede ver la distribución del espectro en la banda de 2.4GHz para la tecnología BLE, que comparte espectro con los canales de *802.11*. Las bandas marcadas en rojo serán las reservadas para *Advertising*.

Perfiles BLE

El *stack* de protocolos de la tecnología *Bluetooth* define como debe ser gestionada la información para la transmisión de los datos de forma eficaz, teniendo en cuenta aspectos como la codificación y decodificación de los paquetes, los diferentes formatos de trama y la multiplexación [6].

Los perfiles definen como van a ser usados estos protocolos en conjunto y existen dos grupos de perfiles:

- ***Perfiles genéricos***
    - *General Access Profile* (*GAP*): Se encarga de definir los roles, modos y procedimientos que realizan las funciones de establecimiento y gestión de la conexión.
    - *General Attribute Profile* (*GATT*): Se encarga de definir los procedimientos básicos para enviar y recibir información.
- ***Perfiles específicos***
    - Perfiles enfocados a un servicio específico elaborado por terceros para cubrir necesidades particulares y generalmente basados en *GATT*.

2.3 Pygame

Pygame es una librería de creación de videojuegos que está basada en *SDL (Simple DirectMedia),* que son librerías en lenguaje *C++* para la gestión de gráficos *2D*, imágenes, audios y periféricos a bajo nivel.

Pygame fue creado por *Sam Lantinga* buscando la forma de simplificar la tarea de portar juegos de una plataforma a otra, de modo que fuera posible implementar la interfaz gráfica del juego una única vez y que se pueda ver de la misma manera en múltiples plataformas.

Debido a la facilidad de uso, se convirtió en una plataforma muy popular entre los desarrolladores de videojuegos desde el año *1998* que saldría a la luz. Aunque *Pygame* ha quedado en la actualidad relegado a un segundo plano debido a la salida al mercado alternativas más potentes, todavía se sigue utilizando en la comunidad para la creación de juegos sencillos y que no requieran muchos recursos.

Pygame se compone de un conjunto de módulos de *Python* diseñados para la creación de interfaces gráficas de código abierto bajo la licencia *LGPL (GNU Lesser General Public License)*. Esta estructura modular permite la utilización por separado cada una de sus funcionalidades de manera absolutamente independiente, ya que hay un módulo para cada uno de los dispositivos que se necesite utilizar en el juego y todos ellos son accesibles desde el espacio de nombres de *pygame.*

Una de las motivaciones más importantes para escoger librería *Pygame* para el desarrollo de esta herramienta es que, al igual que *Python*, su principal filosofía es compaginar la potencia, la flexibilidad y sobre todo la sencillez. Motivo por los que *Pygame* ha sido ampliamente utilizada en el ámbito académico para el aprendizaje [7].

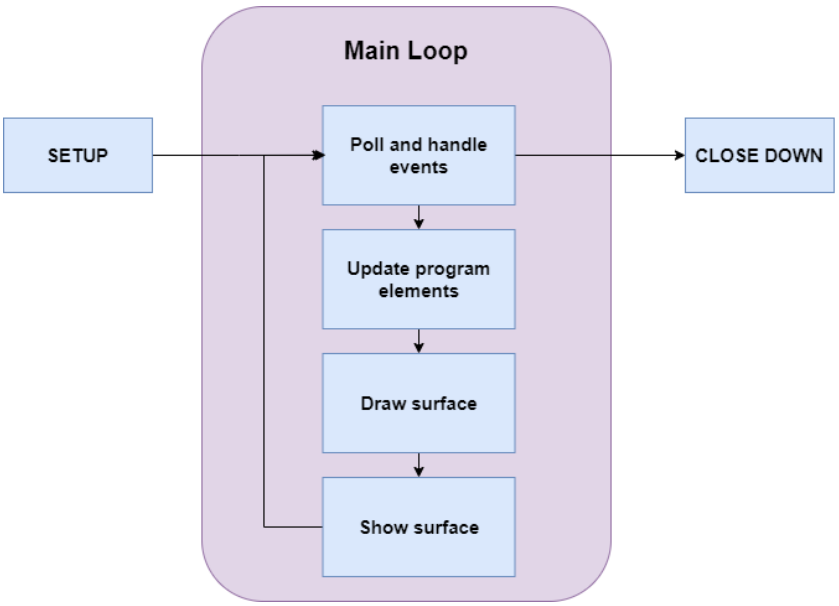

Figura 4 : Flujo de ejecución normal de Pygame.

La estructura básica de cualquier programa construido sobre *Pygame* como se puede ver en la *Figura 4*, consta principalmente de tres etapas:

- **Configuración**: En esta etapa se ejecutará solo al iniciar el programa y se encargará de crear la ventana principal, realizar las configuraciones necesarias de los parámetros de ejecución e iniciar el entorno de *Pygame* para que las funcionalidades de todos los módulos que han sido importados sean accesibles.
- **Bucle principal**: El programa permanecerá en este bucle siempre que el programa esté en ejecución. Durante el bucle principal se utilizan generalmente las siguientes cuatro funciones:
    - Comprobación de los eventos ocurridos por teclado, ratón, etc.
    - Actualización de los datos internos de los objetos y realiza la lógica de la aplicación.
    - Dibujado el estado actual de la pantalla en el *buffer* construyendo la escena (este buffer todavía no será visible).
    - Dibujado del contenido del *buffer* en la ventana principal.
- **Cierre**: Esta etapa solo se ejecutará al finalizar el programa y se encargará de que no queden estados inestables en el programa para poder salir con seguridad.

Con *Pygame* se pueden modificar directamente de forma sencilla escenarios en tres dimensiones, con una gran flexibilidad para la manipulación de imágenes, figuras o píxeles en tiempo real. Además también proporciona la capacidad de actuar como una capa de visualización multiplataforma para *PyOpenGL*, característica que se utilizará en el proyecto en la *sección 4.4*.

2.4 PyOpenGL

PyOpenGL es un conjunto de módulos de *Python* que brinda acceso a las funcionalidades de *OpenGL*, así como a una variedad de utilidades auxiliares y extensiones para completar la interfaz a bajo nivel.

OpenGL (*Open Graphics Library*) es una *API (Application Programming Interface)* multiplataforma y multilenguaje, para la creación de gráficos vectoriales en 2D y 3D. Esta *API* se utiliza generalmente para interactuar con una *GPU* y lograr una representación acelerada por *hardware*. Además, viene instalada por defecto en la mayoría de las plataformas usualmente como parte de sus *drivers* gráficos.

Al utilizar *PyOpenGL*, el módulo de *OpenGL* toma el control de lo mostrado en pantalla ya que *Pygame* solo estará actuando como capa de visualización, aunque se pueden seguir utilizando el resto de módulos disponibles como control de tiempo, entrada por teclado, sonido, etc. [8].

2.5 Introducción a OpenSim

OpenSim es un *software* de código abierto para modelado, simulación y análisis biomecánico. Este *software* ha sido desarrollado por *Simbios*, un centro de computación biomédica de la universidad de *Stanford* fundada en el *2004*. El propósito de este centro es proporcionar un marco común para la investigación y un vehículo para el intercambio de modelos musculoesqueléticos complejos.

OpenSim permite crear, intercambiar y analizar modelos digitales de un sistema musculoesquelético, para generar dinámicas de movimiento y estudiar la coordinación neuromuscular, analizar el rendimiento deportivo o estimar las cargas musculoesqueléticas.

Desde el lanzamiento inicial en *2017* este *software* ha sido utilizado por miles de personas para una gran variedad de aplicaciones como biomecánica, diseño de dispositivos médicos, ortopedia, rehabilitación, educación, etc. *OpenSim* proporciona una plataforma en la que la comunidad dedicada a la biomecánica puede crear simulaciones para intercambiar, probar, analizar y mejorar a través de la colaboración multiinstitucional.

Esta herramienta está escrita principalmente en *C++* y su interfaz gráfica de usuario (*GUI*) cuya apariencia se muestra en la *Figura 5*, está escrita completamente en *Java*.

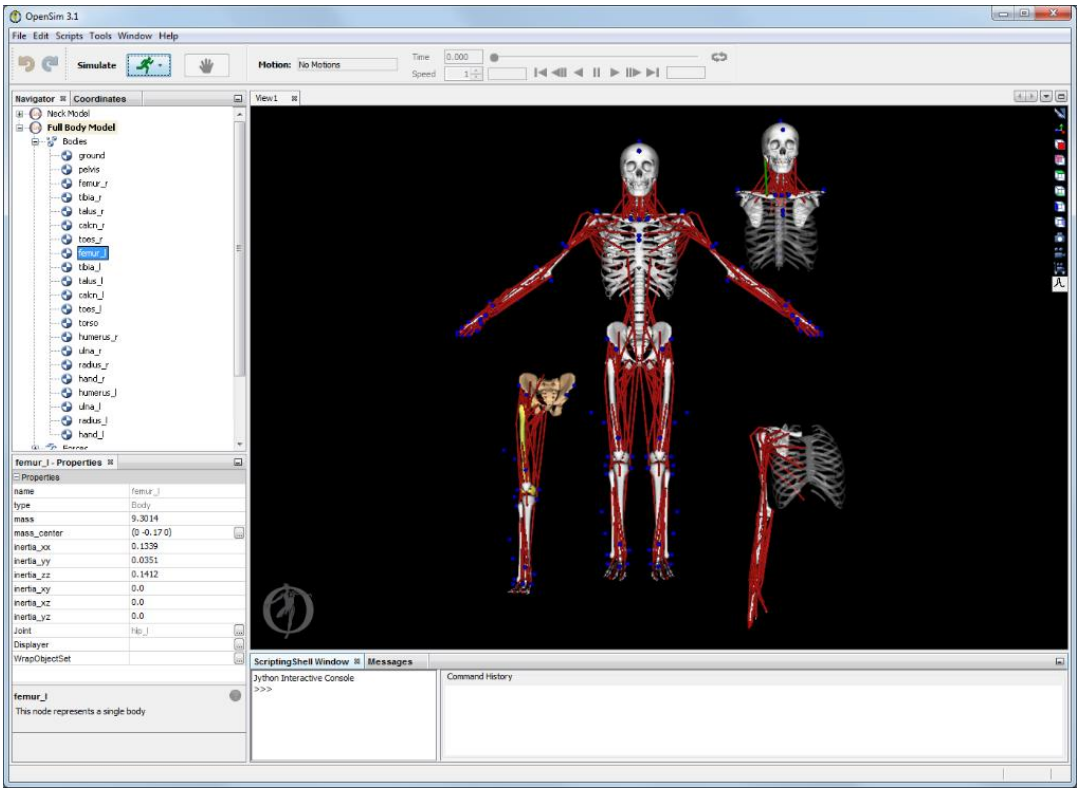

Figura 5 : Interfaz gráfica de OpenSim.

OpenSim además, está diseñado para que sea posible desarrollar *plugins* sobre él de forma que es posible la creación de controladores personalizados, módulos de análisis, modelos de contacto o modelos musculares sin necesidad de modificar ni de compilar el código fuente de nuevo.

Un modelo musculoesquelético en *OpenSim* consta de una serie de segmentos de cuerpo rígidos conectados por articulaciones entre sí. Sobre estos estarán los músculos que abarcarán las articulaciones y generarán fuerzas y movimiento en el modelo.

Una vez que se crea un modelo musculoesquelético *OpenSim* permite al usuario crear análisis personalizados, como la comprobación de los efectos de la geometría musculoesquelética, la cinemática articular o las propiedades músculo-tendinosas sobre las fuerzas y los momentos articulares que pueden producir los músculos [8] [9].

Aunque puedes crear tu propio modelo desde cero configurando cada uno de los parámetros. *OpenSim* ya contiene de base un conjunto de modelos ampliamente variados para las extremidades superiores e inferiores. Además, existen modelos simplificados para la creación de prototipos enfocados en la enseñanza, a partir de los cuales se pueden crear modelos propios.

Una de las características más relevantes para el proyecto que se pretende realizar llega a partir de la versión *3.0* de *OpenSim*, momento donde se incluye una *API* para *Python*. El propósito fundamental de esto, es que desarrolladores de todo el mundo puedan crear sus propios *scripts* para automatizar procesos de análisis pudiendo acceder directamente desde el propio *script* a todas las funcionalidades y parámetros internos de *OpenSim*.

2.6 ChartJs

Para graficar los datos recogidos de los pacientes se necesita una librería que sea sencilla de utilizar, pero con la que se consiga resultados vistosos y que den el aspecto profesional necesario.

Librerías de creación de gráficos para web

- ***ChartJs***: Permite de forma fácil y sin dependencias crear hasta seis tipos de gráficos distintos. La librería es modular para que no se cargue todo lo que no se va a utilizar.
  Es compatible con el diseño responsivo y es muy personalizable incluyendo colores y animaciones con solo añadir un módulo.
  Además, cuenta con una comunidad grande detrás que ayuda al desarrollo de aplicaciones con esta herramienta.
- ***ChartistJs***: Librería para crear gráficos que utilicen *SVG*. Este una gran flexibilidad con una clara separación entre el componente de estilo *CSS* y el de control *JavaScript*. Ofrece gran flexibilidad con un sinfín de opciones de animación de gráficos.

- ***C3 Js***: Librería para crear gráficos basados en *D3 JS*, viene además con gran variedad de *APIs* que se pueden utilizar para controlar los gráficos y darle un aspecto más dinámico.
- ***Flot***: *Plugin* de *jQuery* para crear gráficos con elementos interactivos como puntos de gráficas, paneos, zooms, etc. Las gráficas funcionarán perfectamente en aquellos navegadores que soporten *canvas* de *HTML*.

Al contrario de lo que sucede con la librería *C3Js* que es la librería más utilizada hasta la fecha para la creación de gráficas, *ChartJs* presenta una mayor simplicidad en la implementación manteniendo unos resultados profesionales y responsivos. Por eso finalmente se ha decidido utilizar la librería *ChartJs* como base para el proceso de graficado de los resultados en la web.

Introducción a ChartJs

ChartJs es una de las librerías más populares de visualización y además una de las más fáciles de utilizar, completamente de código abierto y ampliamente soportada por una gran comunidad de desarrolladores a través de *GitHub*. Con muy poco código se puede crear una interfaz completamente interactiva para la creación de gráficas a través del uso de *JavaScript*.

Solamente será necesario cargar los datos dentro de un *array* de *JavaScript* y después se pueden añadir estilos y otras configuraciones de forma simple con programación basada en objetos muy sencillos de utilizar.

ChartJs automáticamente se encarga del escalado de los datos, genera puntos y líneas de referencia e integra todos los elementos de forma totalmente responsiva adatándose a cualquier espacio asignado.

Esta librería admite hasta ocho tipos de gráficas diferentes y todas completamente interactivas: gráficos lineales, gráficos de barras, gráficos radiales, gráficos circulares y gráficos de sectores.

La creación de gráficas con esta herramienta resulta muy simple, primero es necesario importar la propia librería con los módulos de la comunidad que se van a utilizar, en este caso se han utilizado dos módulos muy comunes: el primero será *chartjs-plugin-zoom* que permitirá alejarse y acercarse a los gráficos para ver los detalles y el segundo será *chartjs-plugin-colorschemes* que mejora la presentación añadiendo colores aleatorios a cada entrada.

Una vez realizadas todas las importaciones, solamente se necesitará un pequeño *script* para instanciar el objeto tipo *Chart* con todas las configuraciones de presentación y una etiqueta *"<canvas>"* al que se asociará a la instancia anterior para que se genere la propia gráfica. En este elemento *ChartJs* se encargará automáticamente del control, reescalado y dibujado, manteniendo las proporciones para adaptarse al espacio asignado [10].

2.7 Cuaterniones

Los cuaterniones fueron descubiertos en *1843* por el matemático irlandés *Sir William Rowan Hamilton*, con el objetivo inicial de crear un tipo de números hipercomplejos relacionados en un espacio tridimensional, de la misma forma en la que los números complejos tradicionales se relacionan sobre el plano.

Con el conocimiento de los números complejos y el plano complejo, se puede extender esto a un espacio tridimensional, agregando dos números imaginarios más al sistema además del *i* ya conocido. De tal manera que el cuaternión se podrá expresar como aparece en la *ecuación 1* siguiente.

$$q = w + xi + yj + zk \quad w,x,y,z \in \mathbb{R} \tag{1}$$

Siendo, según la expresión de *Hamilton*:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2}$$

$ij = k$ $\quad$ $jk = i$ $\quad$ $ki = j$

$ji = -k$ $\quad$ $kj = -i$ $\quad$ $ik = -j$

Hamilton utiliza los números imaginarios *i, j* y *k* para representar tres vectores unitarios cartesianos con las mismas propiedades de los números imaginarios, de este modo las operaciones aritméticas elementales pueden tener una interpretación geométrica natural.

Estas nuevas relaciones se pueden utilizar para representar rotaciones de objetos en tres dimensiones al igual que se utiliza la relación del plano complejo en los rotores para representar giros en el plano.

Existen varias formas de notación para el trabajo con cuaterniones, la forma más comúnmente utilizada en el campo de las matemáticas sería representar los cuaterniones como un par ordenado de un número real y un vector de posición en el espacio de los complejos:

$$q = [w, \boldsymbol{v}] \in \mathbb{R}, \boldsymbol{v} \in \mathbb{R}^3 \tag{3}$$

Donde $\boldsymbol{v}$ puede estar representada también como sus componentes individuales y de esta manera se pueden ver mejor las relaciones entre los números complejos y los cuaterniones:

$$q = [w, xi + yj + zk] \qquad w,x,y,z \in \mathbb{R} \tag{4}$$

Otra notación muy extendida en el ámbito de la computación y análoga a la anterior para la representación de cuaterniones sería la siguiente:

$$(w, x, y, z) = [w\ (x\ y\ z)] \tag{5}$$

De forma que corresponderían respectivamente a las componentes de las bases *{1, i, j, k}* y con esto se consigue que la notación sea mucho más clara y compacta.

Comparado con los ángulos de *Euler*, los cuaterniones ofrecen una representación de la orientación de un objeto de una forma más compacta y más rápida en lo que se refiere a la velocidad de cómputo.

Además, la representación con cuaterniones al contrario que con ángulos de Euler permite evitar un fenómeno conocido como "*gimbal lock*", que desemboca en una pérdida de grados de libertad ocurrida cuando dos o más ejes de giro acaban colocándose en paralelo. Un ejemplo de este fenómeno se puede apreciar mejor en la *Figura 6*, donde al realizar un giro sobre el eje *x* se provoca la pérdida de uno de los grados de libertad [11].

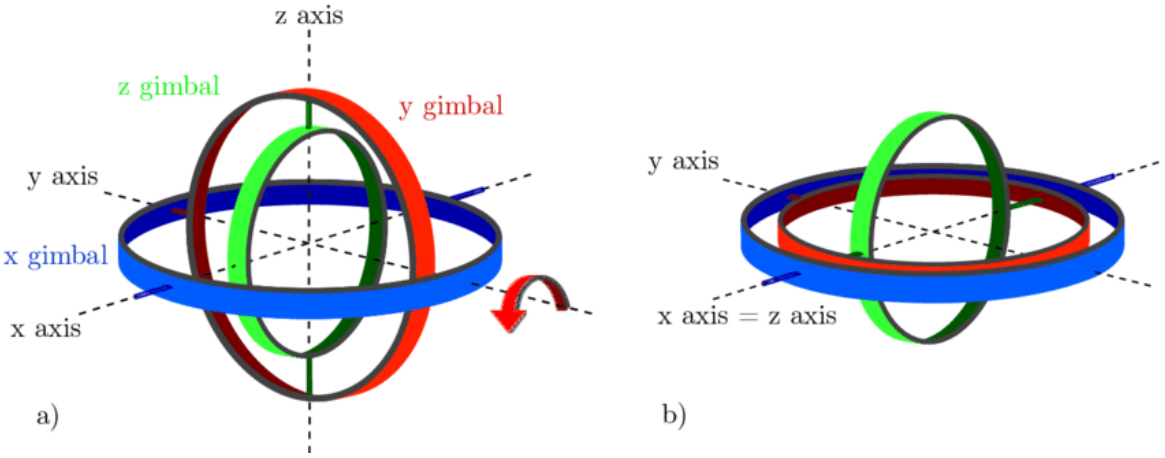

Figura 6 : Ejemplo de efecto Gimbal Lock a la derecha.

Conjugada e inversa de cuaterniones

El conjugado de un cuaternión representado por q^* se obtiene multiplicando por ***-1*** el vector de posición del cuaternión ***v***.

$$q^* = [w\ -\boldsymbol{v}] = [w\ \ (-x\ -y\ -z)] \tag{6}$$

La inversa del cuaternión que se denominará q^{-1} representará la rotación inversa, es decir el giro en sentido contrario al del cuaternión original y se define como se muestra en la *ecuación 7*.

$$q^{-1} = \frac{q^*}{q * q^*} = \frac{q^*}{|q|^2} \tag{7}$$

La inversa de los cuaterniones tiene una interesante correspondencia con la multiplicación de los números reales ya que al multiplicar un cuaternión por su inversa el resultado es el cuaternión identidad, es decir el cuaternión que representaría el giro nulo.

$$q * q^{-1} = \frac{q * q^*}{|q|^2} = \frac{|q|^2}{|q|^2} = (1,0,0,0) \qquad (8)$$

Multiplicación de cuaterniones

El resultado de la multiplicación de cuaterniones es similar al producto vectorial para vectores, sin embargo el resultado final será otro cuaternión y no un escalar. La definición del producto de cuaterniones, también conocido como el producto de *Hamilton* se deduce fácilmente como se indica en la siguiente *ecuación 9*.

$$\begin{aligned} q_1 q_2 &= [w_1\ (x_1\ y_1\ z_1)][w_2\ (x_2\ y_2\ z_2)] = [w_1\ \boldsymbol{v}_1][w_2\ \boldsymbol{v}_2] \\ q_1 q_2 &= [w_1 w_2 - \boldsymbol{v}_1.\boldsymbol{v}_2\ \ w_1\boldsymbol{v}_2 + w_2\boldsymbol{v}_1 + \boldsymbol{v}_1\ x\ \boldsymbol{v}_2] \end{aligned} \qquad (9)$$

Algunas de las propiedades de esta operación son:

- El producto de cuaterniones es asociativo, pero no conmutativo.

$$ab \neq ba$$

- El módulo del producto de cuaterniones es igual a el producto de sus módulos.

$$|q_1 q_2| = |q_1||q_2|$$

- La inversa de un producto de cuaterniones es igual al producto de los cuaterniones inversos en orden inverso.

$$(ab)^{-1} = b^{-1}a^{-1}$$

En general, a la hora de trabajar con cuaterniones para la representación de rotaciones estaremos siempre dentro del conjunto de los denominados cuaterniones unitarios con módulo uno. De esa manera nos aseguraremos que el resultado de la operación producto también sea unitaria independientemente del giro representado y ayuda a corregir las pequeñas imprecisiones de decimales en los cálculos.

$$|q_1 q_2| = |q_1||q_2| = 1 \qquad (10)$$

Respecto a la interpretación del producto en función de las rotaciones de objetos en espacios tridimensionales, se puede interpretar que el producto de q_1q_2 de las siguientes maneras:

- Un giro de q_1 en el sistema de referencia ***global*** del giro q_2.
- Un giro de q_2 en el sistema de referencia de coordenadas ***local*** de q_1.

"Diferencia" de cuaterniones

Haciendo uso de la multiplicación y la inversa, se puede calcular la diferencia entre dos cuaterniones, entendiendo la diferencia como el desplazamiento angular de una orientación a otra. En otras palabras, si tenemos dos cuaterniones ***a*** y ***b***, la diferencia que se denotará con ***d*** será el desplazamiento angular que hay que realizar desde la posición ***a*** a la posición ***b***.

$$da = b \qquad (11)$$

Partiendo de la *ecuación 11*, se despeja el cuaternión diferencia llegando a la siguiente expresión:

$$d = ba^{-1} \qquad (12)$$

Matemáticamente, el concepto de diferencia angular entre dos cuaterniones sería lo más similar conceptualmente a la operación de resta en números reales [12].

Aunque quizás parezca una introducción demasiado detallada sobre los cuaterniones y sus propiedades, estos van a ser de gran utilidad más adelante donde serán la base para algoritmos en diferentes partes del proyecto.

Los cuaterniones son la herramienta más utilizada en el desarrollo de *software* a la hora de representar rotaciones en espacios de tres dimensiones y facilitar el trabajo a la hora de operar con ellas. Un ejemplo claro de uso son los motores de desarrollo de videojuegos como *Unity* o *Unreal Engine,* que trabajan íntegramente con cuaterniones para las rotaciones espaciales.

Capítulo 3 Trabajos Previos

3.1 Estado del arte

Antes de centrarnos en la exposición del desarrollo de nuestro proyecto siempre resulta interesante la realización de un pequeño repaso por las tecnologías actuales en el área de interés, en este caso el ámbito de aplicaciones médicas y más concretamente en aplicaciones relacionadas con la de la telerehabilitación.

A continuación, se presentarán varios estudios y productos actuales que presentan ciertas similitudes con nuestro proyecto y que servirán de un punto de partida para comprender el contexto en el que se va a desarrollar el trabajo.

Realidad virtual aplicada a la telerehabilitación

El centro de investigación de ingeniería de rehabilitación de la *universidad del sur de California* (*NIDRR*), demostró los beneficios de la utilización de juegos de realidad virtual para ayudar en los procesos de telerehabilitación. Utilizaron pequeños desafíos a abordar dentro de los juegos aumentando notablemente el cumplimiento y motivación del paciente durante el proceso de rehabilitación.

El *NIDRR* aprovecha la tecnología de realidad virtual para mejorar las habilidades motoras en sujetos con problemas de movilidad. En este estudio se ha llegado a la conclusión de que cuando los ejercicios se proponen como un juego interactivo existe un potencial para incentivar el compromiso y la motivación necesarios para impulsar el mantenimiento y la mejora del sistema motor [13].

Otro proyecto en torno a la realidad virtual aplicada a la telerehabilitación trata de combinar ambas tecnologías permitiendo al terapeuta supervisar de forma remota el entrenamiento de los pacientes y dar consejos importantes durante el entrenamiento dinámico.

En este estudio participaron seis pacientes que habían sufrido un accidente cerebrovascular. Durante dos semanas los pacientes realizarían ejercicios de entrenamiento del equilibrio en un entorno clínico y durante una semana los realizarían en un entorno doméstico, cinco veces por semana por un máximo de 20 minutos diarios.

El entrenamiento consistía en mantener el equilibrio con los pies situados sobre una plataforma en función de los obstáculos que se iban presentando por pantalla, en la *Figura 7* se puede ver mejor de que trataba este sistema.

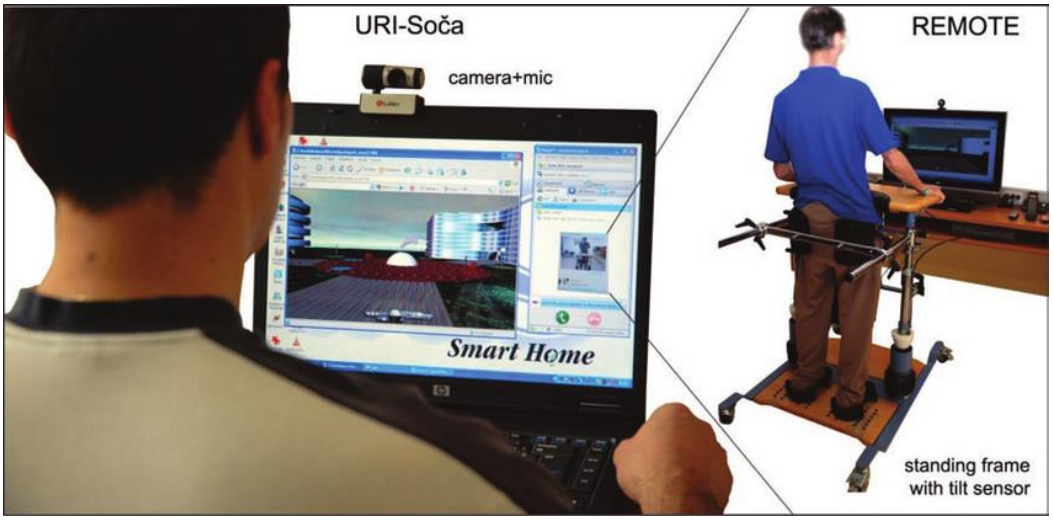

Figura 7 : Entrenamiento del equilibrio con realidad virtual.

Cuando se evaluaron los resultados, no se encontraron diferencias estadísticas significativas entre el enfoque de telerehabilitación con realidad virtual y el entrenamiento con ejercicios de equilibrio tradicionales en el entorno clínico.

En ambos casos se consiguieron mejoras en el equilibrio de los pacientes. Sin embargo, es importante notar que cuando el entrenamiento se realiza de forma telemática en lugar de presencial en el propio hospital tiene un efecto positivo en la saturación hospitalaria, reduce los costes del tratamiento y permite el tratamiento a un mayor número de pacientes simultáneamente [4].

Sensores vestibles aplicados a la telerehabilitación

Uno de los productos con sensores vestibles del mercado más interesantes y que tiene una mayor similitud con lo que se pretende desarrollar en nuestro proyecto, es el sistema *Valedo* desarrollado por la empresa *Hokoma*, una conocida empresa en el ámbito de las tecnologías de rehabilitación.

El sistema *Valedo* como se puede ver en la *Figura 8*, es un dispositivo médico enfocado en el tratamiento dolencias en la espalda baja, con una perspectiva de juego interactivo con retroalimentación basada en los movimientos del cuerpo en tiempo real, que consigue una mayor motivación y cumplimiento por parte del paciente.

Se trata de una combinación de sensores de movimiento inalámbricos con pequeños juegos interactivos sobre una televisión, como una forma de realizar ejercicios terapéuticos que ayuden a mejorar los problemas de espalda.

Con la información recogida el sistema dará al usuario indicaciones de si está realizando de forma incorrecta alguno de los ejercicios e intentará guiarlo durante el proceso para corregir los errores de posición.

Los pacientes pueden establecer objetivos en la terapia y recibir comentarios sobre su desempeño, además de realizar el seguimiento de su progreso semanal.

Los ejercicios se pueden ajustar de acurdo con las necesidades específicas del paciente, facilitando el desafío y logrando un entrenamiento más eficiente.

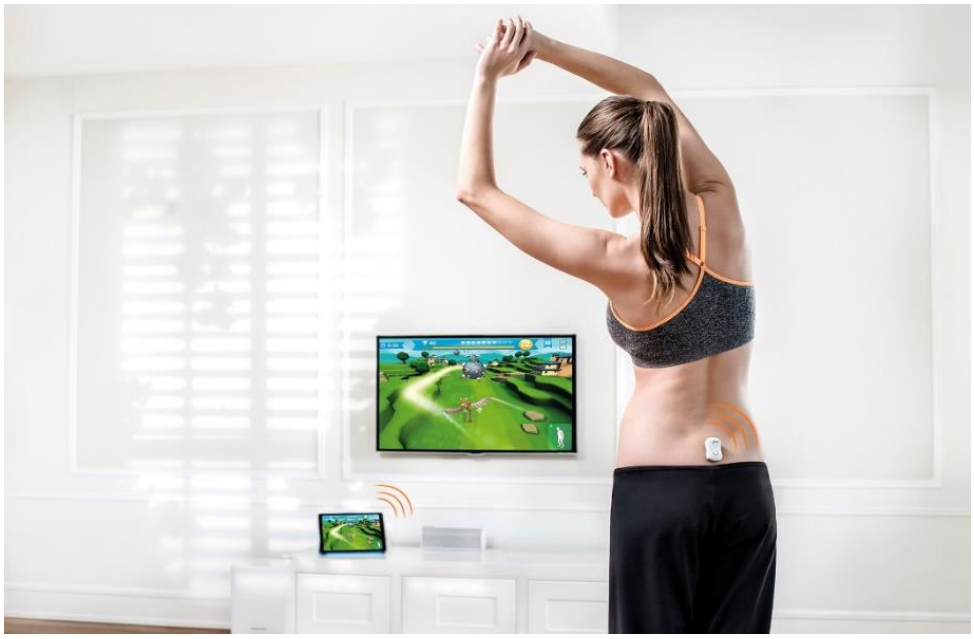

Figura 8 : Sistema Hokoma para ejercicios de espalda baja.

Al igual que en nuestro proyecto, los usuarios acaban generando un informe que puede ser revisado por un médico o fisioterapeuta para evaluar los progresos a lo largo del tiempo. Sin embargo, este servicio no viene incluido en el precio del propio producto [14].

Otro interesante proyecto actualmente en desarrollo, que además incorpora un sistema para recoger simultáneamente datos de movimiento y fisiológicos es el llamado *Stroke Rehab Exerciser* desarrollado por *GE Healthcare*, empresa dedicada a la innovación en tecnología de aplicaciones médicas de diagnóstico.

El nuevo sistema de monitoreo se encarga de recopilar datos fisiológicos y de movimiento, lo que facilitará asignar ejercicios más específicos al paciente y adecuarse a las necesidades de forma más precisa.

Este sistema se enfoca en una secuencia de ejercicios para el sistema motor prescritos por el fisioterapeuta y cargados en la unidad del paciente. Entonces, un sensor inercial inalámbrico registrará los movimientos del paciente, analiza los datos en busca de posibles desviaciones y proporcionará información al paciente y al terapeuta sobre los ejercicios realizados [15].

La universidad de *Aalborg* en Dinamarca, ha estado trabajando en el proyecto *TeleKat*, un proyecto de investigación e innovación que se centra en el desarrollo de nuevos métodos de atención y tratamiento para pacientes crónicos a través del uso de tecnologías de teleasistencia.

El objetivo de este proyecto es permitir a los pacientes con enfermedad pulmonar obstructiva crónica la posibilidad de realizar un autocontrol de su estado y mantener las actividades de rehabilitación desde sus propios hogares.

En este proyecto se instala una caja de monitor de telesalud en la casa del paciente y mediante sensores inalámbricos el monitor de telesalud puede recopilar y transmitir datos como presión arterial, pulso, nivel de oxígeno, etc. al personal sanitario [16].

Utilización de sensores Xsens Dot

Tras realizar una revisión de publicaciones y estudios que incluyeran el uso de los sensores *Xsens Dot* para la monitorización de movimientos, no se ha encontrado la utilización de ningún tipo de *script* a bajo nivel para la interacción directa con los servicios *Bluetooth* como se pretende hacer en nuestro proyecto, en todos los casos el uso de estos sensores se ha realizado a través de un dispositivo móvil.

Algunos de estos trabajos utilizaron directamente la propia aplicación de *Xsens Dot* disponible en *Google Store*, por ejemplo, una publicación del *hospital clínico de La Florida* en Chile que utiliza los datos de los sensores a través de la propia aplicación para la monitorización cardiorrespiratoria del paciente en el domicilio.

La monitorización consistió en un pequeño sensor *Xsens Dot* en el tórax del paciente que recogía las pequeñas aceleraciones producidas durante la respiración. Posteriormente el filtrado y tratamiento de los datos se haría a través de *MATLAB*, realizando un remuestreado de la captura y pasando los datos por un filtro paso banda para obtener los datos producidos por la actividad respiratoria y eliminar los que son producidos por otras actividades [17].

Otro ejemplo de utilización de la propia aplicación *Xsens Dot* sería un artículo publicado este mismo año *2022* por un grupo de investigación de la *universidad de Nottingham* en Reino Unido.

En este artículo se basó en un sistema de reconocimiento de actividades complejas de la vida diaria, a través de la utilización de varios sensores *Xsens Dot* conectados por su aplicación a un teléfono *Android* y la utilización de *Deep Learning* para el tratamiento de los datos y la identificación de cada una de las actividades.

En este estudio no solo se utilizaron los datos de cuaterniones, sino también los de las aceleraciones registradas para reconocer actividades de la vida diaria evaluando el rendimiento del uso de 1, 3 y 5 sensores en el cuerpo en redes *CNN (Convolutional Neural Networks)* y *LSTM (Long Shot-Term Memory Networks)*. Los resultados muestran que con 5 sensores y el modelo *LSTM* se consigue una puntuación de 0.9606 de precisión [18].

Por otro lado, algunos estudios desarrollaron pequeñas aplicaciones sobre el *SDK (Software Development Kit)* de *Xsens* publicada en su página web, por ejemplo, un estudio publicado por la *universidad libre de Ámsterdam*, que trataba de recoger mediciones en pacientes con distonía afectados por parálisis cerebral y realizar un seguimiento domiciliario de su evolución a través de los sensores *Xsens Dot*.

Estas mediciones se realizaron con cuatro sensores situados uno en cada extremidad, conectados a un terminal móvil desde dónde se recogían los datos a través de una aplicación desarrollada por ellos mismos.

A continuación, esos datos eran recogidos y procesados a través de *MATLAB* para desarrollar modelos de aprendizaje automático mediante validación cruzada que fueran capaces de dar una valoración del grado de distonía de cada paciente [19].

De igual manera, en *2021* un grupo de la *universidad de ciencias aplicadas de Niederrhein* en Alemania publicó un estudio sobre la creación de un sistema de capturas de movimiento con sensores inerciales de bajo costo a través del móvil.

En este caso se recogen los datos de los sensores *Xsens Dot* a través de una aplicación desarrollada con el *SDK* de *Xsens*. Esta se encargaría de la recolección de los datos de los sensores y el establecimiento de las proporciones del cuerpo que se va a emular. El aspecto de dicha aplicación se puede ver en la *Figura 9*.

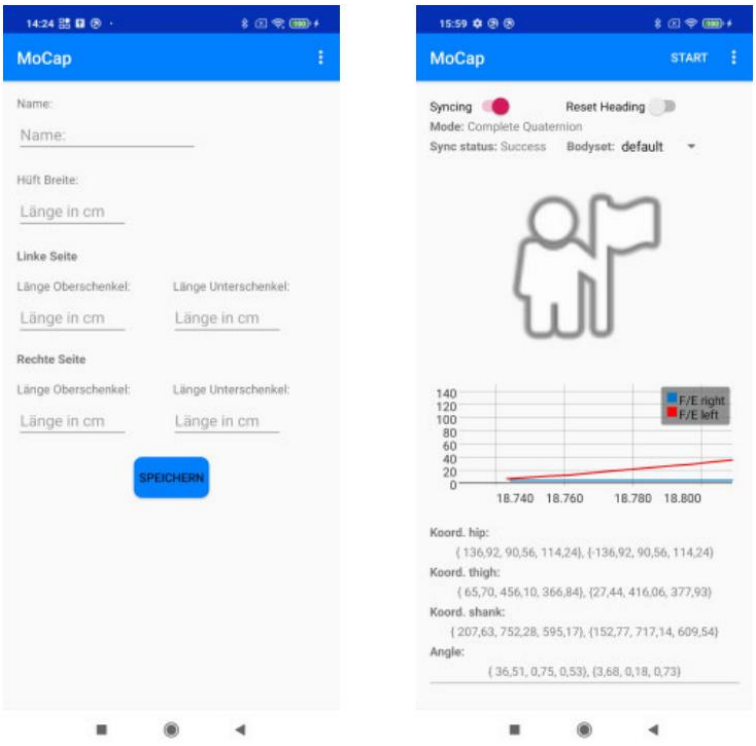

Figura 9 : Aplicación para el cálculo de ángulos con el uso de IMUs Xsens Dot.

Una vez recogidos los datos de los sensores se aplican a un modelo simplificado para hallar diferentes ángulos en las articulaciones, al igual que en el caso de nuestro proyecto, pero sin realizar una representación en tres dimensiones del modelo.

La conclusión a la que llega este grupo de investigadores es que aunque no se puede llegar a conseguir un nivel de precisión igual a de los sistemas comerciales más caros, la precisión alcanzada sí es suficiente para el desarrollo de aplicaciones de investigación asumiendo los márgenes de imprecisión [20].

3.2 Proyecto base para de desarrollo

Todos los desarrollos que se han realizado en el presente trabajo y que serán descritos en profundidad en el *capítulo 4*, parten del proyecto desarrollado en el trabajo de fin de grado de *Javier Delgado Rodríguez* en *2021*, titulado "*Sistema de tele-rehabilitación a domicilio con chatbot y microordenador de bajo coste*" [21].

Este proyecto ha continuado en desarrollo este mismo año *2022* en manos de *Óscar Martín Casares* en su *TFG* titulado "*Evolución de sistema médico rehabilitador con chatbot conversacional en Telegram, backend en Django y ordenador de bajo coste*" [22].

A continuación, se realizará una breve presentación de la infraestructura del proyecto sobre el que se va a trabajar, para esta explicación se utilizará como referencia el esquema de la *Figura 10*.

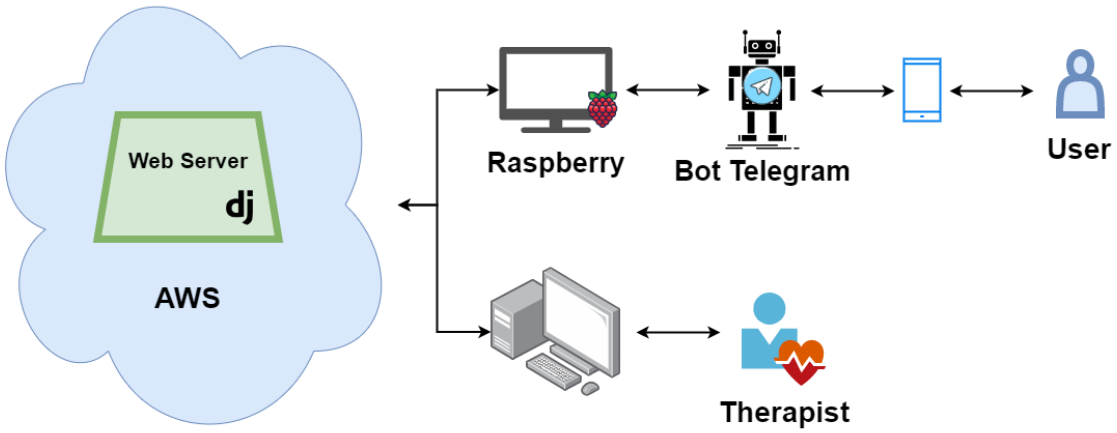

Figura 10 : Infraestructura base del proyecto de telerehabilitación.

Se parte de una instancia en la nube de *Amazon,* donde se encuentra instalado un sistema operativo *Ubuntu 18.04* sobre el que se ejecuta un servidor web *Apache* con *Django*. Este servidor será el encargado de mostrar la página web correspondiente a la clínica desde donde cada terapeuta podrá gestionar los pacientes y asignarles los ejercicios que tendrán que realizar en su domicilio.

En el domicilio del paciente se instalará una *RaspberryPi* con un sistema *Raspbian* basado en *Linux*. En dicho dispositivo estará ejecutándose un *script* en *Python* que recogerá los ejercicios asignados por el terapeuta al paciente de la nube y se encargará de mostrárselos al paciente a través de *Telegram* en su teléfono. Esto se consigue mediante la integración de un bot de *Telegram* en el propio *script* que le permite comunicarse con el dispositivo móvil del paciente.

Existe la posibilidad de conectar la *RaspberryPi* al televisor del paciente a través de un puerto *HDMI* para mostrar algunos videos guía mientras el usuario realiza los ejercicios. Pero en el proyecto base todavía no hay desarrollada una interfaz

gráfica como tal y será uno de los desarrollos realizados en este trabajo y que se detallará más adelante en el *apartado 4.6*.

Una vez realizados todos los ejercicios, la *RaspberryPi* enviará al servidor de la nube los resultados de la sesión incluyendo tiempos, ejercicios realizados y *feedback* de la experiencia del paciente.

Finalmente, el terapeuta podrá acceder al servidor web y comprobar que los ejercicios de la sesión han sido realizados por el paciente, además de tener un informe sobre los resultados de la sesión para poder ver su evolución a lo largo del tiempo.

Capítulo 4 Proyectos desarrollados

4.1 Introducción

Durante la realización del presente trabajo, se ha ido incrementando progresivamente el alcance con el desarrollo de pequeños subproyectos. Aunque siempre con el mismo propósito general, cada uno de estos proyectos ha estado motivado por diferentes necesidades que han surgido a lo largo del tiempo.

Desde la integración de pequeños sensores inerciales *Bluetooth* en el proyecto de telerehabilitación, pasando por el desarrollo de diferentes herramientas para el análisis, visualización y tratamiento de los datos recogidos, hasta la creación de una interfaz gráfica para ayudar al paciente al uso y colocación de los sensores y mejorar su experiencia de usuario.

A lo largo del presente capítulo se irán detallando cada uno de estos desarrollos, así como las motivaciones y tecnologías involucradas en ellos. Es importante entender que, aunque con cierto solapamiento cada uno de estos proyectos han sido realizados en el mismo orden cronológico en el que serán presentados en el capítulo, utilizando en cada proyecto los conceptos aprendidos durante los desarrollos anteriores.

4.2 Desarrollo de librería en Python para Xsens Dot

El objetivo inicial de partida del proyecto es la integración de los sensores *Xsens Dot* al trabajo de fin de grado de *Javier Delgado Rodríguez* de *2021*, titulado "*Sistema de tele-rehabilitación a domicilio con chatbot y microordenador de bajo coste*" [21].

El propósito final de esta integración, sería capturar los movimientos del paciente en su domicilio a través de los sensores inerciales *Xsens Dot*, durante la realización de los diferentes ejercicios de rehabilitación para su posterior análisis en el servidor.

Como se expuso en el *apartado 2.2*, la tecnología *BLE (Bluetooth Low Energy)* al contrario que en el *Bluetooth* clásico, está diseñada para ofrecer un consumo de energía significativamente menor para aplicaciones como sensores de proximidad o monitores de ritmo cardíaco. Estos perfiles de bajo consumo siguen comúnmente la especificación *GATT (Generic ATTribute Profile),* que permite enviar pequeños fragmentos de datos llamados atributos.

Cada atributo es reconocido mediante un identificador universal o (*UUID*) de 128 bits y cada uno de estos atributos tienen un formato en el que se especifican las

características y servicios. Un servicio es una colección de características que representan diferentes acciones disponibles para realizar sobre el dispositivo. Por ejemplo, un servicio denominado "*servicio de batería*" puede tener una característica de "*ver el nivel de batería*".

En cada característica se pueden hacer tres diferentes tipos de acciones:

- ***Read*** (leer): para recibir información sobre el estado del sensor o realizar confirmaciones de operaciones.
- ***Write*** (escribir): para enviar información u órdenes al sensor.
- ***Notify*** (notificar): donde se realiza una suscripción a un canal de notificaciones, de modo que cuando el servidor disponga de información nueva envíe esta automáticamente.

Todos los atributos de *Xsens Dot* tienen un identificador *UUID* con el siguiente formato en hexadecimal ***1517xxxx-4947-11E9-8646-D663BD873D93***, donde los caracteres ***xxxx*** son los valores que cambian en función de las diferentes características y servicios disponibles. Todas estas características con su código vienen especificadas en la *Tabla 1* con las posibles acciones documentadas que se pueden realizar sobre los sensores [23].

Service	Characteristic	UUID	Property
Configuration	***Device information***	*0x1001*	*Read*
	Device control	*0x1002*	*Read +Write*
	Device report	*0x1004*	*Notify*
Measurement	***Control***	*0x2001*	*Read+Write*
	Long payload length	*0x2002*	*Notify*
	Medium payload length	*0x2003*	*Notify*
	Short payload length	*0x2004*	*Notify*
	Magnetic field mapper	*0x2005*	-
	Orientation reset control	*0x2006*	*Read+Write*
	Orientation reset status	*0x2007*	*Read*
	Orientation reset data	*0x2008*	-
Battery	***Battery status***	*0x3001*	*Read+Notify*
Message	***Manage control messages***	*0x7001*	*Write*
	Manage acknowledge messages	*0x7002*	*Read*
	Manage notification message	*0x7003*	*Notify*

Tabla 1 : Servicios y atributos en Xsens Dot.

La propia plataforma *Xsens* proporciona desde su página web varias alternativas que facilitan la tarea de los desarrolladores, pudiendo conectar los sensores y acceder a todas sus funcionalidades de forma sencilla.

Una de las opciones disponibles en la web sería el *SDK* o *kit de desarrollo de software* para la creación de aplicaciones *Android* o *iOS* con todas las funcionalidades necesarias ya implementadas para la interacción con los sensores. Sin embargo, por el momento esta *API* no se va a utilizar ya que el objetivo final es incorporarlo a un bot de telerehabilitación, que se ejecuta sobre una *RaspberryPi* con un sistema operativo *Linux*.

Otra de las opciones disponibles sería utilizar el proyecto en *GitHub* publicado por la compañía *Xsens* llamado "*Xsens_dot_server*". Este proyecto se trata de un servidor simple en el que se pueden conectar varios sensores y realizar capturas de manera sencilla. El lenguaje en que está desarrollado es *JavaScript* ejecutado sobre un servidor *Node.js*, sin embargo, el proyecto de telerehabilitación está escrito originalmente en *Python* de modo que no sería fácil ni eficiente hacer compatibles los dos *scripts*.

La solución que se ha escogido finalmente, es realizar nuestro propio *software* utilizando directamente la documentación proporcionada por el fabricante, donde se encuentran las especificaciones de los diferentes servicios *BLE (Bluetooth Low Energy)* disponibles para los sensores *Xsens Dot*.

Siguiendo esta documentación se puede hacer un *software* propio a bajo nivel que conecte los sensores y además se puede escoger el lenguaje en el que va a ser desarrollado que en este caso será *Python.* De tal modo que la integración con el proyecto general será mucho más sencilla y eficiente consiguiendo además mayor flexibilidad y mantenibilidad del código para poder ser utilizado en proyectos futuros.

Como base para este desarrollo se ha partido de la librería *Bleak*, un cliente de *software GATT* capaz de conectarse a dispositivos *BLE* que actúan como servidor. Los principales motivos para la elección de esta librería son:

- Soporte de la versión *Bluetooth* 5.0 que es la que mejor rendimiento va a tener para los sensores *Xsens Dot* según las propias especificaciones del fabricante.
- Funciona de manera asíncrona de forma nativa mediante el uso de la librería *asyncio. Esta librería* está pensada para tener un alto rendimiento en redes y servicios de conexión concurrentes, lo que resulta perfecto para poder incluir las funcionalidades de forma paralela sin interrumpir el flujo normal del programa.
- Soporte multiplataforma, característica importante ya que se pretende que el desarrollo sea flexible para incluir en diferentes proyectos en múltiples sistemas operativos como *Windows*, *Linux* e *iOS*.

Bleak proporciona la función para escanear los dispositivos *Bluetooth* cercanos disponibles y por otro lado las tres funcionalidades básicas que se va a utilizar para interactuar con los sensores, que como ya se ha visto serán lectura y escritura para dar instrucciones a los sensores y la función de recepción de notificaciones para

recoger las medidas de los sensores cada vez que estas estén disponibles en el servidor [24].

Siguiendo la documentación se ha implementado una librería que incluye todas las funciones disponibles para desarrolladores, teniendo en cuenta los formatos de trama enviados y recibidos y los protocolos de interacción establecidos. Esta librería se ha acabado implementado en las siguientes cuatro clases:

- ***BlueDotScan***: Clase que se encargará de las funciones de escaneo de sensores *Xsens Dot* cercanos disponibles, para esto primero se realiza el escaneo utilizando el propio gestor de dispositivos *Bluetooth* del sistema operativo y luego se filtran los resultados por el nombre de la etiqueta "*Xsens DOT*".
- ***BlueDot***: Clase que representa a un solo sensor y que implementa todas las funcionalidades que aparecen en la documentación, donde aparecen especificados cada uno de los formatos de trama y los protocolos que han de seguirse.
  Esta clase funciona con un bucle principal ejecutándose en un hilo independiente, al cual se van haciendo llamadas asíncronas a sus diferentes métodos que contienen las funcionalidades. Finalmente, se ha diseñado con la opción de poder pasar la referencia de una función externa de *callback* para decidir qué hacer con cada una de las medidas recibidas y que de esta manera la clase pueda ser reutilizada en más proyectos.
- **MeasureParser**: Clase auxiliar utilizada en *BlueDot* y que se encarga de interpretar los diferentes formatos de salida del sensor en función de cómo haya sido configurado, por ejemplo para devolver solo un cuaternión, un cuaternión y la aceleración angular, etc.
- ***BufferReader***: Clase auxiliar utilizada en *BlueDot* y *MeasureParser* encargada de decodificar los diferentes formatos de tramas. Para esto cuenta con un *buffer* donde se almacena la trama, un puntero interno que hace referencia al lugar dónde se está leyendo del *buffer* en ese instante y diferentes métodos que leerán los siguientes bits al puntero y se convertirán al tipo especificado, por ejemplo: a un entero de 8 bits, a un entero de 16 bits, a un *float*, etc.

Es importante notar que la librería *Bleak* en sistemas *Linux*, requiere los permisos necesarios para acceder a los servicios *Bluetooth*. De tal manera que ha de ser ejecutado como *root* o bien añadiendo al usuario al grupo *bluetooth,* siendo esta última opción la más segura de las dos ya que es la más restrictiva en permisos.

4.2.1 Librería *dotManager*

En este punto, ya tenemos una librería que nos permite crear un objeto que representa un sensor y sobre el que podríamos llamar a todas sus funcionalidades. Sin embargo, aunque esto proporciona una gran flexibilidad, también hace que sea más compleja su integración en el proyecto final del bot.

Por lo tanto, se va a necesitar crear otra librería sobre esta clase capaz de gestionar un grupo de al menos cinco sensores a la vez y poder dar instrucciones simultáneas a todos ellos.

A esta clase se la llamará *dotManager* y se encarga de gestionar simultáneamente una lista de objetos tipo *BlueDot* de forma simple y sin bloquear la ejecución del hilo principal dónde estará ejecutándose el bot. Además de lo expuesto hasta ahora, añade las siguientes características importantes:

- Desconexión de los sensores desde el sistema operativo utilizando la librería *pydbus* para *Linux*. Esta es interesante porque en ocasiones al cerrarse el programa de forma inestable algunos de los sensores se pueden quedar conectados por un tiempo afectando al funcionamiento.
- Reintentos de conexión y comprobación de operaciones para mejorar la resiliencia del sistema. Esta característica es importante ya que en ocasiones se producen errores al realizar alguna operación y se puede subsanar de forma sencilla simplemente con un reintento.
- Proceso de identificación de los sensores y establecimiento de la relación entre la dirección *MAC* y la parte del cuerpo en la que está colocado cada sensor.
  Esta característica se utiliza a la hora de realizar las capturas y proporciona flexibilidad para el caso de cambio de posición de los sensores o incluso cuando se deba reemplazar alguno de ellos.
- Procesamiento de los datos recibidos por los sensores y generación de ficheros de captura formateados para su posterior análisis.

Se han utilizado varias configuraciones por defecto en el desarrollo de esta librería con el objetivo de simplificar su uso, aunque estas podrían ser modificadas en cualquier momento dependiendo de los requisitos:

- Se ha establecido la frecuencia de muestreo de los sensores a 30 muestras por segundo, con esto se intentará mejorar el rendimiento cuando hay varios sensores a la vez, dando una resolución suficiente para recoger movimientos propios de una sesión de rehabilitación.
  En el caso de querer recoger movimientos más bruscos como los efectuados en un entorno deportivo, sería recomendable aumentar el número de muestras.
- Las mediciones recibidas serán únicamente cuaterniones ya que esta información será la que por el momento se va a necesitar para realizar los análisis de movimiento, además se reduce la cantidad de información transmitida al mínimo mejorando el rendimiento.
- El número de sensores utilizados en la monitorización será de cinco. Esto se debe al formato en el que son distribuidos los sensores, de tal manera que las capturas se podrán hacer bien de la parte superior o bien de la parte inferior del cuerpo para conseguir la resolución necesaria como se detallará más adelante.

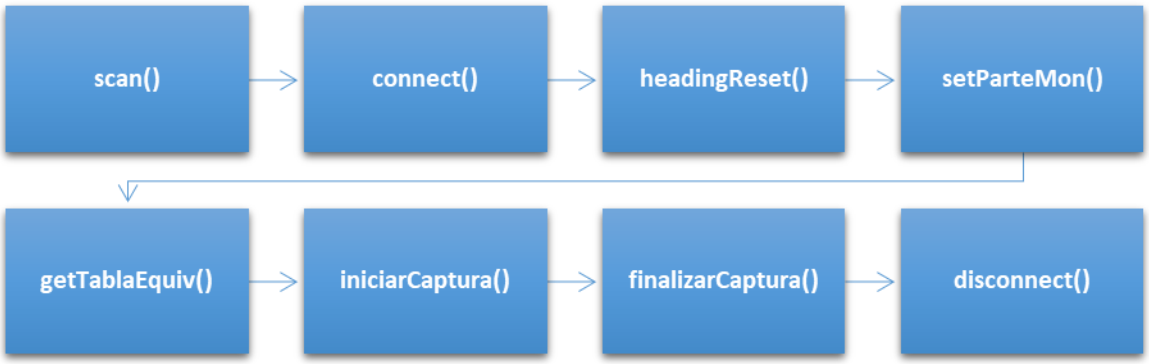

Figura 11 : Utilización estándar de librería dotManager.

En la *Figura 11* se puede ver el diagrama de flujo del uso más común de esta librería. A continuación, se detallará que hace la llamada de cada método del diagrama:

- **scan**: Realiza una búsqueda de cinco sensores *Xsens Dot* cercanos disponibles.
- **connect:** Intenta realizar la conexión a esos cinco sensores y en caso de que alguno de ellos falle reintenta el proceso hasta un máximo de cinco veces cada uno.
- **headingReset**: Al realizar el proceso denominado como "*heading reset*", el sensor toma la posición en la que se encuentra en ese momento como la orientación de referencia cero y todas las demás lecturas de ahí en adelante serán rotaciones relativas a esa posición.
  Este es un proceso obligatorio cada vez que se enciendan los sensores y han de ser colocados todos de la misma forma para que todos partan de un mismo punto de referencia común y las lecturas de unos sensores respecto a otros sean coherentes.
  En este caso se va a aprovechar a realizar este proceso en la misma base de los sensores que los mantiene a todos alineados en la misma posición.
- **setParteMon**: Establece según el argumento si se va a monitorizar la parte de arriba o la parte de debajo del cuerpo, este proceso será necesario porque un sensor podría estar colocado en dos partes del cuerpo diferentes, en función de si es la parte de arriba o la de abajo la que va a ser monitorizada.
- **getTablaEquiv**: Recoge de un fichero la tabla de equivalencias entre la *MAC* del sensor y la parte del cuerpo correspondiente al sensor con esa *MAC*. Luego esta equivalencia se utiliza para identificar en el fichero de captura a que parte del cuerpo corresponde cada medida.
- **iniciarCaptura**: Da la orden de comenzar a enviar las mediciones a todos los sensores y va volcando estas en un fichero de captura en el directorio "*/cap*".

- **finalizarCaptura**: Da la orden de detener las mediciones a los sensores y finaliza el proceso de captura.
- **disconnect**: Desconecta todos los sensores.

Para la utilización de la clase solo haría falta importar la librería e instanciar el objeto *dotManager* al inicio del bot de telerehabilitación, para posteriormente realizar las llamadas a los métodos expuestos anteriormente cuando sea necesario.

Una vez realzadas las capturas, solo habría que proceder a sincronizar los ficheros generados en el directorio de capturas con el servidor donde serán procesadas y analizadas.

4.3 Desarrollo de visualizador de cuaterniones

Los cuaterniones presentados con más detalle en el *punto 2.7,* son una herramienta muy útil para la representación de rotaciones en el espacio, aunque debido a su complejidad resultan muy poco intuitivos a la hora de ser interpretados a simple vista. Para tener una mejor comprensión del funcionamiento de los cuaterniones y poder comprobar que los datos recibidos desde los sensores son correctos se ha desarrollado una pequeña herramienta para la representación de las rotaciones en un entorno *3D* con *Pygame* (*punto 2.3)* y Py*OpenGL* (*punto 2.4)*.

El objetivo inicial de este proyecto consiste en combinar la librería *Bluetooth* para *Xsens Dot* presentada en el *punto 4.2* con *Pygame* y de esta manera poder representar en tres dimensiones las rotaciones de uno o varios sensores en tiempo real.

OpenGL proporciona un gran número de formas primitivas básicas que pueden ser usadas para construir una escena en tres dimensiones, tales como puntos, líneas, etc. Se comenzará dibujando un eje de coordenadas en el centro que represente la referencia global sobre la que se aplicará la rotación del cuaternión al objeto, para realizar esto serán simplemente tres formas primitivas del tipo *GL_LINE* con colores diferentes.

Por convención *x, y, z* se representarán respectivamente como los colores *RGB* (Rojo, Verde y Azul) y para mejorar la comprensión del sistema representado se añaden etiquetas flotantes en los extremos de cada eje con su nombre.

A continuación, se aplica la función de rotación llamada *glRotatef* con el valor del cuaternión a representar. Es importante entender que esta operación hará que los ejes de referencia cambien una vez aplicada la rotación, de tal manera que si a continuación se comienza a dibujar el objeto, este se dibujará ya con el giro aplicado.

De esta forma solamente habría que proceder a dibujar las seis caras del cubo para formar la representación del sensor y sus dimensiones, para esto se utilizará la primitiva *GL_QUADS* indicándole los vértices de cada cara y sus colores siguiendo la convención *RGB,* y nuevamente para facilitar la comprensión de la representación, se dibuja otro eje de coordenadas internamente al cubo anterior con sus etiquetas correspondientes.

Para finalizar, se coloca la cámara mediante las funciones de rotación y translación, de tal manera que se pueda apreciar bien el giro del sensor con respecto al eje de referencia general. En la *Figura 12* se muestra la salida obtenida en el visualizador.

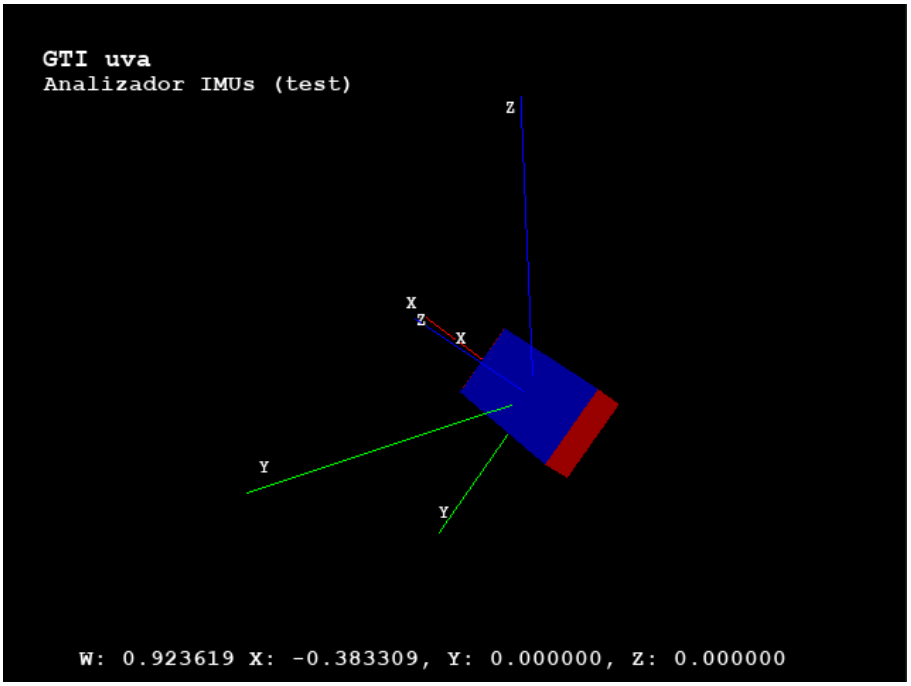

Figura 12 : Visualizador de cuaterniones.

Esta función de representación de cuaterniones ha sido implementada íntegramente dentro de una misma clase llamada *imuViewer,* con la intención de crear un módulo que pueda ser incluido fácilmente en cualquier otra herramienta.

Combinando las clases *imuViewer* y la librería de *Bluetooth* de *Xsens Dot* se ha desarrollado un programa capaz de representar en tiempo real a 30 cuadros por segundo los giros de cada sensor conectado.

Además, se han incluido algunas funcionalidades como la capacidad de realizar un *heading reset* a todos los sensores simultáneamente, cambiar la visualización entre varios sensores, realizar una captura de los datos recibidos de los sensores o ser capaz de mover la cámara para tener una mejor visión de lo que está sucediendo.

Aunque parece un programa sencillo finalmente ha resultado ser una herramienta especialmente útil, no solo nos ha ayudado a entender mejor el funcionamiento de las operaciones matemáticas con cuaterniones, sino que ha servido de apoyo a otros proyectos dentro del laboratorio.

Uno de estos ejemplos ha sido el desarrollo de un programa para los twynsens de *Javier González Alonso* donde se sigue utilizando la clase *imuViewer* para representar los cuaterniones de sus sensores, pero al contrario que en el caso anterior estos datos se envían a través del puerto serie y con un formato ligeramente diferente a los de *Xsens*.

4.4 Desarrollo de avatar simple con PyOpenGL

Este proyecto nació con el propósito inicial de formar parte del bot de telerehabilitación, mostrando al paciente por pantalla una representación de las lecturas que recogían los sensores sobre un avatar sencillo teniendo en cuenta las limitaciones de las *RaspberryPi*. La idea era que el usuario tuviera un *feedback* en tiempo real sobre las lecturas de los sensores, además de ser un potencial atractivo para niños en proceso de rehabilitación.

Más adelante sin embargo, demostraría ser una herramienta muy útil para el análisis de datos permitiéndonos analizar capturas representando los datos recogidos de los sensores sobre un avatar, de tal forma que se podría llegar a ver el movimiento capturado sin ningún tipo de procesado adicional que pudiera alterar los datos. Además, como se verá más adelante permite analizar y comprender de forma visual los efectos de la corrección de errores durante el proceso de calibración.

4.4.1 Colocación de los sensores

En este proyecto se dispone de cinco sensores *Xsens Dot,* ya que como se comentó en el *punto 2.1* el formato en el que estos sensores son vendidos son de cinco sensores junto a su base de carga.

La cantidad de sensores que se utilizan para recoger los movimientos del paciente son importantes para mejorar la precisión en el proceso de reconstrucción del movimiento, aunque sin embargo la conexión de demasiados sensores podría conllevar pérdidas de información debido a las limitaciones del propio protocolo de comunicación *Bluetooth* y derivar en un funcionamiento inestable.

Con la limitación de usar solamente cinco sensores, no se podrá tener una buena resolución de los movimientos si se intentara realizar la captura a todo el cuerpo, de tal manera que a la hora de realizar las capturas se distinguirá entre dos modos de uso:

- ***Monitorización del tronco superior***: Con un sensor situado en la espalda y dos sensores colocados en los laterales de cada brazo, uno para monitorizar la parte superior y otro para la parte inferior.
- ***Monitorización del tronco inferior***: Con un sensor situado en la pelvis y como en el caso anterior, en cada pierna irán dos sensores colocados en los laterales, uno para monitorizar la parte superior y otro para la inferior.

Como indica la *Figura 13,* la nomenclatura que se utilizará para referirnos a cada parte del cuerpo que se va a monitorizar será la siguiente:

- ***Back***: Sensor situado en la espalda.
- ***Pelvis***: Sensor situado en la parte de atrás en la pelvis.
- ***LUA/LLA***: *Left Upper Arm/ Left Lower Arm*
- ***RUA/RLA***: *Right Upper Arm / Right Lower Arm*
- ***LUL/LLL***: *Left Upper Leg / Left Lower Leg*
- ***RUL/RLL***: *Right Upper Leg / Right Lower Leg*

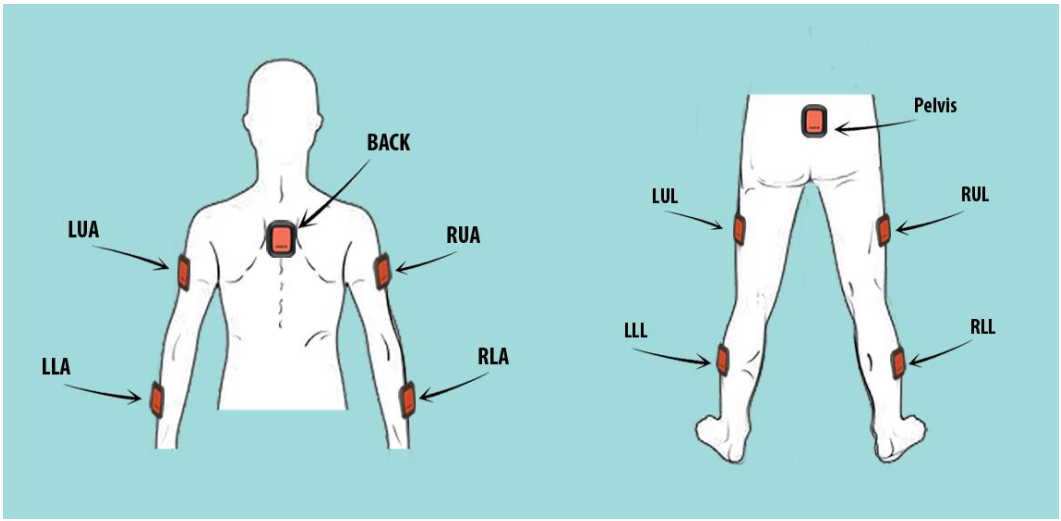

Figura 13 : Colocación de los sensores parte superior e inferior.

Es importante notar que los sensores siempre estarán colocados de la misma forma, con el logo *Xsens* hacia abajo de tal manera que la referencia *x* local del sensor esté apuntando siempre hacia arriba en todos ellos en la posición de calibración inicial.

4.4.2 Algoritmo de dibujado de avatar con PyOpenGL

Se pretende implementar un modelo esquemático simple con el mínimo número de articulaciones que se van a tener en cuenta durante el proceso de monitorización.

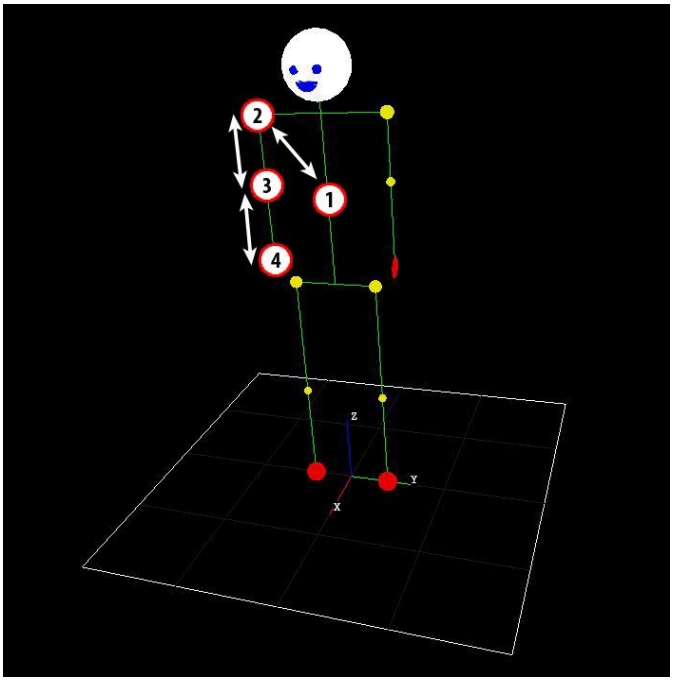

Figura 14 : Algoritmo para dibujado del brazo derecho.

Como se explicó en los *puntos 2.3* y *2.4* anteriores en la introducción a *Pygame* y *OpenGL*, este proceso se realiza a través del uso de formas primitivas simples con la utilización de segmentos que representarán cada parte de la extremidad y unas pequeñas esferas de diferente color que delimitarán cada articulación.

Finalmente, los pies y las manos serán dibujadas utilizando círculos para representar las palmas y poder ver la rotación sobre sí mismas con facilidad.

Para mejorar la comprensión del espacio representado, al igual que en el proyecto anterior se incluirán unos ejes que marcan la referencia de coordenadas general, y se añadirá además un pequeño trozo de suelo a los pies del avatar a modo de referencia espacial al rotar la cámara como se aprecia en la *Figura 14*.

Para explicar el algoritmo de dibujado se tomará de ejemplo el brazo derecho, pero los pasos son similares para el resto de extremidades cambiando proporciones y utilizando los cuaterniones que le correspondan a cada parte del cuerpo.

Se partirá inicialmente de los siguientes datos que se han recogido de los sensores:

- ${}^{\text{hab}}Q_{\text{back}}$: Cuaternión que representa la rotación de la espalda respecto al sistema de referencias de la habitación.
- ${}^{\text{hab}}Q_{\text{RUA}}$: Cuaternión que representa la rotación del brazo superior derecho respecto al sistema de referencias de la habitación.
- ${}^{\text{hab}}Q_{\text{RLA}}$: Cuaternión que representa la rotación del brazo inferior derecho respecto al sistema de referencias de la habitación.

Utilizando como guía las posiciones indicadas en la *Figura 14,* los pasos a seguir para dibujar estas rotaciones en el avatar serían los siguientes:

1. Se realiza una translación a la *posición 1* haciendo uso de la función *glTranslatef*. Es importante resaltar que esto va a hacer que ese punto sea ahora el origen de coordenadas donde en el paso siguiente comenzar a dibujar.
2. Se realiza la rotación del cuaternión de la espalda ${}^{\text{hab}}Q_{\text{back}}$ con la función *glRotatef*. Al igual que en el paso anterior, al aplicar esta rotación el sistema de coordenadas rota, de tal manera que después al dibujar los segmentos pertenecientes al tronco se tendrán que indicar las coordenadas sobre el plano inicial como si no se hubiera hecho dicha rotación. Esta característica es muy interesante ya que al no tener en cuenta la rotación aplicada a la hora de dibujar las referencias siempre serán las mismas simplificando notablemente el proceso.
3. A continuación, se realiza otra translación a la *posición 2* haciendo uso nuevamente de la función *glTranslatef*. Hay que tener en cuenta que las operaciones anteriores han cambiado el sistema de coordenadas, de modo que siempre se encontrará la *posición 2* con las mismas coordenadas.
4. Una vez situados en la *posición 2*, se realiza la rotación que nos indica el cuaternión del brazo superior derecho.
   Es muy importante entender que después de los procesos anteriores ha cambiado nuestro sistema de coordenadas, y esto implica que la rotación de la espalda que se llamará segmento "*padre*" ya estará aplicada, de modo que

la rotación que se necesitará realizar al segmento "*hijo*" que en este caso es el brazo superior derecho ha de que ser una rotación relativa a la espalda y no a la habitación que sería el dato del que se dispone.
El procedimiento para hallar la rotación relativa del brazo superior derecho respecto de la espalda es la que aparece en la siguiente ecuación:

$$({}^{hab}Q_{back})^{-1} * {}^{hab}Q_{RUA} = {}^{back}Q_{RUA} \qquad (13)$$

Una vez se ha aplicado la rotación ${}^{back}Q_{RUA}$ con la función *glTranslatef*, se procede al dibujado del brazo superior y del codo utilizando las formas primitivas con un segmento y una esfera respectivamente.

5. A continuación, se realiza una translación a la *posición 3* con la función *glTranslatef* con las consideraciones de los casos anteriores.
6. Una vez situados en la *posición 3* se procede a realizar el giro teniendo en cuenta todas las rotaciones realizadas hasta este punto. El procedimiento para hallar el cuaternión que se necesita será realizando la siguiente operación:

$$({}^{hab}Q_{back} * {}^{back}Q_{RUA})^{-1} * {}^{hab}Q_{RLA} = ({}^{hab}Q_{RUA})^{-1} * {}^{hab}Q_{RLA} = {}^{RUA}Q_{RLA} \qquad (14)$$

De modo que lo que se necesitará es simplemente multiplicar la inversa en referencia general del padre por el hijo en referencia general. Esta operación sería homóloga si quisiéramos añadir más segmentos siguiendo la jerarquía "padre" e "hijo" con la operación que se muestra en la *ecuación 15*.

$$({}^{hab}Q_{PADRE})^{-1} * {}^{hab}Q_{HIJO} = {}^{PADRE}Q_{HIJO} \qquad (15)$$

7. Una vez realizada la rotación se dibuja el antebrazo y la mano con una esfera que representará la palma y nos permitirá ver el giro del antebrazo sobre sí mismo.
8. Para finalizar, se deberá regresar a la *posición 1* deshaciendo previamente todas las rotaciones y translaciones que se han ido realizando, pero esta vez en orden inverso.
   Se hará inicialmente el proceso de rotación de $({}^{RUA}Q_{RLA})^{-1}$ y la translación a la *posición 2*, a continuación, se realizará otra rotación de $({}^{back}Q_{RUA})^{-1}$ y finalmente otra translación a la *posición 1* donde se había partido. A partir de este punto se seguirán dibujando con la misma metodología expuesta el resto de extremidades.

Todo este proceso de dibujado se realizará dentro de un bucle a 30 cuadros por segundo con los nuevos datos que se estén recibiendo de los sensores.

4.4.3 Procedimiento de calibración

Para la correcta interpretación de los datos recogidos de los sensores es necesario realizar antes un proceso de calibración con el propósito de corregir los errores de cada segmento del cuerpo. Estos errores se derivan de factores como una mala colocación de los sensores por parte del usuario, algo esperable en un entorno doméstico dónde el paciente se los ha de colocar a sí mismo o incluso los ángulos introducidos por la propia morfología del cuerpo de la persona, lo cual es simplemente inevitable y siempre va a estar presente en mayor o menor medida.

Existen múltiples publicaciones que intentan abordar esta problemática desde diferentes puntos de vista. Por ejemplo, en una prueba de concepto presentada en *2007* se utiliza una serie de bobinas situadas en la parte central del cuerpo que emiten una serie de pulsos magnéticos, y con las lecturas de los magnetómetros de los sensores situados en las extremidades tratan de estimar la posición relativa de los sensores en el espacio utilizando las bobinas como puntos de referencia [25].

Otra investigación incluso ha planteado la posibilidad de utilizar redes neuronales entrenadas para estimar la posición del cuerpo solamente con la lectura de orientaciones de los sensores, consiguiendo márgenes de error en torno a los siete grados de precisión y reduciendo el número de sensores necesarios a cinco para la reconstrucción de los movimientos de todo el cuerpo [26].

Sin embargo, la opción más utilizada actualmente en el mercado para realizar la calibración de los sensores debido a su simplicidad es pedir al usuario que al inicio de la sesión se coloque en una posición predefinida, de esta manera se puede estimar la diferencia entre las rotaciones medidas en ese instante y las esperadas si los sensores estuvieran perfectamente colocados en esa posición sobre un "*modelo ideal*" (con extremidades perfectamente rectas).

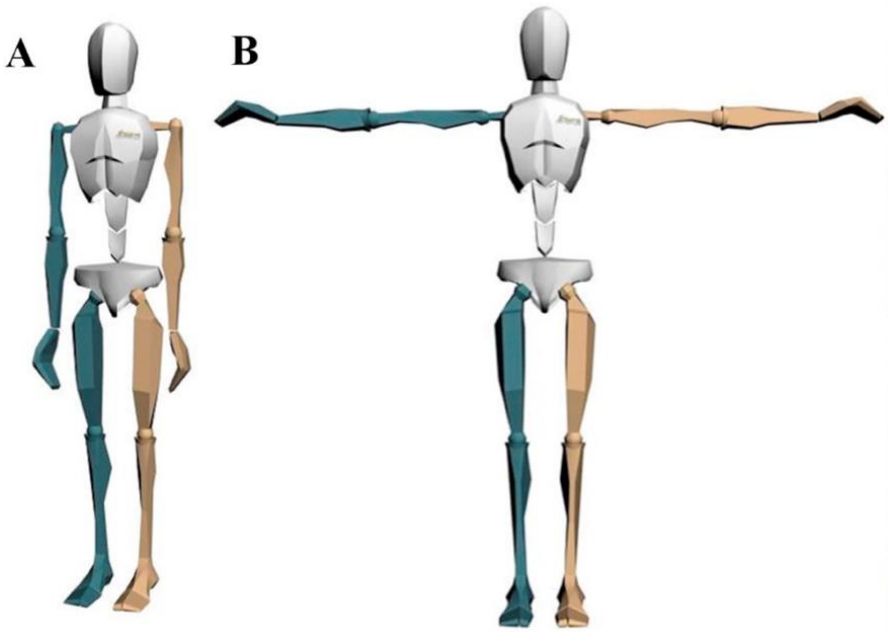

Figura 15 : N-Pose y T-Pose para la calibración.

Un estudio realizado por el instituto de investigación *Robert-Sauvé* de Canadá ha realizado una comparativa de precisión entre las diferentes poses más utilizadas para la calibración de sensores inerciales, comparando también entre los casos en los que una tercera persona coloca los sensores sobre el cuerpo del paciente y la colocación de los sensores a uno mismo. Las dos posiciones iniciales más utilizadas y que fueron estudiadas fueron la denominada "*N-pose*" y la "*T-pose*", que se pueden ver en la *Figura 15* como *A* y *B* respectivamente [27].

Aunque en el estudio se menciona que la postura "*T-Pose"* en el escenario de colocación pasiva es la que ha dado mejores resultados en términos de precisión, esta podría llegar a ser un problema para determinados pacientes en proceso de rehabilitación con problemas de movilidad reducida que les impida adquirir esa pose.

De tal manera que se ha tomado la decisión de finalmente utilizar entonces la "*N-Pose"* como estándar para la calibración de los sensores en el momento inicial, en combinación con un procesado sobre un modelo musculoesquelético del que se hablará en el siguiente *apartado 4.5* con más detalle.

4.4.4 Proceso de corrección de errores

Para poder explicar de una forma más clara el proceso de calibración se explicará siguiendo un ejemplo muy sencillo con ángulos exactos, aunque el procedimiento de corrección de errores será el mismo para todos los casos.

Se partirá realizando al inicio de la sesión la medición del sensor de la espalda en la posición inicial de calibración, a este cuaternión se denominará ${}^{hab}Q_{ini}$ al que se le ha introducido un error de 45 grados debido a una mala colocación del sensor sobre la espalda por parte del paciente. La medición y el error introducido se indican en la imagen de la izquierda de la *Figura 16*.

Asumiendo que el paciente al inicio de la sesión se encuentra situado correctamente en la posición de calibración "*N-Pose*", se procederá a comparar esta lectura con el cuaternión esperado en esa posición sobre un "*modelo ideal*". A este cuaternión esperado se le denominará ${}^{hab}Q_{esp}$ teniendo un aspecto como el que se puede ver a la derecha en la *Figura 16*.

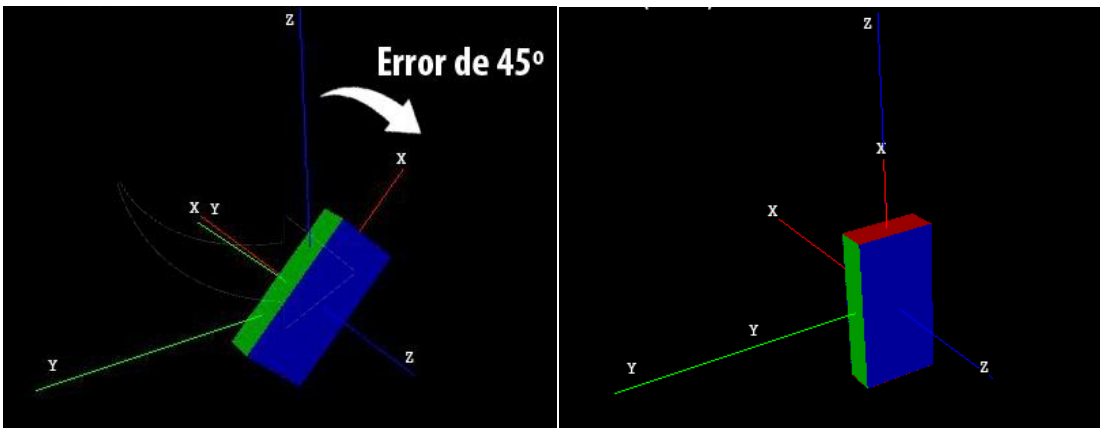

Figura 16 : Cuaternión medido con error y cuaternión esperado sin error.

A continuación, se necesitará extraer este error de posicionamiento de 45 grados denominado ${}^{loc}Q_{err}$ del cuaternión medido, para luego utilizar esta rotación para corregir el resto de medidas, dicha operación se muestra en la siguiente ecuación:

$$({}^{hab}Q_{esp})^{-1} * {}^{hab}Q_{ini} = ({}^{hab}Q_{esp})^{-1} * ({}^{hab}Q_{esp} * {}^{loc}Q_{err}) = {}^{loc}Q_{err} \qquad (16)$$

En la *expresión 16* asumiendo que el único tipo de error existente es de colocación, se utiliza la equivalencia de que el cuaternión inicial ${}^{hab}Q_{ini}$ se puede expresar también como la suma del cuaternión esperado ${}^{hab}Q_{esp}$ más el error en referencia local que es el que se está buscando ${}^{loc}Q_{err}$.

Nótese que el resultado obtenido es el error con respecto al sistema de referencias local, esto ha de ser así para poder corregir el error del sensor en su propio sistema de referencias directamente sin tener en cuenta que rotaciones previas a nivel global se hayan realizado anteriormente.

El resultado de esta operación como se aprecia en la *Figura 17*, es una rotación local sobre el eje *z* de 45 grados tal y como se esperaba.

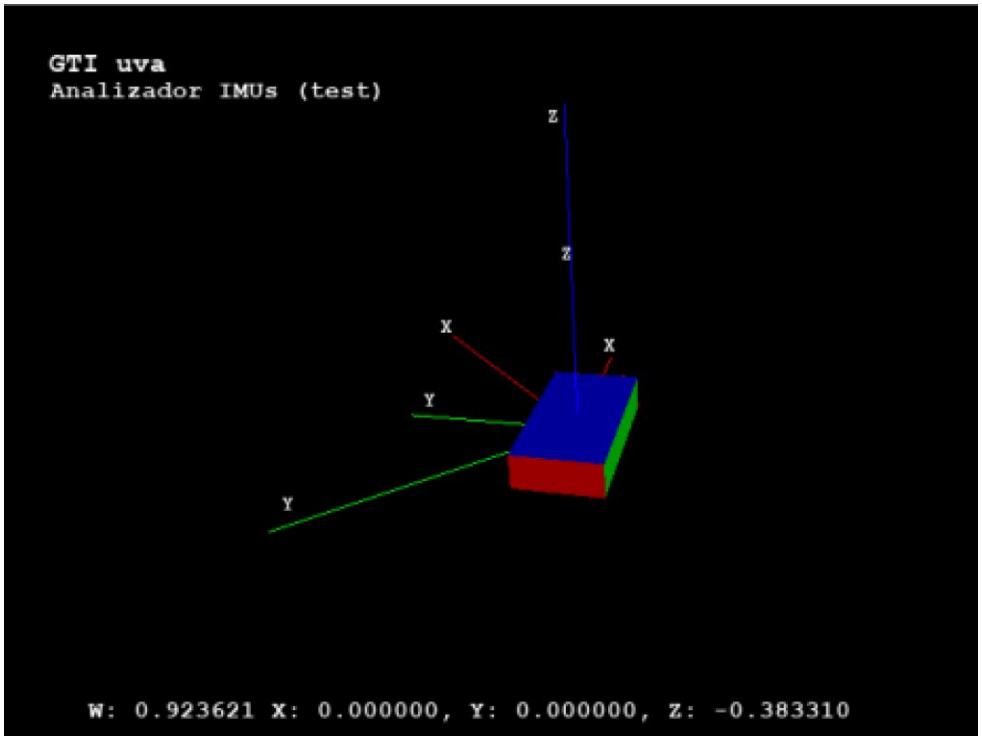

Figura 17 : Cuaternión de error de 45º en z en referencia local.

Este cuaternión de error se guardará en una variable ya que será el que se utilizará para calibrar el resto de las medidas capturadas a partir de entonces.

Siguiendo con el ejemplo anterior, ahora se realizará otra medida del sensor de la espalda donde el sujeto se acaba de tumbar sobre el suelo. Como se puede ver en la

imagen de la izquierda en la *Figura 18*, aunque la medida haya cambiado continúa existiendo un error local de posicionamiento de 45 grados sobre *z*.

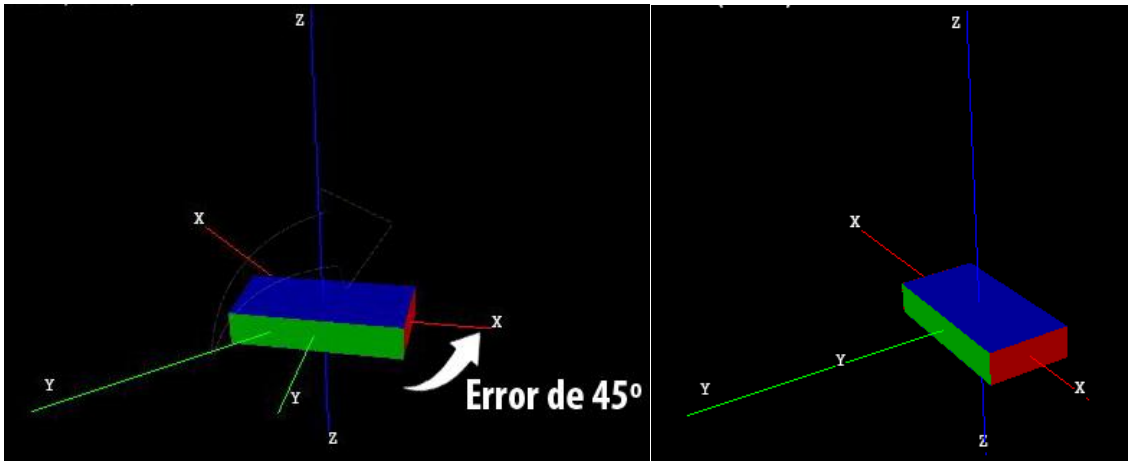

Figura 18 : Medida con error de 45º a corregir y valor esperado sin error.

Se denominará al cuaternión con errores medido en un instante cualquiera como ${}^{hab}Q'_{dat}$ y como ${}^{hab}Q_{dat}$ a ese mismo cuaternión sin errores. El procedimiento para la corrección con el cuaternión de calibración que se había almacenado anteriormente es el que se indica en la siguiente *ecuación 17*.

$$ {}^{hab}Q'_{dat} * ({}^{loc}Q_{err})^{-1} = ({}^{hab}Q_{dat} * {}^{loc}Q_{err}) * ({}^{loc}Q_{err})^{-1} = {}^{hab}Q_{dat} \qquad (17) $$

Como en el caso anterior se utilizará la expresión del cuaternión leído con error ${}^{hab}Q'_{dat}$ como equivalente a la suma del cuaternión sin error ${}^{hab}Q_{dat}$ mas el error local ${}^{loc}Q_{err}$, este error se mantendría constante independientemente de los giros realizados durante el resto de la captura.

Como resultado de esta operación se hallará el giro esperado con el error corregido tal y como se esperaba. La representación de este resultado se puede apreciar en la imagen de la izquierda de la *Figura 19.*

Al inicio del ejercicio, si se han realizado todas las operaciones correctamente se debería ver el modelo en la posición de calibración perfectamente posicionado, representando al modelo ideal.

Por tanto, a partir de ese momento se realizará este mismo proceso de calibración en cada una de las medidas recibidas para subsanar los errores de colocación, pidiéndole al usuario que se sitúe en el instante inicial en la *N-Pose* para poder recoger los errores locales de calibración en variables internas.

En la *Figura 19* se muestra un ejemplo de captura de la marcha, donde la diferencia después y antes de la corrección se vuelve notable. En la imagen de la izquierda estaría corregida y a la derecha sin corregir.

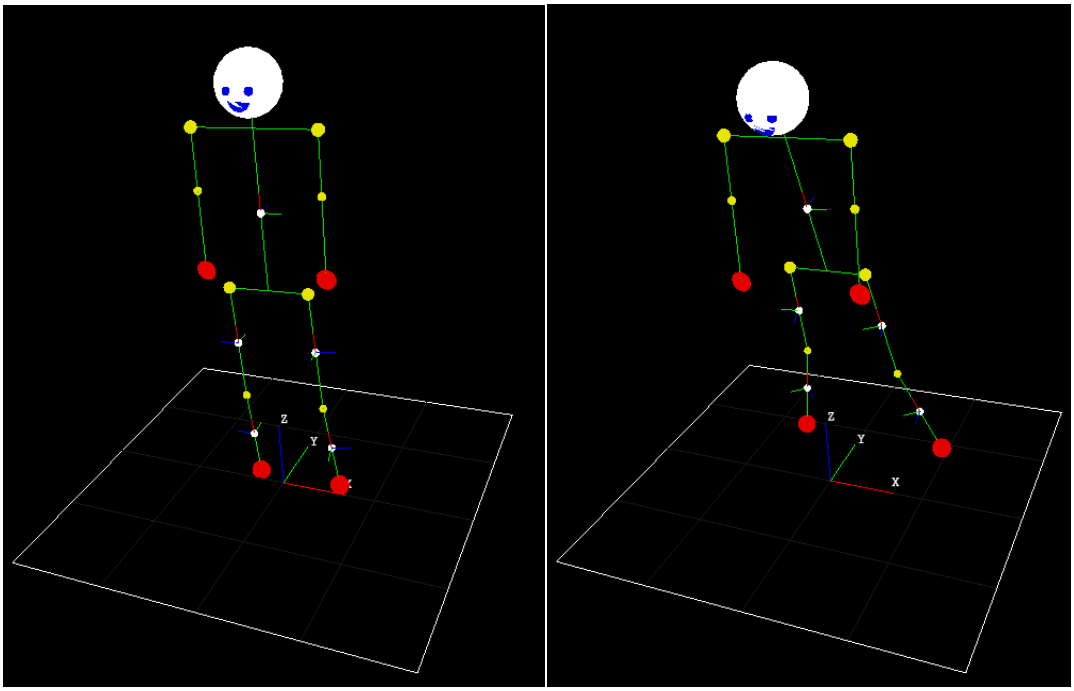

Figura 19 : Errores corregidos izquierda / con errores derecha.

4.4.5 Resultado final

Esta herramienta ha sido uno de los proyectos que más esfuerzo y horas de desarrollo ha llevado del trabajo, pero finalmente ha resultado ser una herramienta fundamental para el análisis de capturas en el laboratorio.

Al igual que los proyectos anteriores, se ha implementado intentando tener la mayor flexibilidad de tal manera que se han incluido todas las funcionalidades dentro de una misma clase y de esta manera se ha podido integrar fácilmente en otros proyectos.

Un ejemplo en el que se ha utilizado esta clase ha sido en el desarrollo para la representación en tiempo real de la lectura de los sensores *Xsens Dot,* añadiendo además la capacidad de poder realizar una captura a un fichero de las lecturas.

Aunque todavía sigue en desarrollo y se seguirán añadiendo más funcionalidades, las características incluidas hasta día de hoy son:

- Movimiento libre de la cámara con el ratón.
- Función de zoom con la rueda del ratón para ver los detalles.
- Parar en cualquier momento la reproducción, aunque se pueda seguir accediendo a todas las funciones de la cámara.

- Activar o desactivar en cualquier momento la corrección de errores pulsando la tecla *e*, esto es especialmente útil para ver los efectos y errores en el proceso de calibración.
- Quitar o poner a través de la tecla *v* unos pequeños ejes de referencia en cada segmento corporal del avatar, para que se pueda apreciar mejor las rotaciones sobre sí mismos.
- Retroceder la captura nuevamente al momento inicial.

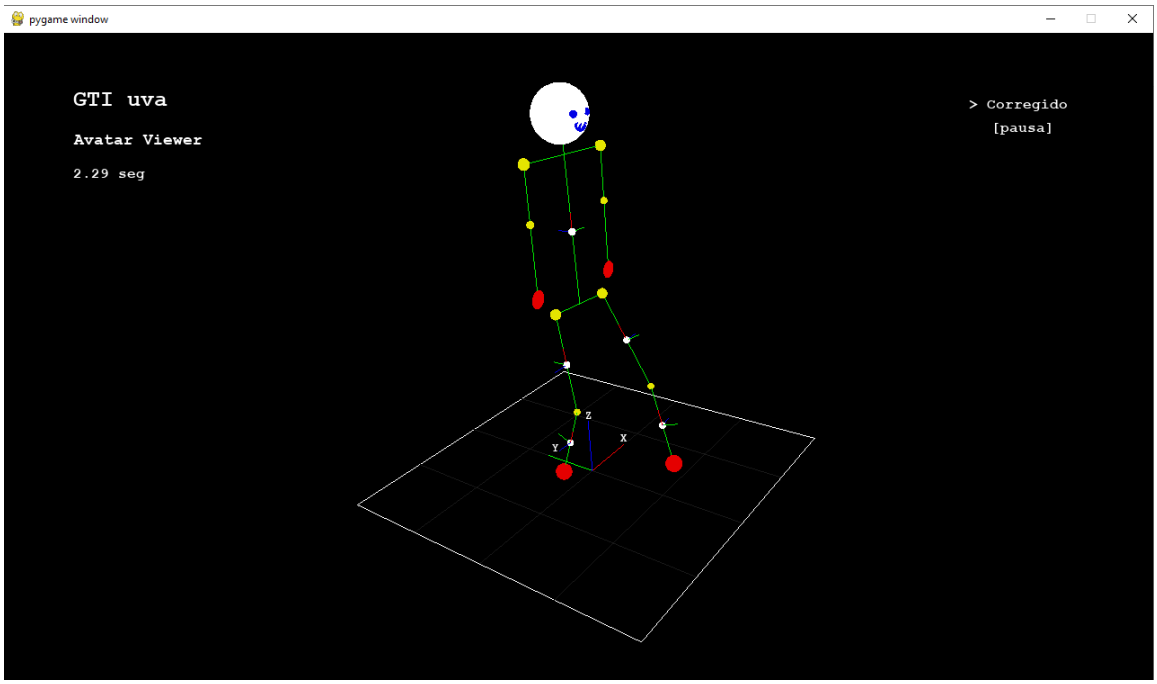

Figura 20 : Captura de marcha representada en el programa AvatarViewer.

Se puede ver el resultado de esta implementación con una captura de análisis de la marcha en la *Figura 20*. En este caso al ser una captura de la parte inferior del cuerpo se mostrarían los pequeños ejes dónde estarían situados los sensores.

4.5 Desarrollo de adaptador para OpenSim

Una vez realizadas las capturas de movimiento durante la sesión de rehabilitación, los datos se enviarán al servidor dónde inicialmente se pasarán por un proceso de corrección de errores a partir de la calibración inicial como se ha expuesto en el *apartado 4.4.4.*

El resultado de este proceso es enviado a un programa en el que se simulan las dinámicas de movimiento sobre un modelo musculoesquelético, que contiene restricciones articulares experimentales del cuerpo humano, y así se logra recomponer el movimiento realizado de la forma más aproximada posible a la realidad.

La herramienta que se utilizará para la realización de este proceso será *OpenSim* presentada en el *punto 2.5*, cuya salida va a proporcionar los datos que posteriormente serán presentados ante los terapeutas con los parámetros biomecánicos relevantes en la terapia generados a partir de las capturas realizadas.

4.5.1 Cálculo de movimientos con OpenSense

Sin lugar a dudas, el módulo más importante para este proyecto que contiene *OpenSim* es un módulo llamado *OpenSense*, una herramienta que permite calcular los movimientos del cuerpo de los pacientes en función de los datos recibidos desde los sensores inerciales (*IMU*).

Para realizar este cálculo se parte de un modelo musculoesquelético simplificado de *OpenSim* que incluye las articulaciones y grados de libertad de movimiento de cada una de ellas, el propio entorno del programa pone a disposición de los desarrolladores algunos modelos en su biblioteca online de forma totalmente libre y gratuita.

A estos modelos se les debe ingresar las lecturas de orientaciones de uno o más sensores inerciales a través de ficheros de captura. Actualmente se asume que en ese fichero el proceso de fusión y sincronización de los datos ya se ha realizado previamente.

Con estos dos ficheros, se asocia cada sensor *IMU* con un segmento del cuerpo del modelo a estudiar y a continuación se proporciona una rutina de calibración básica en la que la primera línea del fichero de captura será tomada como la pose predeterminada para el modelo elegido, aunque nosotros vamos a incluir nuestra propia rutina de calibración con la implementación de nuestros propios *scripts* como se ha detallado en el *apartado 4.4.4*.

Con todos estos datos *OpenSense* realiza un proceso llamado cinemática inversa con el que logra calcular el conjunto de ángulos de cada articulación en cada instante del tiempo, de tal manera que trata de buscar un resultado que minimice los errores entre las lecturas de los sensores experimentales y las restricciones esperadas por el modelo escogido a través de un filtro de mínimos cuadrados.

A la finalización de este proceso, se va a generar un fichero de ángulos referidos a parámetros en las articulaciones del propio modelo, este fichero con extensión "*.mot*" posteriormente se podrá usar como entrada para representar el movimiento final en el visualizador de *OpenSim*, donde además se podrá realizar alguno de los análisis disponibles que tiene *OpenSim* en su biblioteca. En la *Figura 21* se muestra el esquema del proceso descrito que finalmente se acabará incluyendo de forma automatizada en este proyecto de rehabilitación [28].

Otra cuestión interesante a la hora de mostrar los resultados de las sesiones a los terapeutas, es que el fichero generado al finalizar el proceso contendrá el movimiento en función de parámetros biomecánicos con ángulos de abducción, de flexión, etc. Estos datos se pueden graficar en la página web e incluso hacer análisis históricos para que se puedan apreciar los avances conseguidos por el paciente con el proceso de rehabilitación.

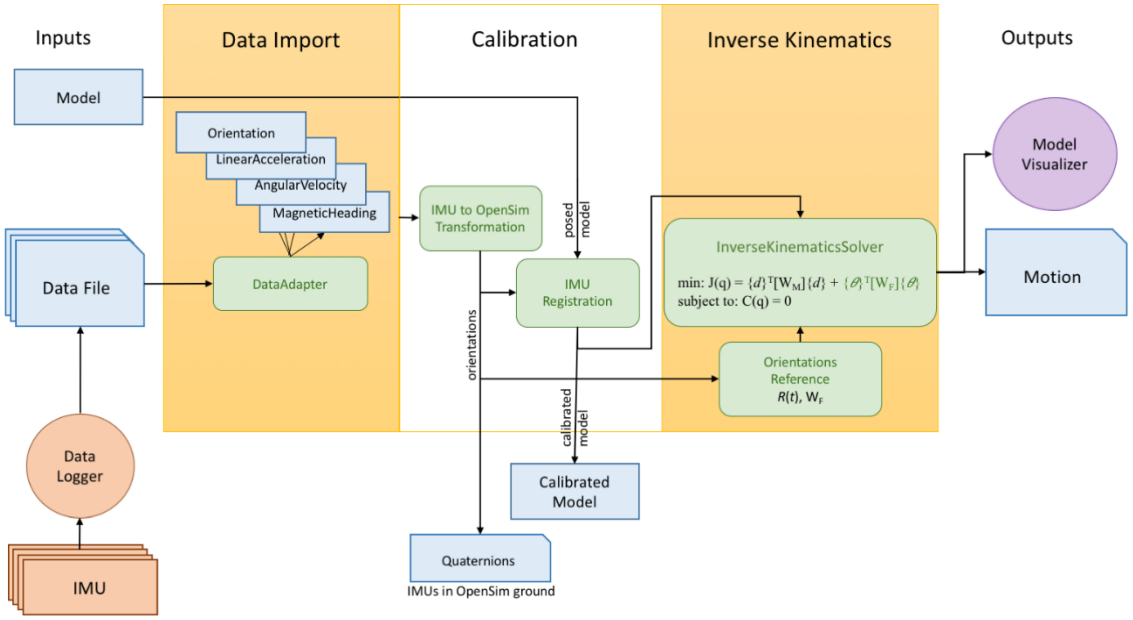

Figura 21 : Proceso de cinemática inversa con OpenSens.

4.5.2 Desarrollo de script para OpenSim

El objetivo de este desarrollo es la automatización del proceso que partiendo del fichero inicial de captura de movimientos del paciente pueda llegar a la generación de la salida ya procesada con extensión "*.mot*", utilizando para esto el procesamiento de cinemática inversa de *OpenSense* descrito anteriormente en el servidor de forma autónoma.

Además, se requiere que esta implementación se realice intentando conservar la flexibilidad para que se puedan adaptar diferentes tipos de formatos de entrada y que se pueda utilizar no solo para el procesamiento de la salida obtenida por los sensores *Xsens Dot* de este proyecto, sino que pueda interpretar cualquier tipo de formato generado por cualquier tipo de sensor inercial que se pueda llegar a incorporar en un futuro.

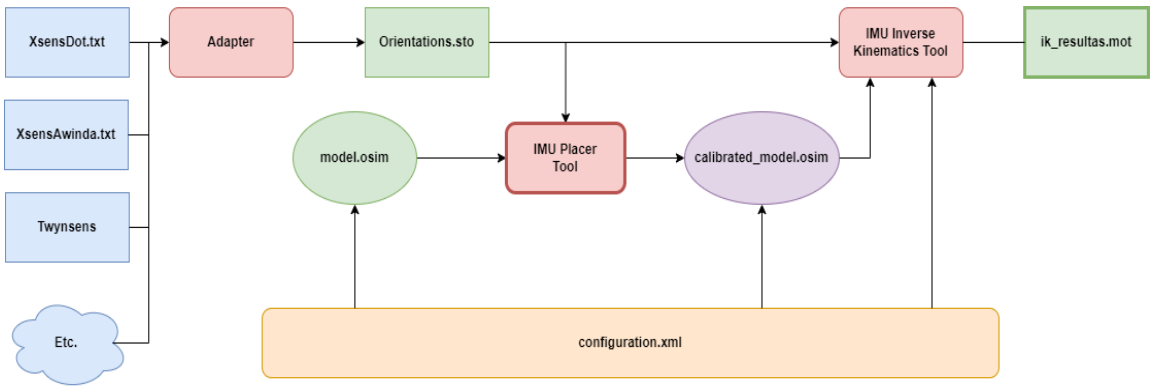

Figura 22 : Diagrama de flujo del Script adaptador a OpenSim.

En el diagrama de la *Figura 22* se muestra el esquema de funcionamiento general del *script* que se detallará paso por paso a continuación.

Primer paso: adaptación de los datos

El primer paso comienza recogiendo los datos de los sensores, el problema es que cada tipo de sensor puede tener un formato de salida diferente. Por ejemplo, la librería *Xsens Dot* presentada en el *punto 4.2* registra todas las lecturas en un único fichero de salida de texto, mientras que por ejemplo el *software* de captura de los sensores *Xsens Awinda* del mismo fabricante, se genera un fichero de texto por cada sensor.

Para hacer frente a este problema de compatibilidad se requiere realizar un proceso adaptador en la fase inicial del *script* y además este *script* estará implementado con un diseño modular, de tal forma que se puedan incluir múltiples formatos de entrada al proyecto. A este módulo adaptador sólo sería necesario indicarle mediante parámetro el nombre del fichero o ficheros de entrada y él se encargará de generar un fichero con un formato de salida estándar con extensión "*.sto*".

El proceso adaptador generará un fichero de salida formateado según las especificaciones de *OpenSense* para que poder ser procesado por este, dicho formato consiste en una cabecera al inicio indicando los parámetros con los que ha sido realizada la captura como la frecuencia de muestreo, el tipo de datos o la versión del programa con la que es compatible.

A continuación, en la siguiente línea del fichero se especifican los sensores de cada segmento del cuerpo que contiene la captura y finalmente todos los datos registrados en forma de cuaternión donde en cada línea se indican cada una de las lecturas de los sensores en un instante dado.

Segundo paso: generación del modelo calibrado

En el segundo paso se utilizará la herramienta *ImuPlacer* contenida en *OpenSense*, con esta herramienta se combina un modelo musculoesquelético base con extensión "*.osim*" y la primera línea del fichero "*.sto*" que según la documentación ha de contener las lecturas de los sensores en la posición de calibración.

A partir de estos dos elementos se genera un nuevo modelo que tendrá incluidos los *IMUs* calibrados para el proceso de corrección de errores. Como ya se ha visto, el proceso de calibración consiste en que el usuario se coloque en una posición inicial de calibración, que en nuestro caso será la llamada "*N-Pose"* como se ha detallado en el *punto 4.4.3*.

Opcionalmente se podría realizar el proceso de calibración desde el *script* inicial con el algoritmo expuesto en el *punto 4.4.4* y de esa manera se puede llegar a controlar mejor los datos introducidos, y se puede subsanar los errores producidos por errores de colocación de los sensores.

Los ficheros "*.osim*" internamente son en realidad ficheros *xml* que describen todos los componentes y propiedades referentes al modelo, incluyendo desde los diferentes huesos y músculos que lo componen con sus características y capacidades, hasta las restricciones articulares que se utilizarán para la reconstrucción del movimiento.

Como ya se ha comentado anteriormente, en este proyecto se diferenciará entre dos modos de captura: uno para la monitorización del tronco superior y otro para el inferior. Por lo tanto, ha sido necesario generar dos modelos diferentes para cada uno de estos modos, partiendo de un modelo de referencia inicial de *OpenSim*.

El modelo de referencia inicial se llama "*Rajagopal2015.osim*" y se encuentra dentro de la biblioteca pública de modelos de *OpenSim*, este ha sido desarrollado por *Adrian Lai, Allison Arnold y James Wakeling* y se trata de un modelo musculoesquelético humano refinado adecuado para el análisis de movimiento especialmente detallado para miembros inferiores.

A partir de este modelo de referencia se han creado dos modelos diferentes, uno para la reconstrucción de movimientos de la parte inferior llamado "*GTI_Bottom.osim*" y otro para la parte superior llamada "*GTI_Upper.osim*", ligeramente modificado con respecto al original para que la posición inicial de calibración sea la "*N-Pose"* que se ha establecido en las especificaciones con los brazos apoyados en los laterales del cuerpo.

En la *Figura 23* se pueden ver los dos modelos utilizados en el proyecto, aunque el *script* está adaptado para poder utilizar cualquier modelo que se desee con solo cambiar el fichero de configuración *xml*.

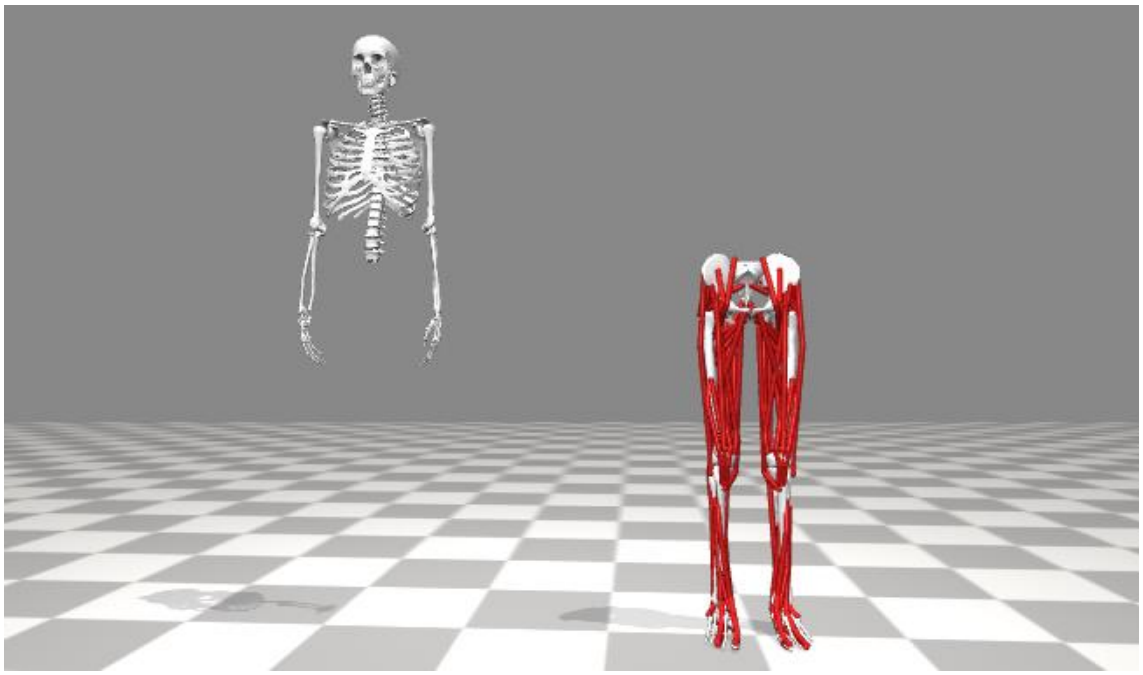

Figura 23 : Modelos GTI_Upper.osim izquierda y GTI_Bottom.osim derecha.

Con la combinación de uno de estos dos modelos como base y la primera línea del fichero "*.sto*" de orientaciones, se generará mediante la herramienta *ImuPlacer* una versión del modelo calibrado con los sensores situados sobre cada segmento del cuerpo correspondiente con las rotaciones iniciales.

Tercer paso: procesado de cinemática inversa

El tercer paso consiste en la realización del proceso de reconstrucción de movimiento con la herramienta "*IMU Inverse Kinematiks Tool*", esta herramienta utiliza como entrada el fichero de orientaciones creado en el primer paso y el modelo musculoesquelético calibrado que se acaba de crear en el segundo paso sobre el que realizará los cálculos.

El resultado final del proceso de cinemática inversa será finalmente un fichero de extensión "*.mot*" que contiene el resultado de la simulación en función de parámetros biomecánicos como flexión o abducción de cada articulación. Este fichero permite reproducir el movimiento con solo importarlo a la interfaz gráfica de *OpenSim*.

Todo el proceso de generación será completamente configurable y podrá ser fácilmente modificado en el fichero de configuración *"config.xml"*, que contiene una configuración diferente para cada tipo de entrada al programa de tal manera que se podrá gestionar de manera independiente cada procesado.

En la siguiente *Figura 24* se muestra el resultado final de una captura de la parte superior, de un ejercicio consistente en lanzar una pelota por encima de la cabeza.

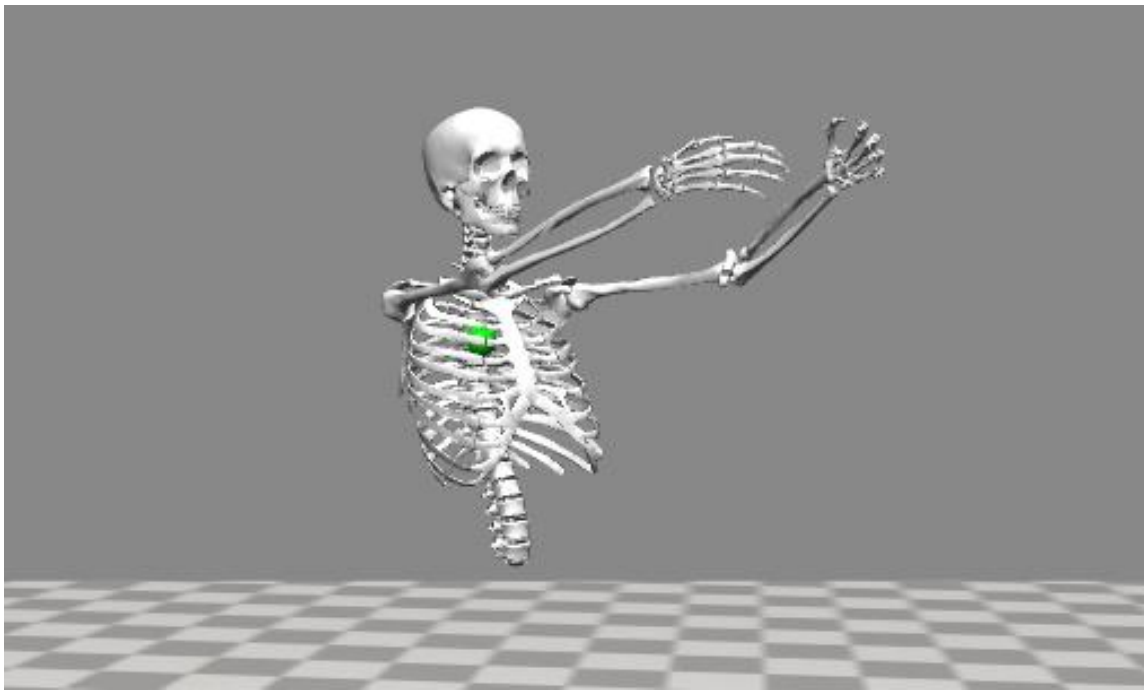

Figura 24 : Simulación de lanzamiento de pelota en OpenSim.

Los datos procesados de las capturas pueden ser enviadas al terapeuta para que pueda reproducir en la clínica el ejercicio que ha realizado el paciente en su casa, e incluso puede realizar procesos de análisis sobre la captura y comparaciones en el tiempo que le permitirán ver las mejoras a lo largo del proceso de rehabilitación o las necesidades específicas del paciente.

Aunque esta herramienta ha sido desarrollada para tener la máxima flexibilidad y adaptabilidad a todo tipo de fuente de datos inerciales, se utilizará una versión más simplificada en el proyecto de rehabilitación en el servidor para los sensores *Xsens Dot* en concreto.

4.6 Desarrollo de interfaz gráfica para bot

La infraestructura del bot de telerehabilitación del que se parte forma parte del trabajo de fin de grado de *Javier Delgado Rodríguez* en *2021* titulado "*Sistema de tele-rehabilitación a domicilio con chatbot y microordenador de bajo coste*" [21].

Uno de los principales obstáculos que se han presentado a la hora de incorporar los sensores de movimiento al bot es que este todavía no contaba con interfaz gráfica, toda la interacción entre el paciente y el bot era exclusivamente a través del chat de *telegram*, donde el usuario enviaba órdenes o recibía mensajes de información.

Esto resultó un problema ya que como se ha visto en el *punto 4.4,* se requiere añadir la funcionalidad de monitorización con sensores al proyecto y esto conlleva ciertas dificultades añadidas para el paciente, como operaciones de calibración y colocación de los sensores. Por lo tanto, sería de gran ayuda que se pudieran mostrar imágenes o videos con las instrucciones de colocación de los sensores y también mostrar retroalimentación al paciente en el proceso.

Inicialmente este proyecto está pensado para que los usuarios del bot de telerehabilitación sean mayoritariamente niños en rehabilitación junto a un acompañante, de modo que tener una pantalla en la que aparezcan indicaciones con pequeños dibujos puede resultar interesante para hacer más entretenida la sesión.

Incluso en un futuro se podría añadir un enfoque más cercano a las mecánicas de un juego, incluyendo recompensas por el cumplimiento de objetivos en los ejercicios y aumentando así la motivación del paciente a cumplir con su rutina de rehabilitación para ir mejorando sus resultados como ya han demostrado los estudios [4].

4.6.1 Estructura de la interfaz

Aprovechando los conocimientos en el entorno de desarrollo de *Pygame* adquiridos durante la realización de los proyectos anteriores, como por ejemplo el visualizador de cuaterniones visto en el *apartado 4.3*, se va a desarrollar una interfaz gráfica relativamente simple para dar instrucciones al usuario durante las

sesiones de rehabilitación mostrando tanto imágenes a modo de diapositivas interactivas, como videos.

Un requisito fundamental para esta interfaz es que se integre de forma totalmente paralela a la ejecución del programa principal con un hilo paralelo y que además esta interfaz pueda ser controlada a través de llamadas asíncronas desde el hilo de ejecución principal, de tal manera que no se produzcan llamadas bloqueantes que puedan interrumpir el flujo normal de ejecución.

Para hacer frente a estos requerimientos se hará uso de la librería *Python asyncio*, librería ya utilizada anteriormente como parte del desarrollo del proyecto para la conexión *Bluetooth* de los *Xsens Dot* detallado en el *punto 4.2*.

Asyncio (Async-Input-Output) permite escribir código concurrente utilizando una sintaxis *async/await*. Esta librería se ha utilizado como base en múltiples *frameworks* que requieren operaciones asíncronas manteniendo un alto rendimiento en *Python*.

Algunas de las características interesantes de la librería *asyncio* serían: capacidad de ejecución y control de varias corrutinas simultáneamente, incorporación de subprocesos de control, distribución de tareas a través del uso de colas y capacidad de sincronización de código concurrente.

Uno de los problemas que surgió durante la realización de este proyecto con respecto a *Pygame* es que cuando se inicializaba en un hilo determinado con la llamada a la función *pygame.init()*, se cargaban todos los módulos y variables en ese espacio de trabajo, de esta manera si se trata de ejecutar alguna función de *Pygame* en un hilo paralelo suele causar muchos problemas ya que no se ha inicializado previamente el entorno en ese hilo.

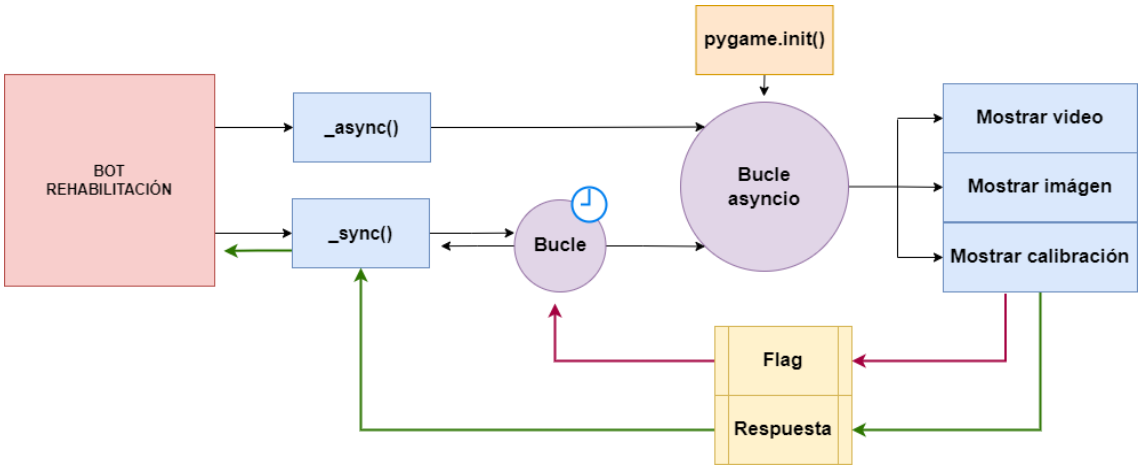

Figura 25 : Diagrama de llamadas síncronas y asíncronas.

Para hacer frente al problema se implementará la interfaz gráfica con todas sus funciones sobre un mismo objeto llamado *teleMedInterface*, este objeto va a contener un bucle de tipo *asyncio* ejecutándose en un hilo paralelo a la espera de las diferentes llamadas asíncronas a sus métodos. Este bucle será dónde se

inicializará todo el entorno de *Pygame* realizando las configuraciones para el funcionamiento y cargando los módulos que se van a utilizar en el momento de la instanciación del objeto.

En alguno de los procesos de los que se hablará más adelante, se necesitará que la llamada sea bloqueante y que tenga que esperar a recibir la respuesta de la interfaz antes de continuar, de modo que se van a utilizar dos atributos del objeto, uno que servirá de *flag* o indicador de que el proceso ya ha finalizado y el otro para almacenar la respuesta del proceso asíncrono, de esta forma solamente se tendrá que implementar un bucle que espere por el *flag* de confirmación y que devuelva la respuesta almacenada en el otro atributo.

Para no correr el riesgo de que surja algún error en tiempo de ejecución y el proceso pueda quedar esperando de manera indefinida, se ha introducido también un temporizador para devolver un fallo en caso de que se sobrepase el tiempo de espera máximo. En la *Figura 25* se muestra el diagrama de los procesos de llamadas y la estructura de funcionamiento.

4.6.2 Funcionalidades

La interfaz tendrá tres funcionalidades principales que se expondrán a continuación y una funcionalidad extra en la que durante la realización del ejercicio se mostraría el avatar del proyecto expuesto en la *sección 4.4*, pero actualmente esta funcionalidad no está siendo utilizada por falta de flexibilidad y problemas de rendimiento sobre la *RaspberryPi*.

Mostrar imágenes

Mostrar imágenes es una de las funciones más utilizadas en interfaz y existen dos variantes de funcionamiento: el funcionamiento base que consiste en mandar una imagen por pantalla a modo de diapositiva simple de forma completamente estática y el otro modo de funcionamiento que sería colocar una imagen base y sobre esta escribir un texto determinado.

Esta última funcionalidad resulta más dinámica y permite la construcción de una escena con partes diferenciadas durante la ejecución, de esta forma se abre la posibilidad de cambiar el texto en tiempo real de forma dinámica, e incluso poder tener varios lenguajes sobre una misma implementación de la interfaz.

Pygame tiene un módulo específico para mostrar imágenes que nos permite cargarla en un objeto tipo *pygame.image* y posteriormente se redimensionará para que se muestra por pantalla perfectamente centrada.

Esta función se llamará de forma totalmente asíncrona ya que no se requiere la respuesta de la interfaz para continuar con la ejecución.

En la *Figura 26* se puede ver una captura de lo que aparecería por pantalla con la función de imagen, en la imagen de la izquierda se ve la diapositiva que aparecerá por pantalla en la televisión y en la de la derecha aparece lo que estaría viendo en su móvil el usuario a través de *Telegram*.

Figura 26 : Captura de pantalla con función de imagen.

Mostrar videos

El objetivo principal de esta función es mostrar el video al paciente, descargándolo del servidor y reproduciéndolo en bucle durante la realización de los diferentes ejercicios de rehabilitación a modo de guía.

Las proporciones del video recibido pueden variar dependiendo de la posición en la que fue grabado o el dispositivo en cuestión, por lo tanto se va a necesitar reescalar y centrar las imágenes respetando siempre la relación de proporciones para evitar las distorsiones.

Para la reproducción de vídeo se ha utilizado el módulo *cv2* de la librería *opencv-python*, una librería diseñada para el desarrollo de soluciones de visión artificial, en este caso la parte interesante de este módulo no tiene nada que ver con la visión artificial, sino con la facilidad que nos da la librería para recoger y modificar uno a uno todos los fotogramas de un video e ir mostrándolos por pantalla.

El video se repetirá en bucle hasta que el usuario indique a través del bot de *Telegram* que ha acabado todas las repeticiones, de modo que se utilizará un atributo como en el caso anterior a modo de indicador o *flag* para que salga del bucle cuando se le indique.

En la *Figura 27* se muestra la captura durante la función de reproducción video, la imagen de la izquierda sería la salida en la televisión donde se estará reproduciendo el video en bucle hasta que no se le indique a través de *Telegram* que se ha terminado pulsando en "*Completado*" y en la imagen de la derecha se muestra lo que se vería en el dispositivo móvil del paciente con *Telegram*.

Figura 27 : Captura de pantalla con función de vídeo.

Proceso de calibración

El proceso de calibración es indispensable cuando se trata de realizar capturas de movimiento con sensores inerciales, para que luego sea viable realizar la reconstrucción del ejercicio, este punto ya se ha discutido con más profundidad en el *apartado 4.4.3* del documento.

Para el proceso de calibración se necesitará que el usuario se coloque en la posición "*N-Pose"* durante unos segundos, es en este momento dónde se recogerán las medidas de todos los sensores como referencia para la calibración.

La realización correcta por parte del paciente de este proceso es crítica para poder tener capturas válidas, por este motivo se ha incluido un procedimiento específico de verificación para comprobar que el paciente se ha colocado en el lugar y orden correcto todos los sensores.

El primer paso será el de la colocación de los sensores sobre el paciente, donde dependiendo del tipo de ejercicio o ejercicios que se vayan a realizar saldrá una pantalla con las diferentes indicaciones de colocación, bien para la monitorización de la parte de arriba o para la parte de abajo. En la *Figura 28* se pueden ver las dos variantes de esta pantalla.

Una vez que el paciente termina la colocación de todos los sensores y confirma la finalización del proceso a través de *Telegram*, se procede al siguiente paso de verificación. En esta verificación se comenzarán a recoger las lecturas de todos los sensores de forma continua y se comprobarán que las rotaciones relativas se corresponden a las esperadas que se obtendrían sobre un modelo ideal. Además, se

incluirá un margen de 45 grados para que el sistema se pueda adaptar a la mayor parte de los casos y pacientes.

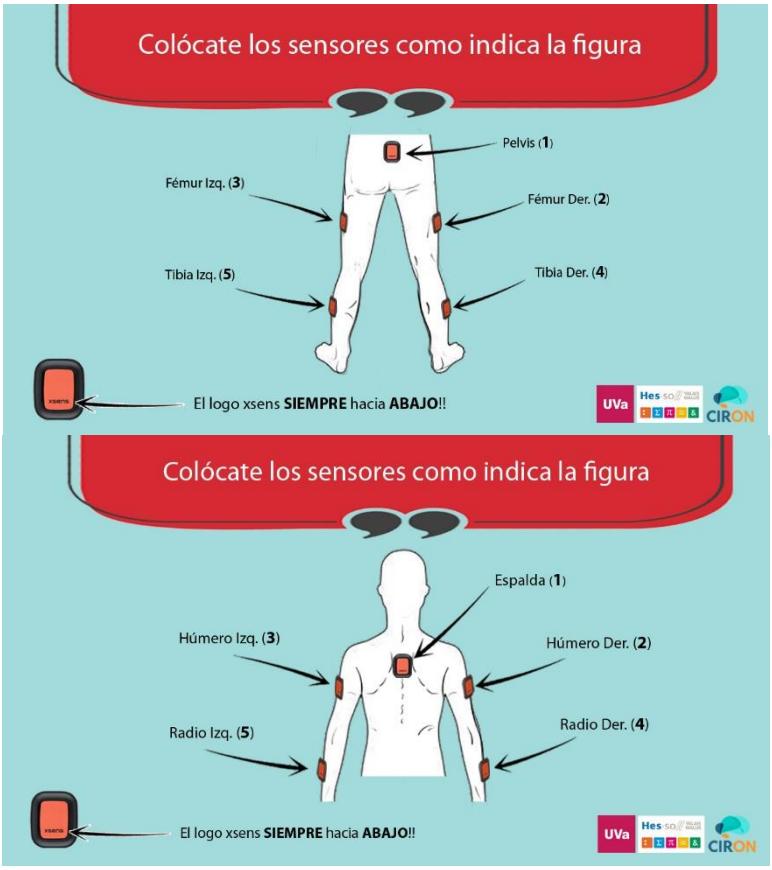

Figura 28 : Indicaciones para la colocación de los sensores.

Para realizar el cálculo de verificación, se toma el cuaternión del sensor de la espalda o la pelvis como base o "padre" siempre marcando esta lectura como correcta, a partir de este cuaternión se calcularán los giros relativos a todos los demás cuaterniones registrados emplazados en las extremidades. Aunque ya se explicó las operaciones básicas con cuaterniones en el *punto 2.7*, se puede ver el proceso para hallar este ángulo relativo en la *ecuación 18* siguiente.

$$({}^{hab}Q_{back})^{-1} * {}^{hab}Q_0 = {}^{back}Q_0 \qquad (18)$$

Una vez halladas la rotación relativa respecto a la espalda ${}^{back}Q_0$, hay que comparar esta rotación con la esperada ${}^{back}Q_{esp}$ respecto al mismo sistema de referencias, para esto se realizará la siguiente operación que se indica a continuación.

$$({}^{back}Q_{esp})^{-1} * {}^{back}Q_0 = {}^{esp}Q_0 \quad (19)$$

Con esto finalmente se obtendría el cuaternión de rotación relativa de la esperada respecto de la recibida ${}^{esp}Q_0$. Para que resulte más sencilla la comparación y la aplicación de los márgenes se pasa el cuaternión a ángulos de *Euler* y se compara el valor absoluto de cada componente *x, y, z* con los 45 grados establecidos como margen.

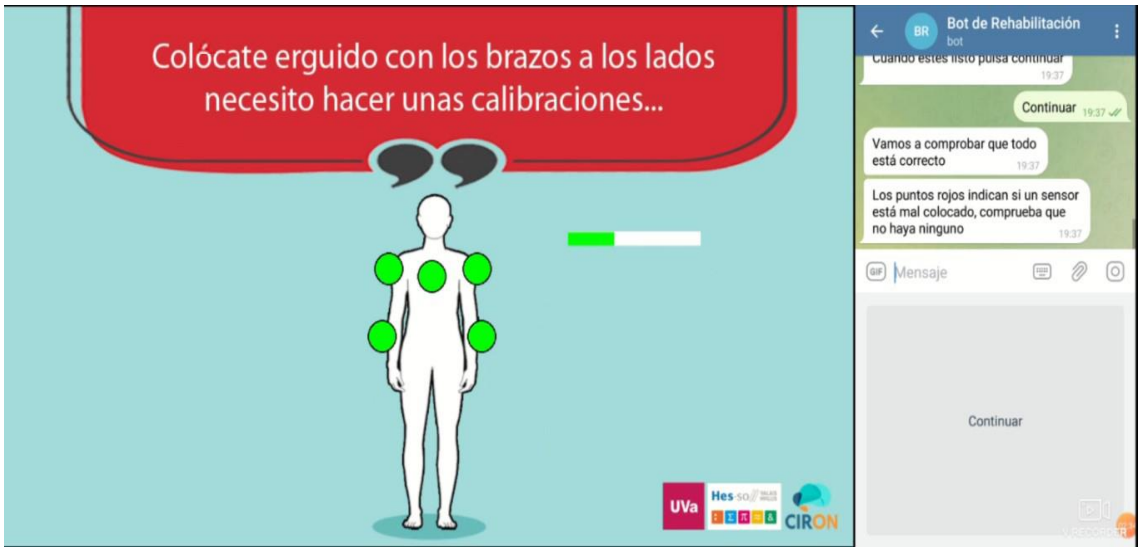

Figura 29 : Captura de pantalla con función de calibración.

Si los valores están dentro del margen se comenzará a llenar una barra de progreso durante cinco segundos, cuando esta llegue a su máximo se procederá a tomar la medida para el proceso de calibración, una vez tomada la muestra se continúa con el flujo normal del programa procediendo a realizar las capturas.

Además, se ha incluido una serie de círculos sobre una imagen donde estarán colocados los sensores que se han detectado y que cambiarán de rojo a verde en función de si el sensor en esa posición está dentro o no de los márgenes establecidos, de esta forma el paciente podrá saber cuál de ellos es el que está en el sitio incorrecto o demasiado desviado de la posición esperada. La pantalla que verá el usuario en la televisión será similar a la que se muestra en la *Figura 29*.

El proceso de calibración está implementado como un proceso bloqueante, de tal manera que se ha establecido un tiempo de espera máximo para la calibración, si pasado este tiempo no se ha conseguido un resultado positivo se le da la opción al usuario de reintentar nuevamente desde el primer paso o bien continuar con la rehabilitación sin el uso de los sensores.

4.7 Desarrollo de gráficas para web

Una vez han sido enviadas al servidor las capturas de los ejercicios de rehabilitación, serán subidas al servidor donde se procesarán tal como se ha explicado detalladamente en el *punto 4.5* a través de la herramienta *OpenSim*.

Al final de este proceso lo que se obtiene será un fichero con extensión "*.mot*", que contendrá una representación sobre un modelo musculoesquelético de lo que el usuario ha hecho durante el ejercicio.

Cuando el terapeuta se dirija a la página web donde se encuentre el informe de la sesión realizada podrá además descargar ese fichero para reproducirlo en su propio ordenador si lo necesita, además de poder realizar todo tipo de análisis incluidos en la herramienta de *OpenSim*.

Este fichero con extensión "*.mot*" es un fichero de texto con un formato interno muy similar a un *csv*, contiene los movimientos capturados con parámetros relativos a parámetros biomecánicos de las articulaciones del modelo como por ejemplo abducción, flexión, rotación, etc. El objetivo de este pequeño proyecto será hacer uso de todos estos datos para generar gráficas en la web interactivas y que pudieran ser de utilidad para las labores del terapeuta.

4.7.1 Generador de gráficas

El servidor web del proyecto utiliza el entorno de desarrollo de *Django* escrito en *Python* como el resto de desarrollos y sigue el patrón de diseño conocido como *modelo-vista-controlador* (*MVC*).

Figura 30 : Gráfica de codo derecho e izquierdo en un ejercicio.

Cuando el terapeuta acceda al informe de la sesión, podrá pedir un análisis gráfico de las capturas de los ejercicios y cuando esto suceda, el servidor interpretará los datos de los ficheros "*.mot*" y generará con ellos una serie gráficas interactivas con ayuda de la librería *ChartJs* presentada en el *punto 2.6*.

Un ejemplo de la salida de este *script* es el que se muestra en la *Figura 30*, donde se puede ver la comparación del grado de flexión de la articulación del codo derecho y del izquierdo durante la realización de un ejercicio.

Además, *ChartJs* proporciona la flexibilidad de poder realizar la superposición de datos sobre una gráfica si se necesitara o incluso poder realizar consultas históricas de los resultados del mismo ejercicio a lo largo del tiempo, esta es una característica interesante para que el terapeuta pueda ver los progresos logrados por el paciente.

4.8 Visión general de los desarrollos

A lo largo de este capítulo se han presentado múltiples desarrollos en muchos puntos diferentes del proyecto, de manera que para tener una visión más general se hará un breve repaso para comprender en qué punto del proceso interviene cada uno de estos desarrollos.

Como se indica en la *Figura 31,* se van a diferenciar tres etapas diferentes en el tratamiento de datos.

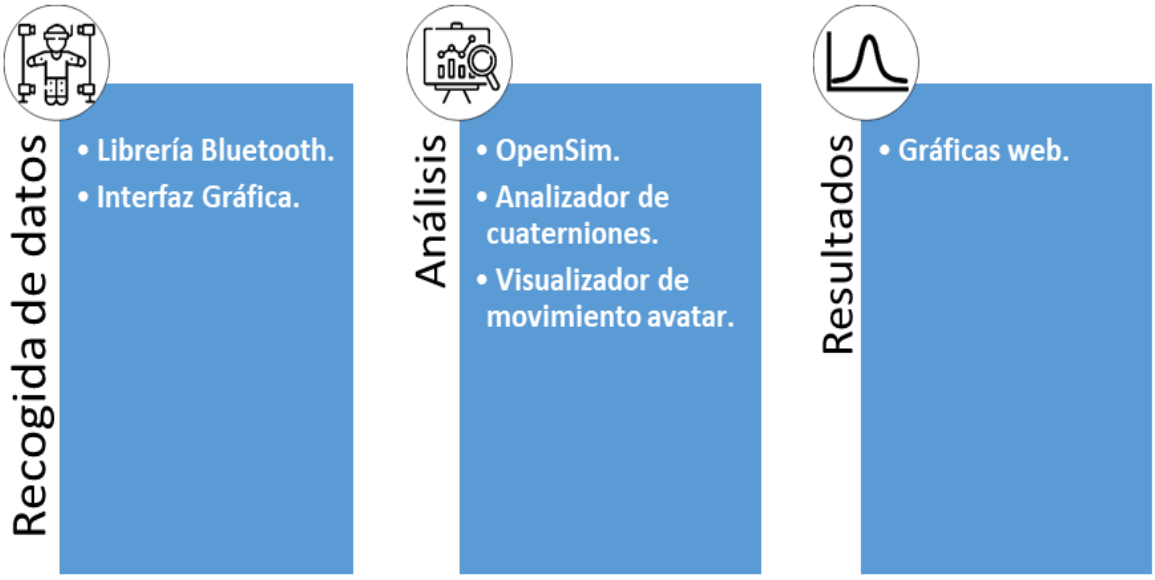

Figura 31 : Visión general de los desarrollos.

Se distinguen entre tres etapas del tratamiento de datos:

- **Recogida de los datos**
  - *(4.2)* Librería *Bluetooth* desarrollada en *Python*: permite la gestión de sensores inerciales de forma sencilla y registrar las lecturas de orientación durante el desarrollo de los ejercicios del paciente.
  - *(*4.6*)* Interfaz gráfica de usuario para el bot: permite dar instrucciones más claras al paciente para la utilización de los sensores, especialmente importante durante la calibración. Además, mejora notablemente la experiencia de usuario.
- **Análisis de los datos**
  - *(*4.5*)* Adaptador de capturas para *OpenSim*: esta herramienta se encarga de realizar un procesado de cinemática inversa y reconstruir el movimiento del paciente para un posterior análisis.
  - *(*4.3*)* Representador de cuaterniones: con esta herramienta se puede realizar el análisis de capturas y visualizar las orientaciones de cada sensor a lo largo del tiempo, resulta especialmente útil para comprobar que el proceso de calibración parte de una buena colocación.
  - *(*4.4*)* Visualizador de movimiento en avatar: analizador especialmente útil para comprobar sin ningún tipo de procesamiento que pueda afectar, cuáles son las lecturas reales de los sensores y cuál es el efecto de corrección de calibración de una forma visual.
- **Presentación de resultados**
  - *(*4.7*)* Representación de gráficas en web: permite generar gráficas a partir de los análisis de *OpenSim* y presentárselas de forma visual y sencilla al terapeuta. Además, en esa misma web podrá descargar directamente el fichero generado y realizar la reproducción del movimiento en su ordenador.

Capítulo 5 Conclusiones y líneas futuras

5.1 Conclusiones

Los acontecimientos vividos durante la crisis sanitaria de *2020*, han puesto de manifiesto la necesidad inmediata de incorporar a los procesos de digitalización el sistema sanitario, desarrollando nuevas herramientas tecnológicas que puedan servir de soporte para ofrecer un mejor servicio adaptado a las nuevas circunstancias.

El área de la medicina de rehabilitación ha sido una de las más afectadas, ya que ha aumentado significativamente el número de pacientes que han acabado por interrumpir su terapia con consecuencias graves de cara a la recuperación.

En este contexto soluciones de telerehabilitación como las que se propone a lo largo de este proyecto, pueden resultar sumamente interesantes para las clínicas mejorando el servicio y disminuyendo los costes.

Este trabajo toma como base el proyecto de telerehabilitación ya desarrollado por *Javier Delgado Rodríguez* en su *TFG* [21]. Este proyecto consistía en una web sobre la que los terapeutas establecían sesiones de rehabilitación y una *RaspberryPi* en la casa del paciente que mostraba y controlaba la realización de la sesión.

El objetivo principal de este trabajo ha consistido en la integración de los sensores inerciales *Xsens Dot* al proyecto de telerehabilitación anteriormente descrito, esta integración ha consistido en pequeños subproyectos alrededor de los sensores inerciales y el tratamiento de las medidas realizadas.

El primer objetivo alcanzado ha sido la creación de una librería *Bluetooth* en *Python* para la conexión de los *Xsens Dot* a la *RaspberryPi*. Para este subproyecto se ha utilizado como base la librería *Bleak* que ofrece funcionalidades asíncronas para la conexión *Bluetooth 5.0* y la documentación que proporciona *Xsens* sobre los servicios *BLE* de los sensores, para crear una librería que permite realizar la captura y gestión de todas las funcionalidades a un grupo de sensores de forma simultánea. Esta librería es completamente modular y se ha integrado fácilmente en el programa principal que contiene el bot de la *RaspberryPi*.

El segundo objetivo alcanzado ha sido la creación de una interfaz gráfica sencilla para el bot con la utilización de *Pygame* con varias funcionalidades como mostrar diapositivas y visualizar videos durante los ejercicios. Esta interfaz permite dar instrucciones más claras sobre el uso de los sensores al paciente y proporciona *feedback* de las lecturas de los sensores durante su uso.

El desarrollo de la interfaz gráfica ha resultado muy interesante para el proyecto final, no solo por proporcionar una experiencia de usuario notablemente mejor,

sino porque el propio proceso de calibración de los sensores puede resultar complejo y es vital su correcta realización para obtener datos fiables con el que poder reconstruir luego el ejercicio.

El tercer objetivo alcanzado ha sido poder analizar y tratar de forma fiable los datos recogidos por los sensores durante el ejercicio, para esto se han creado dos herramientas a través de *Pygame* y *PyOpenGL* con los que se ha logrado representar en un espacio tridimensional las lecturas de los sensores.

La primera herramienta consiste en una representación simple de las rotaciones de los sensores durante la captura, esta resulta especialmente útil para comprobar que el proceso de calibración ha sido realizado correctamente. La segunda herramienta es una representación de las lecturas sobre un avatar simple en tres dimensiones de la captura, sobre este avatar se ha aplicado un algoritmo para la corrección de errores de colocación con un proceso de calibración inicial, esta última herramienta ha resultado de especial utilidad para poder visualizar los datos directos de los sensores sin ningún tipo de procesado y ver el efecto de la corrección de errores de una forma inmediata.

El cuarto objetivo alcanzado ha sido la adaptación de los datos recogidos por los sensores a *OpenSim*, software encargado de realizar la reconstrucción del movimiento del paciente a través de un procesado de cinemática inversa sobre un modelo musculoesquelético. Lo interesante de esta fase que será llevada a cabo en el servidor, es que su fichero de salida permite realizar posteriormente procesos de análisis cuyos resultados pueden resultar interesantes para el personal clínico.

El quinto y último objetivo alcanzado ha sido el desarrollo de un pequeño *script* en *Python* capaz de construir gráficas a partir de los datos generados en el proceso anterior con el uso de la librería *ChartJs*. Si bien esta herramienta ya es plenamente funcional y está pensada de forma modular para mejorar la integrabilidad, todavía no se ha implementado dentro del servidor web de *Django*.

En su conjunto el desarrollo de este proyecto tiene implicaciones muy interesantes tanto para la parte de los pacientes, como para la parte del personal médico.

En lo que se refiere al paciente, le proporcionará una mayor comodidad para realizar los ejercicios en su propio domicilio mientras es supervisado remotamente por un terapeuta, algo muy interesante para pacientes con movilidad reducida a los que les pueda suponer un problema el desplazamiento a la clínica.

En lo que se refiere a las clínicas, proyectos como este les permitirá atender de forma simultánea a un mayor número de pacientes de forma simultánea, contribuyendo a la mejora del estado de saturación del sistema sanitario y disminuyendo los costes por paciente de la terapia de forma que los pacientes con menos recursos también puedan acceder a estas.

En lo personal, creo que este proyecto resulta muy interesante para las clínicas como forma de adaptación a las nuevas necesidades que presenta la sociedad. Este proyecto no debería quedarse en ningún caso como un simple *TFG*, sino que debería seguir creciendo para convertirse en un producto que contribuya a la mejora en el servicio sanitario y la calidad de vida de las personas.

5.2 Líneas futuras

Este proyecto ya ha demostrado ser de interés en el ámbito científico y en el mercado de dispositivos de telerehabilitación, presentándose como un proyecto de largo recorrido con un sinfín de ideas interesantes detrás para ser desarrolladas.

Interfaz Gráfica

Una característica fundamental para el paciente es la facilidad de uso que presenta el producto, especialmente cuando se trata de tratamiento a personas de avanzada edad, de tal manera que desarrollar una interfaz intuitiva es clave para la experiencia de usuario.

Otra característica que considero muy interesante a desarrollar sería el aumento de interactividad de la interfaz, ya que hasta ahora está limitado a simples imágenes y videos. Una pantalla táctil o incluso algún tipo de mando podría ser un elemento muy interesante a incorporar y que cambiaría por completo la experiencia de usuario.

Sensores

La incorporación de los sensores al proyecto son una vía de desarrollo que abre las puertas a aplicaciones muy interesantes, una de estas aplicaciones podría ser el desarrollo de pequeños juegos interactivos con fines terapéuticos que han demostrado ser muy útiles aumentando la motivación del paciente para realizar sus tratamientos.

Actualmente en este proyecto se están utilizando cinco sensores para la monitorización de la parte superior o de la inferior, pero hay estudios que apuntan a que a través de la aplicación de inteligencia artificial podría reducirse el número de sensores necesarios pudiendo monitorizar todo el cuerpo con el mismo número de sensores.

Otra rama muy interesante podría ser la incorporación de más tipos de sensores al proyecto que puedan proporcionar datos clínicos relevantes, como frecuencia cardíaca, presión arterial, etc.

Web

La web es un elemento vital para el proyecto, ya que es el medio por el que las clínicas van a interactuar con el paciente, por esta razón hay que seguir trabajando en su desarrollo constantemente aumentando su funcionalidad y mejorando su aspecto para ofrecer la mejor experiencia de usuario posible.

Un punto que considero vital es el aspecto de la seguridad ya que se trata de una aplicación médica. Es imperativo cuidar durante el desarrollo las medidas de seguridad en el tratamiento de datos y que todo esté de acuerdo a la normativa impuesta en la ley general de protección de datos.

REFERENCIAS

[1] INE "*Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia 2020*" [online] (2020). Disponible en: https://www.ine.es/ Fecha de última consulta: Junio 2022.

[2] SERMEF "*Conclusiones de un studio realizado en diversos hospitals de España: Más de la mitad de los pacientes hospitalizados por COVID-19 necesitan rehabilitación al alta*" [online] (2021). Disponible en: https://www.sermef.es/wp-content/uploads/2021/06/50-hospitalizados-COVID-RHB.pdf Fecha de última consulta: Junio 2022.

[3] SERMEF "*Los médicos rehabilitadores instan a la nueva ministra de sanidad a: mediar con las comunidades para no cerrar los servicios de rehabilitación para atender la tercera hola*" [online] (2021). Disponible en: https://www.sermef.es/wp-content/uploads/2021/02/NOTA_Tercera-ola.pdf Fecha de última consulta: Junio 2022.

[4] Cikajlo I., Rudolf M. y Goljar N. "Telerehabilitation using virtual reality task can improve balance in patients with stroke" (2011). *Disability and rehabilitation*.

[5] Xsens Technologies B.V. "*Xsens Dot User Manual"* (2021).

[6] Townsend K., Cufi C. y Davidson R. "*Getting Started with Bluetooth Low Energy*" (2014). *Ed. O'Reilly*.

[7] McGugan W. "*Beginning Game Development with Python and Pygame*" (2007). *Ed. Apress*.

[8] Hicks. P., Seth A., Dunne J., Delp S., et al "OpenSim Documentation – User's Guide" (2017). Disponible en: https://simtk-confluence.stanford.edu:8443/display/OpenSim/OpenSim+Documentation Fecha de última consulta: Junio 2022.

[9] Delp S., Anderson F., Arnold A. et al. "*OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement*" (2007). *IEEE Transactions on Biomedical Engineering.*

[10] Rocha H. "Learn *Chart.js: Create interactive visualizations for the web with Chart.js 2*" (2019). *Ed. Packt*.

[11] Morais P., Georgiev S. y Spröbig W. "*Real Quaternionic Calculus Handbook*" (2014). *Ed. Birkhäuser*.

[12] Dunn F. y Parberry I. "*3D Marth Primer for Graphics and Game Development*" (2011). *Ed. CRC Press*.

[13] Flynn S. et al. "An overview of a USC Rehabilitation Engineering Research Center: The use of virtual reality for a range of motor impairments" (2009). *Virtual Rehabilitation International Conference*.

[14] Mass N. *"Hocoma changes the game in at-home back training with introduction of valedo" [online]* (2014). Disponible en: http://www.multivu.com/players/English/7370551-hocoma-innovative-medical-device-therapeutic-gaming-technology-valedo-low-back-health/ Fecha de última consulta: Junio 2022.

[15] Lenfermann G., Vrugt J., Timmermans A., Bongers E. y Lambert N "*Philips stroke rehabilitation exerciser*" (2007). *Technical Aids for Rehabilitation-TAR January 2007*.

[16] Dinessen B. y Mogens B. "*The Telekat project: Telehomecare, chronic patients and the cooperating health system*" (2011). *European wide Innovation Procurement in Health and Care 2011*.

[17] Presti D., Massaroni C., Caponero M., Formica D. y Schena E. "*Monitorización cardiorrespiratoria mediante un sistema mecánico y óptico*" (2021). *Simposio internacional IEEE sobre mediciones y aplicaciones médicas*.

[18] Woodward K., Kanjo E., Taylor K. y Hunt J. "*A multi-sensor deep learning approach for complex daily living activity recognition*" (2022). *Proceedings of the 2022 Workshop on Emerging Devices for Digital Biomarkers 2022.*

[19] Hartog D., Krogt M., Burg S.; Aleo I., Gijsbers J., Bonouvrié L, Harlaar J., Buizer A. y Haberfehlner H. "Home-Based Measurements of Dystonia in Cerebral Palsy Using Smartphone-Coupled Inertial Sensor Technology and Machine Learning: A Proof-of-Concept Study" (2022). *Sensors, vol. 22, nº 12, junio de 2022*.

[20] Schlage N., Kitzig A., Stockmanns G. y Naroska E. "Development of a mobile, cost-effective and easy to use inertial motion capture system for monitoring in rehabilitation applications" (2021). *Current Directions in Biomedical Engineering, vol. 7, nº 2, octubre de 2021*.

[21] Rodríguez Delgado J. "*Sistema de tele rehabilitación a domicilio con chatbot y microordenador de bajo coste"* (2021). *TFG*.

[22] Martín Casares O. "*Evolución de sistema médico rehabilitador con chatbot conversacional en Telegram, backend en Django y ordenador de bajo coste*" (2022). *TFG*.

[23] Xsens Technologies B.V. *"Xsens DOT BLE Services Specifications"*. (2021).

[24] Bilidh H. *"Bleak Documentation" [online]* (2020). Disponible en: https://bleak.readthedocs.io/en/latest/ Fecha de última consulta: Junio 2022.

[25] Rotemberg D. y Slycke P.J. "Ambulatory Position and Orientation Tracking Fusing Magnetic and Inertial Sensing" (2007). *IEEE Transactions on Biomedical Engineering, vol. 54, nº 5, mayo de 2007.*

[26] Wouda F., Giuberti M., Belluci G. y Veltik P. "Estimation of Full-Body Poses Using Only Five Inertial Sensors: An Eager or Lazy Learning Approach?" (2016). *Sensors, vol. 16, nº 12, diciembre de 2016.*

[27] Lachaine X., Mecheri L. y Plamdon A. *"Accuracyand Repeatability of Single-Pose Calibration of Inertial MeasurementUnits for Whole-Body Motion Analysis"* (2017). *Gait & Posture, vol. 54, mayo de 2017*.

[28] Delp S., Anderson F., Arnold A. et al. "*OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement*" (2007). *IEEE Transactions on Biomedical Engineering.*

Anexos

Anexo I – Presupuesto del proyecto

En esta sección se va a realizar una estimación del coste de todo el trabajo realizado en este proyecto y el coste del *hardware* que se ha necesitado, ya que todo el *software* que se ha utilizado es gratuito.

Para calcular el coste de las horas trabajadas se parte de los datos actuales, en 2022 un ingeniero de telecomunicaciones gana un salario mensual entre 1.722€ y 3.099€ al empezar en el puesto de trabajo. Convengamos un salario de 2.000€ + 30% en concepto de costes patronales con 40 horas a la semana, saliendo aproximadamente 16.25€/h.

Con lo que se refiere al uso del portátil, siendo el precio inicial de este 750€ y estableciendo una amortización de unos 4 años, sale a unos 15.62€/mes.

ELEMENTO	COSTE
RaspberryPi 4 Modelo B+ con 8 GB de RAM	188.00€
Tarjeta micro-SD de 32GB	6.90€
Kit disipador y carcasa para RaspberryPi	12.99€
Cable HDMI a micro HDMI	7.59€
Teclado y ratón inalámbricos	12.99€
Sensores Xsens Dot	650.00€
Amortización Asus 750€ (15.62€/mes)	62.48€
Salarios 640h (16.25€/h)	10400€
TOTAL	**11340.95€**

Tabla 2 : Presupuesto del proyecto.

Por lo tanto, el presupuesto final aproximado del trabajo realizado según estimaciones serán unos **11340.95€**.