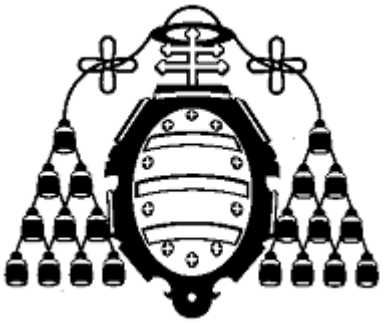

UNIVERSIDAD DE OVIEDO

CENTRO INTERNACIONAL DE POSTGRADO

MÁSTER EN INGENIERÍA MECATRÓNICA

TRABAJO FIN DE MÁSTER

MEJORA DEL CONTROL DE UN PAR DE ORTESIS ACTIVAS PARA LA MARCHA DE LESIONADOS MEDULARES

JULIO de 2015

Alumna:

Covadonga Quintana Barcia

Tutores:

Urbano Lugrís Armesto

Álvaro Noriega González

AGRADECIMIENTOS

Gracias a todos aquellos que me han ayudado en el camino que me ha llevado hasta donde estoy, especialmente a mis padres y mi hermano Pablo.

"Su sonrisa era una de esas extrañas que logras ver cuatro o cinco veces en la vida. Parecía entenderte y creer en ti justo cómo quisieras que te entendieran y creyeran…"

RESUMEN

En un proyecto anterior, el equipo de investigación con sede en el LIM investigó el uso de técnicas de dinámica multicuerpo para el diseño de ortesis activas para ayuda a la marcha y ha construido un primer prototipo para validación. Con el objetivo de comercializar este prototipo, se definen tres líneas de acción: ***diseño mecánico***, buscando mejoras en el confort del sujeto y reducción del consumo energético del dispositivo, y explorando sistemas alternativos para el bloqueo y actuación de la rodilla. ***Control***, añadiendo electromiografía (EMG) y estimulación eléctrica funcional (FES) a los sensores y actuadores existentes, respectivamente, y desarrollando estrategias de control basadas en modelos musculares y finalmente, ***simulación***, persiguiendo un modelado muscular más preciso de los sujetos lesionados capaz de considerar actuación por FES, detallando el modelo de lesionado con ortesis para dar apoyo a las líneas de diseño mecánico y control, e implementando un algoritmo de predicción del movimiento humano. Este Trabajo Fin de Máster ayudará en la investigación a portando una herramienta de ***simulación*** en la que se podrán realizar todo tipo de pruebas y cambios en el control con diferentes sensores, parámetros y herramientas de control, sin la necesidad que existía hasta el momento de realizarlas sobre la paciente que colabora en el estudio. Se ha comprobado la mejora que supone la utilización de unos sensores frente a otros y las diferentes posiciones donde deberían ir posicionados estos sensores para que la lectura y ejecución del ***control*** sea óptima.

PALABRAS CLAVE

Ortesis - IMU – Simulación dinámica - Control - Multicuerpo - Algoritmo

TABLA DE FIGURAS

TABLAS

1. Introducción

El presente Trabajo Fin de Máster, englobado dentro de un proyecto de investigación de mayor envergadura, cuya finalidad es abordar el diseño, construcción y ensayo de una ortesis activa innovadora para ayudar a la marcha de lesionados medulares incompletos. Se trata de plasmar las ideas sobre el tema de un especialista en la rehabilitación de estos pacientes, el Dr. Antonio Rodríguez Sotillo, jefe de la Unidad de Lesionados Medulares del Complejo Hospitalario Universitario de La Coruña (CHUAC), antiguo Hospital Juan Canalejo, que forma parte del equipo del primer subproyecto, en un prototipo que posea potencial para una futura comercialización. El diseño de un dispositivo semejante necesita apoyarse en técnicas de simulación del movimiento humano, que son actualmente objeto de intensa investigación en la comunidad biomecánica, por lo que el avance en este terreno es también un objetivo esencial del proyecto.

1.1. Objeto

A continuación se muestra la manera en la que se ha planteado el presente Trabajo Fin de Máster. Se detallarán las tareas que se van a realizar con el fin de mostrar una idea más pormenorizada de cada una de ellas para facilitar la comprensión de la planificación temporal de las tareas representativas del TFM.

1.1.1. Familiarización con el proyecto

En esta fase se procederá a la lectura de la documentación existente para conseguir un conocimiento más profundo del estado del arte del proyecto iniciado por el LIM al que pertenece el presente TFM.

Una vez concluida la fase documental, se iniciará la familiarización con las herramientas software comerciales disponibles así como el hardware existente en el proyecto. Además, se deberá realizar un proceso de análisis y comprensión de la programación existente con el fin de conocer el software diseñado para la ortesis objeto del presente TFM.

Se harán pruebas de los motores utilizados para el movimiento de la ortesis con el software adecuado para tal efecto, haciendo diferentes ensayos para comprobar la respuesta de los mismos ante entradas diferentes de velocidad, aceleración y corriente.

En esta fase se conocerá la respuesta de las ortesis ante diferentes estímulos inducidos por los sensores plantares incluidos en las ortesis.

1.1.2. Mejorar controlador con el material existente

Esta fase es el grueso del TFM propuesto. Se trata de explorar posibles inputs, sin restringir estas entradas a variables del movimiento que pueda producir la pierna contraria.

Esta fase del proyecto se subdivide en diferentes etapas, siendo algunas de ellas de ejecución paralela.

- Desarrollar herramienta software para conseguir realizar las diferentes pruebas del control que se debe aplicar a la ortesis sin necesidad de llevar a cabo dichos ensayos con la ortesis física.

  Esto permitirá simplificar el proceso de realización de ensayos de posibles cambios sobre el controlador dado que no sería necesario utilizar el montaje completo. Además, con esta herramienta se evitaría la necesidad de la presencia de la paciente para realizar las pruebas, dado que su disponibilidad, por asuntos laborales y personales es limitada y hace que esta herramienta cobre vital importancia.

- Realizar capturas de diferentes movimientos y velocidades. Obtener a partir de estas capturas, una base de datos que servirá para realizar las pruebas del controlador. De esta forma, se obtendrán diferentes datos en función de que el movimiento sea de traslación, giro, o cambios en las velocidades de ambos tipos de movimientos.

- Explorar la posibilidad de introducir sensores nuevos. Esta tarea se podrá realizar de forma paralela a las anteriores. Se pretende buscar soluciones nuevas que puedan facilitar y mejorar la respuesta del control existente. Se realizará un estudio de mercado de los diferentes sensores sensibles a futura inclusión en el diseño de la ortesis.

- Una vez obtenida la herramienta de ensayos de los diferentes cambios en el control, se deben probar las diferentes estrategias de control que se puedan plantear como solución al objetivo planteado en el presente TFM.

  Se deberá tener en cuenta la trayectoria descrita según posibles cambios en los ángulos descritos tanto por la rodilla como por el tobillo, y la relación entre ambos, así como la manera en que puedan afectar al funcionamiento general de la ortesis.

  Se realizará una valoración de todas aquellas posibles variables que puedan afectar al movimiento de la ortesis para tenerlas en cuenta en la formulación del controlador.

1.1.3. Migración control a BeagleBone® Black

Uno de los futuros cambios en el hardware de la ortesis será la utilización del computador embebido BeagleBone® Black en lugar del ordenador Intel NUC (sin teclado, pantalla ni batería) existente. Esto implica la conversión de la programación actual en un lenguaje compatible con la BeagleBone®. Se utilizará el software comercial QtCreator® que posee comunicación directa con el dispositivo y que utiliza lenguaje c++. Además, contiene librerías y funciones propias que facilitan la programación en el lenguaje utilizado.

La utilización de la BeagleBone® Black implica no solo ventajas en cuanto a reducción de espacio y peso del hardware existente, permitiría también la utilización de sus entradas y salidas analógicas y digitales. Además, se reduce en gran medida el consumo y aumenta la robustez del conjunto frente a sistemas basados en Windows®. Por otro lado, se facilita la comunicación con algunos sensores que podrían incluirse en el transcurso del proyecto.

Se podrá depurar el código directamente sin necesidad de comunicación directa con el ordenador en el que se haya implementado el software, lo que disminuirá el tiempo de computerizado.

1.1.4. Documentación

A lo largo de los cuatro meses asignados a la realización del TFM se irá redactando el documento memoria que reflejará la forma en que se han llevado a cabo cada una de las tareas. Se pretende añadir diagramas de flujo que proporcionen una orientación al lector más visual de lo que se ha realizado así como fotografías que faciliten la comprensión.

1.2. Condiciones de diseño

Se deben utilizar las herramientas software disponibles en el Laboratorio de Ingeniería Mecánica de la Universidad de La Coruña.

Se debe enfocar cualquier modificación al diseño mecánico existente, contemplando la imposibilidad de cambios inmediatos en el modelo ortésico, debido, principalmente, a que el diseño mecánico ha sido función de los investigadores pertenecientes a la Universidad Politécnica de Cataluña, por tanto, si se quisiera realizar algún cambio mecánico sería inviable de manera inmediata. Se debe procurar adaptar el control y la inclusión de sensores a la ortesis existente.

El control estará dirigido a la mejora del movimiento de la paciente que realiza las pruebas con las ortesis existentes, pudiendo ser extensible a nuevos pacientes siempre y cuando se obtengan las capturas de movimiento adecuadas, adaptando las entradas al control: ángulos que describen el tobillo y la rodilla, teniendo en cuenta que el control permite realizar un determinado movimiento (de marcha, de giro etc.) dependiendo del ángulo descrito por el tobillo de la pierna contraria. Esto es, el ángulo que se le introduce a la rodilla derecha, dependerá del ángulo descrito por el tobillo izquierdo. De esta forma se conocerá si el paciente está emprendiendo el movimiento de marcha o por el contrario se encuentra parado, girando sobre sí mismo o realizando cualquier movimiento que diste de comenzar a caminar.

La elección de sensores estará sujeta al presupuesto del proyecto de investigación, pudiendo realizar pruebas con sensores de precio inferior y siendo extensible a sensores de mayor calidad en caso de poder hacer frente al gasto que estos suponen.

1.3. Alcance

La ortesis es un dispositivo mecánico que se coloca alrededor de la pierna del paciente (como un exoesqueleto), y trata de paliar las deficiencias de su aparato locomotor para permitir una marcha más o menos eficiente. Las ortesis de rodilla-tobillo están dirigidas a pacientes cuyo nivel neurológico (vértebra por encima de la cual todo funciona correctamente) sea L2 (segunda vértebra lumbar) o más bajo, lo que significa que poseen flexores de cadera. Los modelos comerciales de uso más generalizado son de tipo pasivo. Constan de un sistema de bloqueo de rodilla, que evita que ésta se doble bajo el peso del cuerpo en la fase de apoyo, y de un componente antiequino en el tobillo, que evita que el pie cuelgue y tropiece con el suelo en la fase de balanceo.

El bloqueo de rodilla se activa manualmente cuando se va a caminar, por lo que la marcha se realiza con la pierna estirada en todo momento, y se desactiva, también de forma manual, para pasar a posición de sentado. Existen modelos comerciales más avanzados, pero de muy escasa difusión entre los pacientes, al menos en España, en los que el sistema de bloqueo de la rodilla se activa automáticamente en la fase de apoyo, y se desactiva también automáticamente durante la fase de balanceo,

todo ello merced a un sensor de presión situado en el talón. Estas ortesis podrían ser calificadas como semiactivas.

El tipo de ortesis que se ha desarrollado en el proyecto es de tipo activo. Su principal diferencia con las anteriores radica en que se incluye un motor en esta articulación, para ayudar a la pierna a realizar su movimiento pendular durante la fase de balanceo. Además, al sensor de presión del talón se añaden sensores angulares en tobillo, rodilla y cadera (encoders los dos primeros y electrogoniómetro el tercero). De esta manera, se cuenta con información suficiente para generar la estrategia de control más adecuada para el sistema de bloqueo y el motor de la rodilla, así como para la coordinación de las ortesis de ambas piernas.

Una mejora significativa se basa en la sustitución del motor por estimulación eléctrica funcional, consistente en aplicar pequeñas descargas eléctricas a los músculos para provocar su contracción.

Sin embargo, el objetivo del proyecto no es simplemente la construcción y el ensayo de un prototipo de ortesis activa novedoso, sino también investigar la utilidad de aplicar la dinámica de sistemas multicuerpo al diseño de estos dispositivos. Dicha disciplina ha sido utilizada desde hace tiempo para analizar la marcha humana. En esencia, se trata de generar un modelo computacional del individuo a analizar, al que se anima con el movimiento real, grabado mediante cámaras de infrarrojos que ven unos marcadores reflectantes dispuestos sobre el sujeto. También se miden las fuerzas de contacto pie-suelo mediante placas instrumentadas.

Posteriormente, se lleva a cabo un análisis dinámico inverso del movimiento del modelo, cuyo resultado son las historias temporales de los esfuerzos articulares realizados por el individuo para producir el movimiento grabado. Por lo tanto, tras el análisis se dispone de toda la información cinemática (posiciones, velocidades y aceleraciones) y dinámica (fuerzas y momentos) sobre la marcha del sujeto, información que puede ayudar a los médicos en su labor de diagnóstico y tratamiento, en caso de marcha patológica. En el contexto de este proyecto, el análisis de la marcha comporta algunas diferencias frente al caso convencional, ya que los pacientes que llevan ortesis necesitan también muletas para caminar. Por un lado, muletas y ortesis han de ser incluidos en el modelo computacional del individuo y, por otro, las fuerzas de contacto muleta-suelo han de ser medidas experimentalmente durante la marcha.

El desarrollo de esta herramienta tiene por objeto analizar el movimiento y los esfuerzos de un paciente concreto para diseñar un controlador de la ortesis que se adapte a su estilo de marcha, valorar el efecto de distintas estrategias de control de la ortesis, estudiar la adaptación del lesionado a la ortesis con el entrenamiento, y avanzar en la comprensión de la interacción entre sistema muscular y ortesis. Es de esperar que todo ello contribuya a mejorar el diseño de este tipo de dispositivos.

1.4. Hipótesis de trabajo

Se han tenido en cuenta todas las especificaciones de diseño propuestas en el enunciado del presente proyecto. Además, se ha llevado a cabo teniendo presentes algunas hipótesis de trabajo extra, que no siendo de obligado cumplimiento, se ha creído conveniente tener en cuenta en la realización del mismo.

Por un lado, se han utilizado todos aquellos componentes, necesarios, disponibles en el Laboratorio de Ingeniería Mecánica de la UDC para la puesta en marcha del montaje electrónico y de control.

Además, se han utilizado las herramientas disponibles en el mismo añadiendo material de los componentes del grupo de investigación para el montaje y prueba de diferentes componentes.

El trabajo se ha generado en torno a la ortesis existente y a las capturas de movimiento tomadas de la paciente que colabora en el ámbito del proyecto en el que se engloba el presente Trabajo Fin de Máster.

La programación realizada, los sensores elegidos y las pruebas, se han enfocado al sistema existente, pudiendo ser extensible a otros sistemas similares, teniendo en cuenta las diferencias que pudieran darse en cuanto a medidas y localización de los sensores.

Las entradas para el control serán integradas dinámicamente en relación con la paciente a la que se aplicarán, de tal forma que se deberá tener en cuenta la captura de movimiento previa para conocer la posible respuesta de la misma.

1.5. Planificación

El presente TFM se ha realizado entre los meses de febrero y mayo. El planteamiento de las tareas, explicadas en el apartado de objetivos del trabajo, se especifica mediante un diagrama Gantt en las imágenes siguientes.

Nombre	Fecha de inicio	Fecha de fin
T1-Familiarización	2/02/15	17/02/15
T2-Mejora controlador	13/02/15	28/05/15
T2.1-Desarrollo herramienta control	17/02/15	18/03/15
T2.2-Estudio nuevos sensores	13/02/15	19/05/15
T2.3-Capturas	25/02/15	13/03/15
T2.4-Estrategias de control	18/03/15	28/05/15
T2.4.1-Estudio trayectorias	18/03/15	16/05/15
T2.4.2-Estudio de variables	19/03/15	28/05/15
T3-Migración control a BeagleBone	8/05/15	30/05/15
T4-Documentación	2/03/15	22/05/15

Figura 1: Principales tareas

Figura 2: Duración de las principales tareas

1.6. Estructura de la memoria

La memoria se estructura según una serie de apartados que comienzan con una introducción en la temática del Trabajo Fin de Máster. El documento continúa con un apartado dedicado al estado del conocimiento, para situar al lector en el contexto del proyecto de investigación y la situación actual del mismo. Los siguientes apartados están dedicados al trabajo propiamente dicho: animaciones, programación, control de las entradas al sistema, sensorización y finalmente la migración a un computador embebido del software existente.

Por último, se añaden como anexos, los códigos de programación creados en el transcurso del TFM así como las hojas de datos de los componentes, esquemas de funcionamiento a tamaño completo y un análisis de árbol de fallos que puedan surgir durante el funcionamiento de la ortesis activa.

2. Estado del arte

En este apartado se realiza una breve explicación de la anatomía de la rodilla, articulación más importante que interviene en este proyecto y además, se detalla conceptualmente el proyecto completo dando a conocer detalles vinculados con el presente Trabajo Fin de Máster de los que depende en gran medida la realización del mismo. Una de las líneas es el diseño mecánico, realizado en colaboración con la UPC.

2.1. Biomecánica de la rodilla

Principalmente la rodilla cuenta con un solo grado de libertad: flexión y extensión. Este movimiento permite a la rodilla regular la distancia de separación del cuerpo con el suelo, a través de acercar o alejar el extremo de la pierna a la raíz de la misma, es decir, acercando o alejando el glúteo. Además de este principal grado de libertad, la rodilla cuenta con un segundo grado de libertad accesorio, que se presenta solo en la flexión. Este movimiento es de rotación sobre el eje longitudinal de la pierna. La articulación de la rodilla, desde el punto de vista mecánico, es sorprendente ya que realiza dos funciones que pueden ser contradictorias:

- Debe poseer mucha estabilidad cuando se encuentra en extensión completa, en este punto es donde la rodilla soporta el peso del cuerpo.
- Debe poseer gran movilidad en la flexión, dado que durante la marcha debe proveer al pie una buena orientación.

2.1.1. Huesos participantes

Los tres huesos principales que componen la rodilla son el fémur, la tibia y la rótula. El fémur es el hueso más largo del cuerpo humano. Presenta una forma oblicua hacia su parte interna, lo que permite la separación entre ambas rodillas y por tanto entre las tibias. El fémur se conecta en su parte superior con la cadera, y cuenta con cabeza, cuello, trocánter mayor y trocánter menor. Posee una forma tubular que permite la inserción de los músculos que lo rodean transmitiendo líneas de carga o fuerza desde el tronco hasta la rodilla. En la parte inferior, termina en los cóndilos y junto con la tibia, forman la articulación. Los diferentes huesos se pueden ver en la Figura 3.

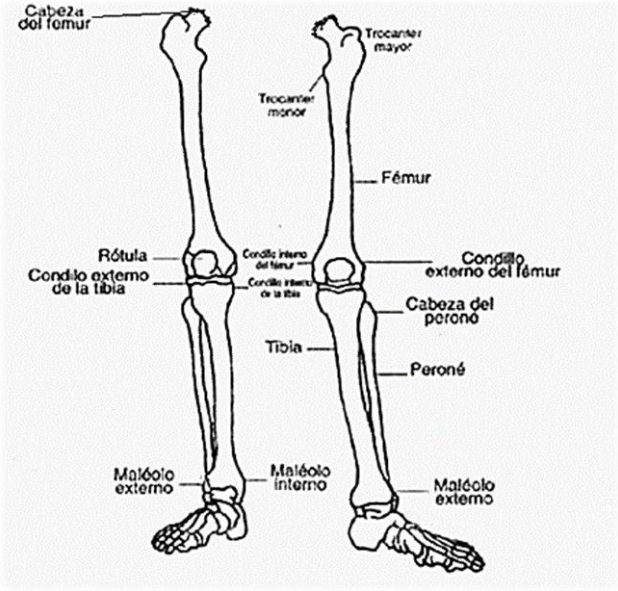

Figura 3: Extremidades inferiores. Huesos participantes

La rótula es un hueso de forma plana, de apariencia redonda u ovalada que se prolonga hacia abajo por su polo o vértice inferior. Está compuesta por dos caras:

- La cara anterior con forma convexa que sirve de polea para los tendones del cuádriceps y del rotuliano
- La cara posterior, orientada hacia el interior de la articulación. Tiene dos aspectos, interno y externo, que contactan con los cóndilos femorales, ajustando su forma cóncava con la forma convexa de los cóndilos.

Por otro lado, la tibia, forma junto con el peroné la pierna en sí misma. Es el hueso más robusto entre ambos, dado que soporta el peso corporal y es el encargado de transmitir las fuerzas de la rodilla al tobillo. En su extremo superior se encuentran los platillos tibiales, interno y externo, en los que se apoyan los cóndilos femorales. En su extremo inferior se encuentra el maléolo interno del tobillo que junto con el maléolo externo encontrado en el peroné conforman una abrazadera que soporta al astrálago.

2.1.2. Músculos y tendones

Diversos músculos y tendones cruzan la rodilla provocando sus movimientos de flexión y extensión. Es por eso que se pueden dividir en dos grupos diferentes: extensores y flexores.

Extensores

El músculo extensor más importante es el cuádriceps femoral, formado por el recto anterior, vasto interno, vasto externo y vasto intermedio, todos ellos unidos con el tendón del cuádriceps que a su vez es el tendón de mayor tamaño. Este tendón sujeta la rótula en su parte superior, pasa por encima

de ella y se convierte después en el tendón rotuliano. Su función es la de extender la rodilla manteniendo el equilibrio de la rótula para que esta pueda deslizar correctamente sobre la escotadura intercondílea.

Flexores

Estos músculos (Figura 4) se encuentran en la parte posterior del muslo y son:

- Músculo semitendinoso: provocan una rotación interna cuando la pierna está flexionada.
- Bíceps femoral: debido a que se encuentran en la parte lateral, provoca una rotación externa tras la flexión de la pierna.
- Pata de ganso: unión de tres músculos, semitendinoso, recto interno y sartorio. Este conjunto de músculos recibe el nombre de músculos isquiotibiales.
- Músculo gastrocnemio: conocido como gemelo, viene de la cara posterior del fémur y baja hasta el talón llegando al tendón de Aquiles.
- Poplíteo: baja desde el cóndilo externo hasta la tibia por su parte posterior. Su función es la de flexionar la rodilla además de crear una rotación externa.

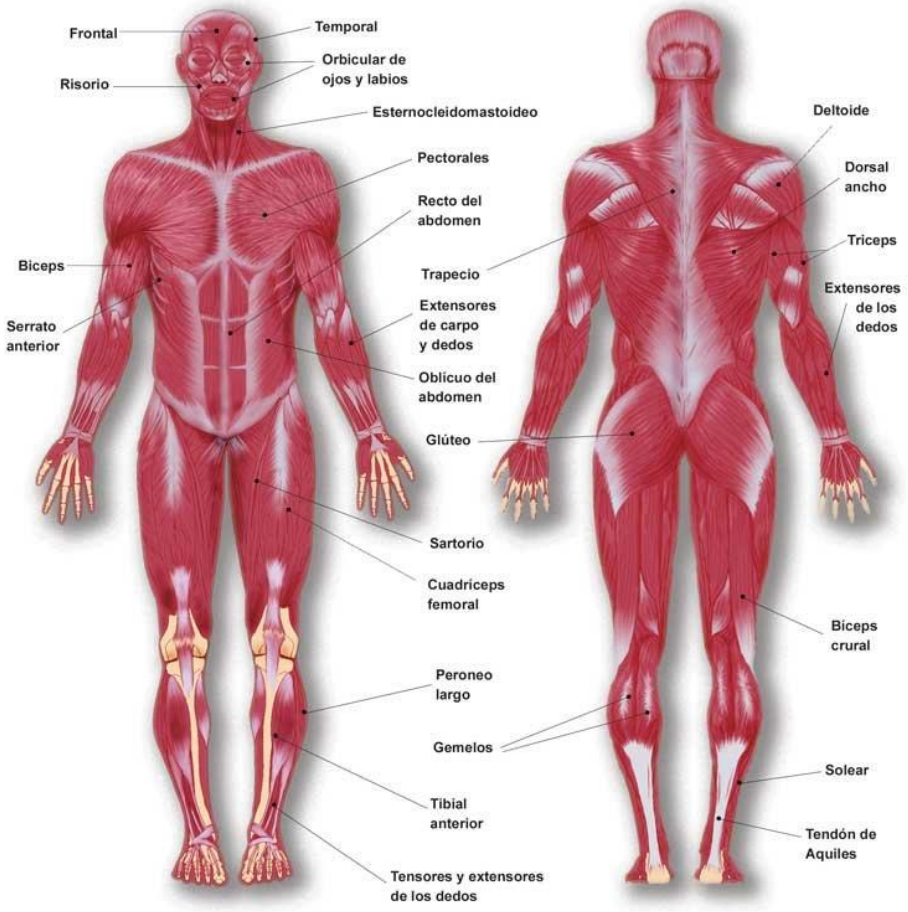

Figura 4: Sistema muscular

2.1.3. Planos y ejes de movimiento del cuerpo humano

Para tener una idea más precisa de los movimientos de las articulaciones, se hace necesario conocer los planos y ejes en los diferentes movimientos que se pueden realizar con el cuerpo humano. Existen diversos términos para describir los tres planos mutuamente perpendiculares en los que la gran mayoría de movimientos de las articulaciones ocurren. Estos sistemas ortogonales pueden ser descritos dependiendo del punto común de intersección de los planos. Este punto de intersección puede ser definido ya sea como el centro de la articulación estudiada o como el centro de masa de todo el cuerpo.

Existen tres planos de movimiento y tres ejes de movimiento en el cuerpo humano.

- **Plano sagital**: es un plano vertical que va de la parte posterior a la parte frontal del cuerpo, dividiéndolo en mitad derecha y mitad izquierda. También es conocido como plano anteroposterior.

- **Plano frontal**: al igual que el plano sagital, es un plano vertical que va de derecha a izquierda dividiendo al cuerpo en dos mitades, anterior y posterior. También es conocido como plano coronal.
- **Plano horizontal**: divide al cuerpo en mitades superior e inferior. Se le conoce como plano transversal.

Los movimientos de las articulaciones músculo-esquelético son en gran medida movimientos rotacionales y tienen lugar sobre una línea perpendicular al plano en el que ocurre dicho movimiento. A esta línea se le conoce como eje de rotación. Existen tres ejes de rotación que pueden definirse por la intersección de los ejes de movimiento antes mencionados:

- **Eje sagital**: pasa horizontalmente desde la mitad posterior a la anterior del cuerpo, formado por la intersección del plano sagital con el plano horizontal.
- **Eje frontal**: pasa horizontalmente de izquierda a derecha y se forma por la intersección de los planos frontal y horizontal.
- **Eje vertical o longitudinal**: pasa verticalmente de la mitad inferior a la superior del cuerpo, siendo formado por la intersección entre los planos sagital y frontal.

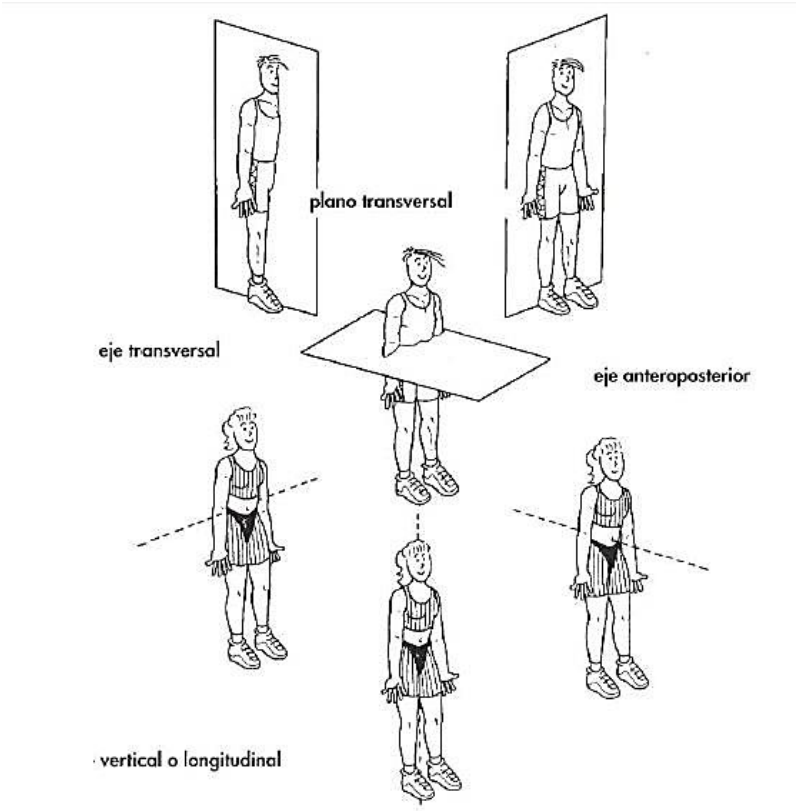

Figura 5: Planos y ejes del movimiento del cuerpo

Ejes de la rodilla

Los movimientos de flexión y extensión de la rodilla se llevan a cabo sobre su eje transversal en el plano sagital. Al mismo tiempo, visto desde el plano frontal, el eje transversal atraviesa los cóndilos femorales horizontalmente. Este eje, al ser horizontal, forma un ángulo de 81˚ con el fémur y de 93˚ con la pierna. Por esta razón, cuando la rodilla se encuentra en flexión completa, el eje de la pierna no se posiciona exactamente detrás del eje del fémur.

2.1.4. Descripción articular

La rodilla es la articulación más grande del esqueleto humano. Constituye una articulación de suma importancia para la marcha y la carrera, que soporta todo el peso del cuerpo en el despegue y la recepción de saltos.

Su mecánica articular resulta muy compleja, pues por un lado ha de poseer una gran estabilidad en extensión completa para soportar el peso corporal sobre un área relativamente pequeña, pero, al mismo tiempo, debe estar dotada de la movilidad necesaria para la marcha y la carrera y para orientar eficazmente al pie en relación con las irregularidades del terreno.

Actualmente, además de los trabajos donde se utilizan técnicas de disección en cadáveres y radiografías, la articulación de la rodilla se investiga a través de resonancias magnéticas nucleares, fotografiada en distintos ángulos durante sus movimientos. Su estudio está condicionado por la alta incidencia de traumatismos y enfermedades osteodegenerativas que la afectan.

La rodilla se clasifica como biaxial y condílea, en la cual una superficie cóncava se desliza sobre otra convexa alrededor de dos ejes. Como superficies articulares presenta cóndilos del fémur, superficie rotuliana del fémur, carilla articular de la rótula y meniscos femorales (estructuras cartilaginosas que actúan como cojinetes, amortiguando el choque entre el fémur y la tibia).

La cápsula articular es grande y laxa, y se une a los meniscos. Por otro lado, conviene destacar que otros anatomistas sostienen que la articulación de la rodilla está compuesta, desde el punto de vista morfológico, por la yuxtaposición de dos articulaciones secundarias: la femororrotuliana (que es troclear) y la femorotibial (que es condílea con meniscos interpuestos). La primera de las cuales constituye una articulación por deslizamiento. Protege por delante el conjunto articular y al mismo tiempo, eleva el cuádriceps, permitiendo que las tracciones de este sobre la tibia tengan lugar con un cierto ángulo de inclinación y no en sentido paralelo, lo que aumenta su poder de tracción.

La articulación femorotibial, por su parte, es dividida por el menisco en dos cámaras: la proximal o superior, que corresponde a la articulación femoromeniscal, responsable de los movimientos de flexión y extensión de la pierna; y la distal o inferior, que corresponde a la articulación meniscotibial y permite los movimientos de rotación de la pierna.

La rodilla humana está construida normalmente con un cierto grado de valguismo lo que significa que estando extendido el miembro inferior, los ejes del fémur y de la tibia no son colineales, sino que forman un ángulo obtuso abierto hacia el exterior (ángulo femorotibial).

Este ángulo de divergencia de los dos huesos que constituyen la articulación mide, como término medio, de 170 a 177˚. Conviene distinguir desde el punto de vista de construcción de la rodilla humana, el eje anatómico o diafisario del fémur (línea que une el centro de la escotadura intercondílea con el vértice del trocánter mayor) del llamado eje mecánico o dinámico de este, que es la línea que une el centro de la cabeza femoral con el centro anatómico de la rodilla y el centro de la articulación tibiotarsiana. Este último eje representa la línea de apoyo o gravedad de toda la extremidad inferior.

En los individuos normales, el eje mecánico o dinámico pasa por el centro de la articulación, o desviado ligeramente hacia dentro (cóndilo interno) o contrariamente, hacia fuera (cóndilo externo). No sucede lo mismo en las desviaciones patológicas conocidas como *genu valgum* y *genu varum*. En estos casos, la línea pasa completamente por fuera (*genu valgum*) o por dentro de la rodilla (*genu varum*).

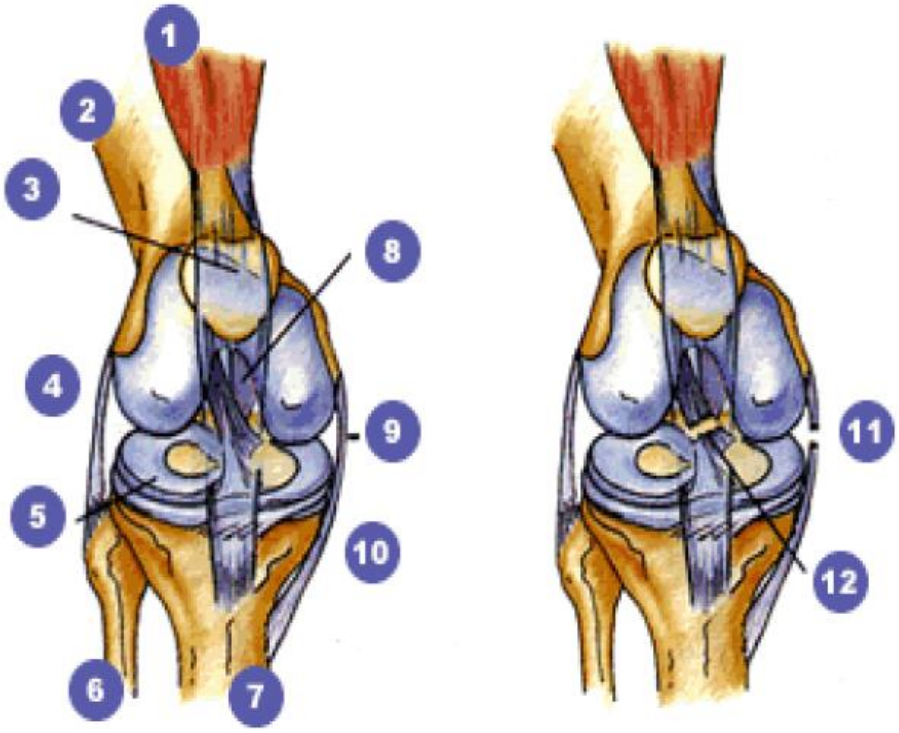

Figura 6: Vista anterior de la articulación de la rodilla

2.1.5. Mecánica articular

La articulación de la rodilla puede permanecer estable cuando es sometida rápidamente a cambios de carga durante la actividad. Este hecho se conoce como estabilidad dinámica de la rodilla y es el resultado de la integración de la geometría articular, restricciones de los tejidos blandos y cargas aplicadas a la articulación a través de la acción muscular y el punto de apoyo que sostiene el peso.

La arquitectura ósea de la rodilla suministra una pequeña estabilidad a la articulación, debido a la incongruencia de los cóndilos tibiales y femorales. Sin embargo, la forma, orientación y propiedades funcionales de los meniscos mejoran la congruencia de la articulación y puede suministrar alguna estabilidad, que es mínima considerando los grandes pesos trasmitidos a través de la articulación.

La orientación y propiedades materiales de los ligamentos, cápsula y tejidos musculotendinosos de la rodilla contribuyen significativamente a su estabilidad. Los ligamentos de la rodilla guían los segmentos esqueléticos adyacentes durante los movimientos articulares y las restricciones primarias para la traslación de la rodilla durante la carga pasiva.

Las restricciones de fibras de cada ligamento varían en dependencia del ángulo de la articulación y el plano en el cual la rodilla es cargada. La estabilidad de la rodilla está asegurada por los ligamentos cruzados anterior y posterior y los colaterales interno (tibial) y externo (peroneo). El ligamento cruzado anterior (LCA) tiene la función de evitar el desplazamiento hacia delante de la tibia respecto al fémur. El cruzado posterior (LCP), evita el desplazamiento hacia detrás de la tibia en relación con el

fémur, que a 90˚ de flexión se orienta verticalmente y se tensa siendo el responsable del deslizamiento hacia atrás de los cóndilos femorales sobre los platillos tibiales en el momento de la flexión, lo cual proporciona estabilidad en los movimientos de extensión y flexión.

Los ligamentos laterales brindan una estabilidad adicional a la rodilla. Así, el colateral externo o peroneo (LLE), situado en el exterior de la rodilla, impide que esta se desvíe hacia dentro, mientras que el colateral interno o tibial (LLI) se sitúa en el interior de la articulación, de forma que impide la desviación hacia afuera, y su estabilidad depende prácticamente de los ligamentos y los músculos asociados.

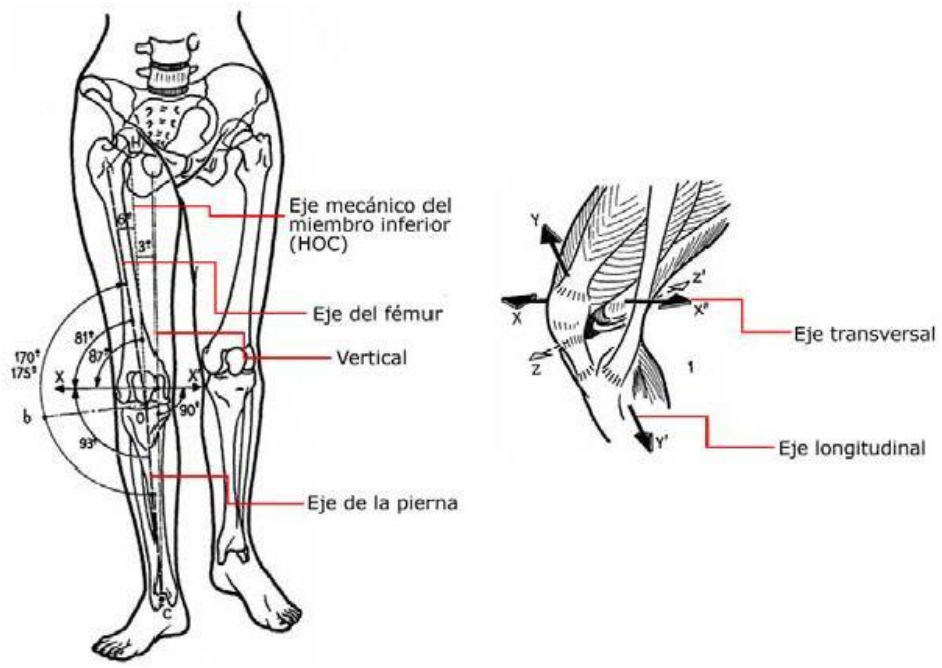

Figura 7: Ejes de movimiento de la rodilla

Consecuentemente, en la mayoría de los casos hay muchos ligamentos que contribuyen sinergísticamente a la estabilidad dinámica de la rodilla. Mientras que los esfuerzos combinados de ligamentos y otros tejidos blandos suministran a la rodilla buena estabilidad en estados de carga con fuerzas moderadas aplicadas a la articulación. La tensión aplicada a estos tejidos durante alguna actividad agresiva (detener o cambiar con rapidez la dirección en ciertos deportes) suele exceder a su fuerza. Por esta razón se requieren fuerzas estabilizadoras adicionales para mantener la rodilla en una posición donde la tensión en los ligamentos permanezca dentro de un rango seguro.

Las fuerzas compresivas de la rodilla, resultantes del soporte del peso del cuerpo y las cargas aplicadas a los segmentos articulares por actividad muscular, suministran estas fuerzas estabilizadoras. La articulación de la rodilla realiza fundamentalmente movimientos en dos planos perpendiculares entre sí: flexoextensión en el plano sagital (eje frontal) y rotación interna y externa en el plano frontal

(eje vertical). Para los movimientos debe tenerse en cuenta que el espesor y volumen de un ligamento son directamente proporcionales a su resistencia e inversamente proporcionales a sus posibilidades de distensión.

Movimientos de flexión y extensión

Se realizan alrededor de un eje frontal, bicondíleo, que pasa los epicóndilos femorales. La cara posterior de la pierna se aproxima a la cara posterior del muslo en el curso de la flexión, pero sucede lo contrario durante el movimiento de extensión. A partir de la posición 0˚ (posición de reposo: cuando el muslo y la pierna se prolongan entre sí colinealmente, formando un ángulo de 180˚), la flexión de la pierna alcanza por término medio 130˚, pero el límite máximo de la amplitud de ese movimiento no es este, pues tomando el pie con una mano puede ampliarse.

La flexoextensión de la rodilla resulta de la suma de dos movimientos parciales que ejecutan los cóndilos femorales: un movimiento de rodadura y un movimiento de deslizamiento de aquellos sobre las cavidades glenoideas. Este último de mayor amplitud que el primero.

El movimiento de rotación o rodadura tiene lugar en la cámara femoromeniscal, mientras que la fase de deslizamiento, en la meniscotibial.

En los movimientos de flexión-extensión, la rótula se desplaza en un plano sagital. A partir de su posición de extensión, retrocede y se desplaza a lo largo de un arco de circunferencia, cuyo centro está situado a nivel de la tuberosidad anterior de la tibia y cuyo radio es igual a la longitud del ligamento rotuliano. Al mismo tiempo, se inclina alrededor de 35˚ sobre sí misma, de tal manera que su cara posterior, que miraba hacia atrás, en la flexión máxima está orientada en un plano situado hacia atrás y hacia abajo. Por tanto, experimenta un movimiento de traslación circunferencial con respecto a la tibia.

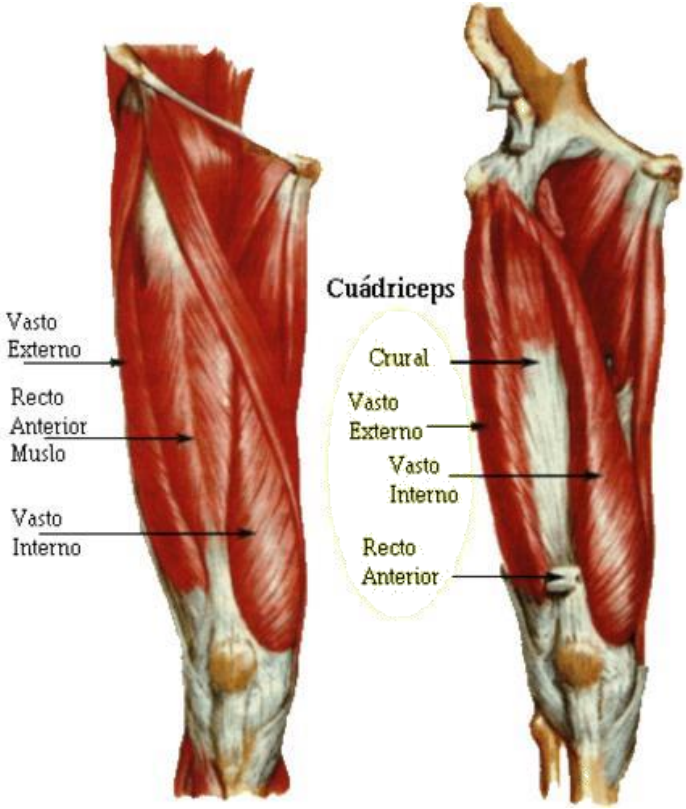

Figura 8: Músculos femorales

Limitantes de la flexión:

- Distensión de los músculos extensores (cuádriceps crural o femoral).
- Por la masa de los músculos flexores en el hueco poplíteo.
- El segmento posterior de los meniscos.

Limitantes de la extensión:

- Distensión de los músculos flexores.
- El segmento anterior de ambos meniscos.
- La distensión de la parte posterior del manguito capsulo-ligamentoso.
- Ambos ligamentos laterales, que al estar situados por detrás del eje de movimientos, se ponen cada vez más tensos a medida que el movimiento de extensión progresa.

En la fase de postura, la flexión de la rodilla funciona como un amortiguador para ayudar en la aceptación del peso. La función de los ligamentos cruzados en la limitación de los movimientos angulares de la rodilla varía, según la opinión de los diferentes autores.

Movimientos de rotación de la rodilla

Consisten en la libre rotación de la pierna, es decir, en que tanto la tibia como el peroné giran alrededor del eje longitudinal o vertical de la primera, en sentido externo o interno. La rodilla puede realizar solamente estos movimientos de rotación cuando se encuentra en posición de semiflexión, pues se producen en la cámara distal de la articulación y consisten en un movimiento rotatorio de las tuberosidades de la tibia, por debajo del conjunto <<meniscos-cóndilos>> femorales.

En la extensión completa de la articulación, los movimientos de rotación no pueden realizarse porque lo impide la gran tensión que adquieren los ligamentos laterales y cruzados. La máxima movilidad rotatoria activa de la pierna se consigue con la rodilla en semiflexión de 90˚. La rotación externa es siempre más amplia que la interna (cuatro veces mayor, aproximadamente).

En la rotación interna, el fémur gira en rotación externa con respecto a la tibia y arrastra la rótula hacia afuera: el ligamento rotuliano se hace oblicuo hacia abajo y hacia dentro. En la rotación externa sucede lo contrario: el fémur lleva la rótula hacia dentro, de manera que el ligamento rotuliano queda oblicuo hacia abajo y hacia fuera, pero más oblicuo hacia fuera que en posición de rotación indiferente. La capacidad de rotación de la articulación de la rodilla confiere a la marcha humana mayor poder de adaptación a las desigualdades del terreno y, por consiguiente, mayor seguridad. Los movimientos de rotación desempeñan también una función importante en la flexión de las rodillas, cuando se pasa de la posición de pie a la de cuclillas. La capacidad de rotación de la rodilla permite otros muchos movimientos, por ejemplo: cambiar la dirección de la marcha, girar sobre sí mismo, trepar por el tronco de un árbol y tomar objetos entre las plantas de los pies. Por último, existe una rotación axial llamada “automática", porque va unida a los movimientos de flexo-extensión de manera involuntaria e inevitable.

Cuando la rodilla se extiende, el pie se mueve en rotación externa. A la inversa, al flexionar la rodilla, la pierna gira en rotación interna. En los movimientos de rotación axial, los desplazamientos de la rótula en relación con la tibia tienen lugar en un plano frontal. En posición de rotación indiferente, la dirección del ligamento rotuliano es ligeramente oblicua hacia abajo y hacia fuera.

Ambos ligamentos cruzados limitan el movimiento de rotación interna, aumentando su cruzamiento, y deshaciéndolo cuando la pierna realiza un movimiento de rotación interna, por lo que no pueden restringir este movimiento de ninguna forma. El movimiento de rotación externa es limitado por el ligamento lateral externo, que se tuerce sobre sí mismo, y por el tono del músculo poplíteo. Al igual que sucede en los movimientos de flexo-extensión, los meniscos también se desplazan en el curso de los movimientos rotatorios de la pierna. En estos desplazamientos reside la causa de su gran vulnerabilidad. Las lesiones meniscales solamente se pueden producir, según esto, en el curso de los movimientos articulares, y no cuando la rodilla se encuentra bloqueada en extensión.

Combinaciones no coordinadas de los movimientos de rotación (sobre todo la interna), que hunden el menisco en el ángulo cóndilo-tibial, punzándole, con los músculos encargados de la flexión y la extensión, son causantes de tales lesiones meniscales.

Se pueden describir otras dos clases de movimientos en la rodilla: movimientos de abducción y aducción. Son más conocidos en semiología con el nombre de movimientos de inclinación lateral y corresponden realmente más a un juego mecánico de conjunto, que a una función que posea una utilidad definida. En la posición de extensión, y fuera de todo proceso patológico, son prácticamente inexistentes. Su amplitud es del orden de 2 a 3˚ y obedecen a una de las características del cartílago articular, que es el de ser compresible y elástico.

Movimientos de la rótula

Generalmente se considera que los movimientos de la rótula no influyen directamente en los de la rodilla. La patela (rótula) sufre un ascenso en la extensión y desciende en la flexión lo que provoca desplazamientos en la articulación femororrotuliana. El movimiento normal de la rótula sobre el fémur durante la flexión es una traslación vertical a lo largo de la garganta de la tróclea y hasta la escotadura intercondílea. El desplazamiento de la rótula equivale al doble de su longitud (8 cm). Lo efectúa mientras gira en torno a un eje transversal. En efecto, su cara posterior, dirigida directamente hacia atrás en posición de extensión, se orienta hacia arriba cuando la rótula, al final de su recorrido, se aplica en la flexión extrema, debajo de los cóndilos, por lo cual se trata de una traslación circunferencial.

Desplazamientos de la rótula sobre la tibia

Es posible imaginarse la rótula incorporada a la tibia para formar un olecranon como en el codo, disposición que al impedir todo movimiento de la rótula en relación con la tibia, limitaría de modo notable su movilidad e inhibiría incluso cualquier movimiento de rotación axial. La rótula efectúa dos clases de movimientos con respecto a la tibia, según se considere la flexión-extensión o la rotación axial.

Las fuerzas que actúan sobre la rodilla durante la marcha son: el peso del cuerpo, consiguiendo el equilibrio de fuerzas con las reacciones impuestas por el suelo y las contracciones de los grupos musculares, que originan un movimiento entre los elementos articulares mediante el desplazamiento de las superficies articulares entre sí, producido por el par de fuerzas generado por el peso del cuerpo y las contracciones musculares. La fuerza resultante que cierra y equilibra al sistema que actúa sobre la articulación, sin producir movimiento, es la fuerza de reacción articular que comprime las superficies articulares entre sí.

Durante las actividades del miembro inferior se generan fuerzas en la rodilla: una de ellas en la articulación femororrotuliana y otra en la femorotibial, que a su vez puede descomponerse en un componente en el compartimento medial y otro en el lateral. Dichas fuerzas son las causantes del daño progresivo de las superficies articulares al ir lesionando la estructura del cartílago con sus componentes de compresión, fundamentalmente, y de cizallamiento. Este último se desprecia en los estudios biomecánicos, por ser prácticamente inexistente, debido al bajo coeficiente de fricción cartílago-cartílago que obedece, por un lado, a las propiedades visco-elásticas de este y, por otro, a la lubricación proporcionada por el líquido sinovial.

La articulación femorotibial (FT) posee un movimiento tridimensional y, por tanto, tres componentes de giro: angulación varovalgo (plano frontal, eje anteroposterior), rotación (plano transversal, eje vertical) y flexoextensión (plano sagital, eje transversal).

También tiene tres componentes de desplazamiento: mediolateral, anteroposterior y compresión-separación, de los cuales solo es trascendente el segundo en un mecanismo combinado con el movimiento de rodadura de los cóndilos femorales sobre la tibia, guiado por el ligamento cruzado posterior, que predomina en los primeros grados de flexión y el desplazamiento al final de esta. El desplazamiento mediolateral resulta mínimo, atribuible a la congruencia articular proporcionada por los meniscos y las partes blandas (ligamentos y contracción muscular). El movimiento de rotación suele ser generalmente automático e involuntario y de un orden de magnitud poco importante (nulo en extensión completa, con máximo de 10 a 90° de flexión). Por tanto, el movimiento principal es el de flexoextensión.

Conviene señalar que el grado de flexión de la rodilla en un ciclo de marcha, varía a lo largo de dicho ciclo, pero nunca logrará estar completamente extendida. Este movimiento de flexoextensión funciona como un helicoide y no como una bisagra simple, pues existe una combinación de flexoextensión con rotaciones, debida a la mayor dimensión próximo-distal del cóndilo medial respecto al lateral. Asimismo, para el movimiento de flexión, el deslizamiento anteroposterior femorotibial aumenta la potencia del aparato extensor hasta en 30 %, al obtener un momento mecánico más favorable. Por el mecanismo de rotación automática descrito anteriormente sucede el fenómeno conocido como autoatornillamiento, que produce el bloqueo femorotibial en extensión completa y aumenta la estabilidad articular, entre otras situaciones, en el instante del apoyo del talón en la marcha. Dicho mecanismo tiene lugar mediante la rotación externa progresiva, con la extensión de la rodilla en fase de balanceo, y provoca el bloqueo progresivo en los últimos 15° de extensión.

El centro instantáneo de rotación de la articulación FT para la flexoextensión se encuentra, en condiciones normales, en el fémur, aproximadamente en la inserción de los ligamentos colaterales en la perpendicular al punto de contacto y va desplazándose dorsalmente con la flexión, en una línea curva suave de concavidad craneal. Este desplazamiento es explicable, entre otros factores, por el deslizamiento femoral sobre la tibia durante la flexión.

A causa de esta variación, los diferentes grupos musculares van variando su momento en un sentido que favorece su funcionalismo.

2.2. Descripción general del sistema

El sistema mecánico se basa en el diseño de una ortesis, es decir, un apoyo o dispositivo externo aplicado a un determinado cuerpo para modificar los aspectos funcionales o estructurales del sistema neuromusculoesquelético. En esta investigación, la ortesis se extiende desde la cadera hasta el pie. En la zona superior, la cadera, no existe parte mecánica, si no que el dispositivo se sujeta a la propia pierna. En cuanto a la parte inferior, el pie, se ha incluido un zapato que permite la incrustación de sensores de presión para la mejora de la captura de movimiento del dispositivo y del control del mismo. El usuario se calzaría la bota sensorizada y se ajustaría la ortesis a la pierna del mismo. Hay que tener en cuenta que la ortesis con la que se trabaja en este TFM está diseñada exclusivamente para el uso de una persona en concreto. La parte mecánica cambiaría para cada usuario, pues es preciso el máximo ajuste para que la marcha sea correcta.

2.3. Diseño mecánico

El diseño mecánico de ortesis activas y exoesqueletos es un campo de investigación que cubre las áreas de ingeniería mecánica, biomecánica, robótica y neurorehabilitación. Estos dispositivos tienen por finalidad asistir la marcha de pacientes que sufren alguna discapacidad motora causada, por ejemplo, por una lesión medular, daño cerebral adquirido, parálisis cerebral o síndrome postpolio. Su uso, además, ayuda a rehabilitar parte de la función motora que estas personas han perdido. Aunque en los últimos años se ha presentado una gran cantidad de diseños para la extremidad inferior (Dollar y Herr, 2008; Pons, 2008), son aún muchos los retos que debe afrontar la comunidad científica para diseñar sistemas que puedan ser utilizados en la vida diaria de los pacientes. Las ortesis se clasifican según las articulaciones que asisten (Hsu et al., 2009). Por ejemplo, existen las llamadas AFO (ankle-foot-orthosis) cuya función es controlar el ángulo de flexión del tobillo.

En algunos casos, como en pacientes con lesión medular o parálisis cerebral, se emplea una AFO para evitar la flexión plantar excesiva, que es una de las causas de la marcha patológica de pie equino (Romkes y Brunner, 2002). Otro tipo de ortesis son las KAFO (knee-ankle-foot-orthosis), que están dirigidas a pacientes con niveles de disfunción en la marcha más severas, incluyendo ausencia total o parcial de control muscular en las extremidades inferiores (Kaufman et al., 1996). Finalmente, también existen ortesis que incluyen la articulación de la cadera (Lewis y Ferris, 2011; Audu et al., 2010). Un caso particular de ortesis es la SCKAFO (stance-control knee-ankle-foot-orthosis), que incorpora un sistema de bloqueo de la flexión de la rodilla para la fase de apoyo (Yakimovich et al., 2009; Yakimovich et al., 2006).

Este dispositivo está indicado para sujetos con debilidad en el músculo cuádriceps, por ejemplo lesionados medulares incompletos con niveles AIS C y D (según clasificación ASIA, American Spinal Injury Association). Yakimovich et al. (2009) presentan diferentes sistemas de bloqueo, comerciales y no comerciales, y discuten sus ventajas y limitaciones. Hay sistemas que se basan en mecanismos de trinquete que bloquean la flexión en un ángulo fijo de la rodilla, otros 4 bloquean la rotación mediante un embrague electromagnético (wrap-spring clutch).

Finalmente, también hay sistemas que bloquean por fricción (leva-anillo o correa-palanca). Las ortesis activas son aquellas equipadas con sistemas motorizados para asistir el movimiento de las articulaciones utilizando una fuente de alimentación externa. Más adelante se habla también de otro tipo de ortesis que se combinan con estimulación eléctrica funcional (FES) de músculos denervados. En la literatura se reflejan diferentes sistemas de actuación. Por ejemplo, Blaya y Herr (2004) presentan una AFO activa destinada al tratamiento de la marcha patológica de pie equino (drop-foot gait). Este prototipo se actúa mediante un motor DC en serie con un elemento lineal elástico (SEA, Series Elastic Actuator) que aplica una fuerza entre el talón y la parte posterior de la pierna para controlar la flexión plantar/dorsal del tobillo. Se utilizan sensores plantares y potenciómetros angulares para el control. Diseños de actuación similares se han utilizado en (Oymagil et al., 2007), también para la articulación del tobillo, y en (Pratt et al., 2004) para la articulación de la rodilla. Quintero et al. (2011) emplean motores DC sin escobillas para controlar el movimiento de cadera y rodilla.

La ortesis presentada se utiliza junto con una AFO pasiva e incluye potenciómetros angulares y acelerómetros situados en el muslo para el control. Kawamoto y Sankai (2002) han desarrollado el “HAL” (Hybrid Assistive Limb), un exoesqueleto diseñado para asistir el movimiento del tren inferior de ancianos, que cuenta con motores DC rotativos para controlar el movimiento de las articulaciones

de cadera y rodilla. Su sistema de control genera pares externos en las articulaciones a partir de mediciones electromiográficas (EMG), de ángulos articulares y de fuerzas de contacto pie-suelo.

En esta línea, Hung et al. (2011) presentan una ortesis con un motor eléctrico y un freno magneto-reológico para actuar y bloquear la rodilla, respectivamente. Otro tipo de actuación ampliamente utilizado son los músculos neumáticos artificiales (músculos McKibben). Sawicki y Ferris (2009) presentan una KAFO destinada a rehabilitación que dispone de seis músculos neumáticos, dos encargados de controlar la flexión del tobillo y cuatro la flexión de la rodilla. Su funcionamiento se asemeja al de los pares agonista-antagonista del sistema músculo-esquelético. Se emplean potenciómetros, sensores plantares y señales EMG para el control. El mismo grupo también ha utilizado este tipo de actuación en una AFO (Kao et al., 2010).

Finalmente, cabe destacar que existen dispositivos comerciales que asisten la marcha de parapléjicos como el "ReWalk" (Argo Medical Technologies, Ltd.), "eLegs" (Ekso & Berkeley Bionics), o "REX" (Rex Bionics), aunque aún son excesivamente caros y muy voluminosos para ser utilizados en la vida diaria. Dos aspectos fundamentales en el diseño de ortesis activas son el confort del usuario y la eficiencia energética del dispositivo.

Silva et al. (2010) presentan un estudio en el que se evalúan los pares articulares y las fuerzas de contacto entre la extremidad inferior y una AFO. Los autores concluyen que los pares articulares en la rodilla y la cadera aumentan por el hecho de llevar la ortesis, y que las fuerzas de contacto ortesis-pierna son sensibles a la posición de los puntos de contacto, a la geometría de las superficies de contacto y a las propiedades de los materiales.

La eficiencia energética es también un aspecto crucial para aumentar la autonomía de estos dispositivos. Para tal fin, algunos autores utilizan elementos elásticos entre partes de la ortesis (Mankala et al., 2010), o en serie con el actuador (Bae et al., 2011; Veneman et al., 2005). Vanderborght et al. (2009) presentan una comparación exhaustiva del consumo energético de diferentes diseños de actuadores, aplicables a dispositivos ortésicos, que incorporan un elemento elástico para mejorar su eficiencia. El diseño de una ortesis activa eficiente, confortable, segura y económica es fundamental para que el usuario acepte la utilización de esta ayuda en su vida diaria. Este es uno de los objetivos que se van a abordar en este proyecto con la finalidad de obtener un dispositivo que tenga potencial para una futura comercialización.

Prototipo de ortesis

La Figura 9, recogida del artículo referenciado en bibliografía [1] de J Cuadrado et al, muestra una representación del prototipo de ortesis desarrollado en el proyecto. Lo distintivo de esta ortesis es, por un lado, su carácter activo debido al motor eléctrico situado en la articulación de la rodilla y, por otro, su gran cantidad de sensores.

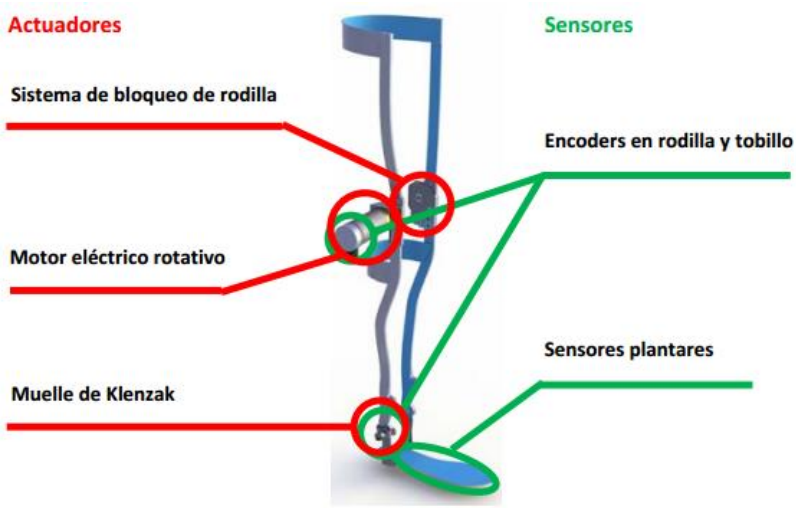

Figura 9: Prototipo de ortesis

2.4. Control de ortesis

Este subapartado se centra en el control de funcionamiento de la ortesis completa, entradas y salidas al control de ambos motores situados a la altura de las rodillas del paciente.

2.4.1. Estimulación eléctrica funcional (FES)

La estimulación eléctrica funcional (EEF o FES, Functional Electrical Stimulation) es ampliamente utilizada desde los años 60 para asistir la bipedestación y la marcha de lesionados medulares. Esta técnica emplea estímulos eléctricos aplicados sobre nervios periféricos para producir la contracción involuntaria de músculos denervados, logrando asistir un movimiento funcionalmente útil.

El empleo de FES presenta ventajas para el paciente respecto a las soluciones puramente robóticas: previene la atrofia muscular, mejora el flujo sanguíneo local, puede reducir la espasticidad, y su tamaño y peso es reducido (Nightingale et al., 2007). Sin embargo, su uso en solitario presenta numerosos inconvenientes: aparición temprana de fatiga muscular, y marcha errática debido a la dificultad para controlar el par a nivel articular (Hausdorff y Durfee, 1991).

Las ortesis híbridas combinan la actuación mediante FES con ortesis pasivas o activas para superar las limitaciones de la actuación mediante FES en solitario. La mayoría de los diseños que se han presentado emplean FES junto con ortesis pasivas que bloquean las articulaciones en ciertas posiciones y reducen el número de grados de libertad para simplificar el problema de control (Jailani et al., 2010).

El ejemplo más representativo es la denominada Reciprocating Gait Orthosis (RGO) (Isakov et al., 1992) que combina una ortesis pasiva que estabiliza la rodilla y el tobillo con la estimulación de forma alternada de los extensores de la cadera para conseguir una marcha pendular. Variaciones de la RGO son la Energy Storage Orthosis (ESO), que emplea FES y almacena el exceso de energía durante la estimulación del cuádriceps empleando muelles neumáticos y la entrega para asistir la extensión de

la cadera más adelante, y la Spring-Brake-Orthosis (SBO) (Gharooni et al., 2001) que combina elementos de bloqueo, almacenamiento de energía y un sistema de FES.

Respecto a ortesis activas híbridas, se han propuesto varios modelos: Popovic et al. (1989) y Ferris et al. (2005) emplean un motor de DC actuando una o varias articulaciones en paralelo con un sistema de FES. Goldfarb et al. (2003) plantearon el uso combinado de un sistema de FES que estimula el cuádriceps durante la fase de balanceo y una ortesis semiactiva que bloquea mediante frenos regulables las articulaciones de rodilla y cadera durante la fase de apoyo.

Kobetic et al. (2009) introdujeron la llamada neuroprótesis híbrida (HNP, hybrid neuroprosthesis), que emplea un sistema de FES implantado quirúrgicamente para estimular ocho grupos musculares de la cadera, la rodilla y el tobillo, combinado con un novedoso control de impedancia en la cadera (Variable Constraint Hip Mechanism, VCHM).

2.4.2. Sensorización y control

Un aspecto trascendental en el diseño de una ortesis activa híbrida es el sistema de sensorización y control que gobierna la actuación coordinada de los actuadores, equipo de FES y elementos de bloqueo. En este sentido, recientes estudios (Cain et al., 2007) han mostrado que el control electromiográfico produce reducciones apreciables de las activaciones musculares en comparación con el control basado en detección de fases de marcha mediante sensores plantares.

Los sistemas de control de ortesis activas propuestos pueden dividirse de forma genérica en las siguientes categorías, atendiendo a su evolución histórica (Jiménez-Fabián y Verlinden, 2011): control manual, control basado en generadores de patrones de marcha, sistemas de control jerárquicos con identificación de la fase de marcha, y sistemas de control con reconocimiento de intención motora.

El control basado en generadores de patrones de marcha asume que el movimiento de marcha es esencialmente periódico. El controlador realiza un seguimiento de una trayectoria preprogramada que puede ser ajustada (Jiménez-Fabián y Verlinden, 2011). Los controles jerárquicos con identificación de la fase de marcha emplean varios controladores en cascada: los del primer nivel identifican la fase actual de marcha mediante señales biomecánicas de la ortesis o del sujeto, y llaman a un controlador de más bajo nivel para cada una de las fases de la marcha (Jiménez-Fabián y Verlinden, 2011).

La detección de intención motora trata de identificar la intención de movimiento del usuario de la ortesis para generar un campo de trayectorias o actuaciones acordes con dicha intención. Existen dos enfoques: el más básico (Marinček y Bajd, 2007), que genera una señal de control proporcional al patrón de activación muscular y, el más avanzado, que emplea modelos musculares EMG-driven, en los que las señales de EMG son convertidas en fuerzas musculares que se emplean como señal de referencia (intención motora) para el control del actuador (Fleischer y Hommel, 2006). Respecto a los sistemas de control de FES, la mayoría de los sistemas de FES para asistencia a la marcha no tienen en cuenta el modelo de músculo denervado y emplean esquemas de control manual operados por el usuario, o bien controles de lazo abierto o realimentado, en los que patrones de estimulación preprogramados son activados en cada fase de la marcha por sensores o señales electromiográficas (Marinček y Bajd, 2007).

Estos sistemas no han resuelto aún el problema de la aparición temprana de fatiga debido al alto coste metabólico de la marcha resultante (Ferguson et al., 1999). Para solventar este problema, se

han propuesto controladores basados en modelos musculares estimulados por FES que adaptan el control teniendo en cuenta el comportamiento del músculo denervado, compensando la estimulación por la aparición de fatiga (Sharma y Stein, 2011).

Por último, el control coordinado de ortesis activas actuadas mediante FES y actuadores convencionales es un campo muy reciente, donde se han aplicado esquemas de control jerárquico FES-ortesis, en los que cada sistema de actuación se controla de forma independiente (Nekoukar y Erfanian, 2010; Schill et al., 2011). Existen muy pocos trabajos donde se aborde la optimización de la actuación combinada de los dos sistemas de actuación para conseguir una marcha natural con bajo coste metabólico. El primer trabajo en esta línea (Sharma y Stein, 2011) calcula las estimulaciones óptimas que minimizan las activaciones musculares simulando la dinámica del sistema FES+ortesis.

Los sensores plantares son simples sensores de presión on/off, que indican si el punto donde se hallan situados está por encima o por debajo de un cierto umbral de presión. Sirven para detectar el comienzo y el final del apoyo del pie en el suelo. Los encoders de tobillo y rodilla miden el ángulo en la articulación respectiva. Los electro-goniómetros (no representados en la Figura 10, recogida de [1], por colocarse sobre el muslo y la pelvis del sujeto, no sobre la ortesis) miden el ángulo en la cadera. La información recogida por todos estos sensores será utilizada por el controlador del motor de la rodilla.

Figura 10: Componentes control ortesis

Para el bloqueo de la rodilla se ha empleado el sistema comercial Neurotronic de la empresa alemana Fior&Gentz, que bloquea o libera la articulación de la rodilla en función del estado de los sensores plantares correspondientes: rodilla bloqueada si el pie está apoyado, y libre si el pie está en el aire. Para evitar que el pie quede colgando y tropiece con el suelo, se ha incorporado un resorte conocido como muelle de Klenzak en la articulación del tobillo.

Y, para ayudar a la pierna en la fase de balanceo, se ha dispuesto un motor eléctrico rotativo de corriente continua con una reductora en serie que permite alcanzar el par necesario para variar el ángulo de la rodilla. Una tarjeta de control recibe la información de los sensores y, en función del

algoritmo de control programado en ella, envía las órdenes de actuación al motor a través del correspondiente servo-amplificador.

El controlador que se ha implementado inicialmente sigue la estrategia siguiente: se mide el ángulo de la cadera y, en función de éste, se calcula el ángulo de rodilla que correspondería a una marcha normal, enviando al motor la señal de que proporcione ese ángulo a la articulación de la rodilla. No obstante, al verificar la utilidad de este esquema de control en las pruebas con lesionados medulares, se ha detectado que se produce un mejor control del sistema si la entrada al mismo es el ángulo del tobillo que se encuentra en contacto con el suelo, y no el de cadera.

En la Figura 11 se puede ver una foto del prototipo construido.

Figura 11: Prototipo

2.5. Simulación

En simulación del movimiento humano hay dos grandes tendencias, dependiendo del campo de aplicación (Xiang et al., 2010b). En robots bípedos, cuando el objetivo es el control en tiempo real, se emplean generalmente modelos simplificados, como el péndulo invertido o el modelo de marcha pasiva, mientras que para la planificación de trayectorias se suele utilizar el método basado en el punto de momento nulo. Sin embargo, en la biomecánica del movimiento humano, es más común el uso de modelos multicuerpo, que replican el aparato locomotor con mucho más detalle que los anteriores. Dentro del enfoque multicuerpo para el estudio de la biomecánica de la marcha, existen dos tipos de problemas.

Por un lado, está el análisis de movimientos reales, cuya cinemática a nivel de posición se captura generalmente de manera óptica mediante sistemas de cámaras sincronizadas, mientras que las fuerzas de contacto pie-suelo se miden con placas dinamométricas. Dicha información se traslada a un modelo multicuerpo que puede ser esquelético o músculo-esquelético. En el primer caso, se asume

que la actuación se concentra en cada articulación como un par articular, de manera que un simple análisis dinámico inverso proporciona las historias de los pares articulares que han producido el movimiento.

En el segundo caso, se detalla la actuación, modelando los músculos que la producen. Se llega entonces a un problema redundante pues, en general, cada grado de libertad articular es actuado por varios músculos. Es el conocido problema de reparto muscular o problema de reclutamiento muscular, y los métodos empleados para resolverlo son los siguientes:

- Optimización estática. Se plantea un problema de optimización en cada instante de tiempo (tras haber realizado una discretización temporal del movimiento), en el que las variables de diseño son las fuerzas musculares, y las condiciones de que las fuerzas musculares de cada articulación produzcan el momento articular resultante del análisis dinámico inverso antes mencionado se introducen como restricciones.
- Optimización estática fisiológica. El método descrito anteriormente no tiene en cuenta la dinámica muscular, pudiendo dar lugar a fuerzas musculares con grandes discontinuidades que no son factibles desde un punto de vista fisiológico. Entonces, la optimización estática fisiológica tiene en cuenta la dinámica muscular, de manera que, para un determinado instante de tiempo, se calcula cuál puede ser la fuerza máxima y mínima que realice cada músculo en el instante siguiente, obligando al problema de optimización de ese nuevo paso a que la fuerza muscular obtenida se encuentre en dicho rango.
- Optimización dinámica. En este caso, el problema de optimización es único para toda la duración del movimiento, siendo las variables de diseño las historias temporales de las excitaciones de los diferentes músculos, mientras que el movimiento conocido se introduce como función de coste junto con algún criterio fisiológico. Las ecuaciones del movimiento, junto con las de activación y contracción muscular son consideradas restricciones. Cada evaluación de función supone un análisis dinámico directo de todo el modelo, por lo que el coste computacional de este método suele ser muy alto.
- Control. Los métodos que introducen control tratan de superar el inconveniente mencionado del gran coste computacional de la optimización dinámica, pero manteniendo la consideración de la dinámica de activación y contracción muscular. Se trata de realizar un único análisis dinámico directo en el que, en cada paso de integración, se realiza una optimización estática y se aplica una técnica de control para seguir el movimiento real.

Los métodos mencionados requieren definir una función de coste para el problema de optimización, para lo que ha habido gran variedad de propuestas, tales como la suma de fuerzas musculares, o de éstas elevadas a una determinada potencia, la máxima fuerza muscular, o criterios análogos pero considerando el valor relativo de cada fuerza respecto al máximo en ese músculo, o las tensiones musculares en lugar de las fuerzas, la fatiga muscular, etc., produciendo cada criterio resultados distintos.

Y, por otro lado, está la simulación de movimientos, esto es, la predicción, en nuestro caso, de cómo va a caminar una persona determinada tras alguna modificación en sus características, que puede consistir, por ejemplo, en una intervención quirúrgica que afecte a algún parámetro biomecánico, en el implante de una prótesis o en la utilización de una ortesis.

La solución de este problema, de gran dificultad, posee ciertamente un enorme interés, y ello motiva que sea un tema de intensa investigación en este momento por parte de la comunidad científico-técnica dedicada al estudio del movimiento humano. En general, los métodos desarrollados hasta la fecha se apoyan en los métodos de análisis antes mencionados, pudiéndose distinguir los siguientes:

- Métodos basados en dinámica inversa (Ren et al., 2007). Se trata de resolver un problema de optimización en el que las historias de las coordenadas que definan el movimiento del sistema sean las variables de diseño. Cada evaluación de función supone la resolución de la dinámica inversa para obtener los pares articulares, por lo que es un método computacionalmente barato.

- Métodos basados en dinámica directa (Anderson y Pandy, 2001). En este caso, las variables de diseño del problema de optimización son las historias de las excitaciones musculares, y la evaluación de función implica resolver la dinámica directa del sistema, lo que implica un alto coste computacional. Es fácil que muchas de las evaluaciones de función den como resultado la caída del modelo, al ser la marcha un movimiento inestable.

- Métodos de dinámica predictiva (Xiang et al., 2010a; Ackermann y van den Bogert, 2010). Estos métodos pretenden recoger los aspectos positivos de los dos anteriores, y plantean un problema de optimización en el que tanto el movimiento como las fuerzas que lo producen son considerados variables de diseño, y las ecuaciones del movimiento son introducidas en forma de restricciones. De esta forma se evita resolver la dinámica directa en cada iteración, reduciéndose drásticamente el coste computacional.

- Métodos que incluyen control (Millard et al, 2009; Rustin et al., 2011). Estos métodos combinan la optimización y el control, de manera similar a como se hacía en el análisis de movimientos reales, para tratar de resolver la dinámica directa del sistema evitando que el modelo se caiga durante la marcha.

La dificultad de obtener resultados en estos métodos es tanto mayor cuanto más se aleje el movimiento solución de un movimiento patrón conocido, por lo que los ejemplos presentados se centran en sujetos sanos. La predicción de movimiento en sujetos con lesión medular utilizando dispositivos ortésicos que se pretende abordar en este proyecto de investigación supone, por tanto, un gran reto en el campo de la dinámica del movimiento humano.

2.6. El sistema de análisis de la marcha

El sistema de análisis de la marcha que se ha desarrollado en este proyecto comienza por un laboratorio de análisis de marcha que consta de varios componentes. En primer lugar, se han instalado doce cámaras de infrarrojos OptiTrack FLEX: V100 de la empresa americana Natural Point, que se han colocado sobre una estructura ligera de barras elevada, de manera que se cubra la zona de captura de la mejor forma posible. Estas cámaras tienen la misión de grabar el movimiento de los marcadores, unas bolitas reflectantes que se colocan sobre el sujeto. Además, se han instalado dos placas de fuerza Accugait, de la empresa americana AMTI, embebidas en una plataforma de madera desmontable, de manera que las placas se puedan situar de distintas formas dependiendo del tipo de marcha a medir (una detrás de otra para medir la marcha de sujetos sanos, una junto a la otra para personas con discapacidad, o incluso otras disposiciones). Las placas sirven para medir las fuerzas de interacción que se producen entre el pie y el suelo.

Sin embargo, los dispositivos anteriores no son suficientes para el análisis de la marcha en el caso de personas que caminen con muletas, ya que las fuerzas de interacción entre éstas y el suelo también deben ser medidas. Por este motivo, se han adquirido un par de muletas convencionales, y se han instrumentado mediante extensometría.

En cada muleta, se pretende medir la fuerza axial y las dos componentes de fuerza cortante que recibe la muleta del suelo. Para ello, se han instalado tres puentes de Wheatstone bajo la empuñadura que miden, respectivamente, la fuerza axial y los momentos flectores en dos direcciones perpendiculares. Además, conviene medir también la fuerza que hace el antebrazo sobre el soporte superior de la muleta, de manera que se puedan determinar los esfuerzos que soporta la muñeca, por lo que se ha instalado un cuarto puente sobre la empuñadura que mide el momento flector correspondiente. Además de la parte experimental descrita, el análisis de marcha requiere de la parte computacional. Ésta comienza por el desarrollo de un modelo matemático del individuo a analizar.

Cuando se analiza la marcha de personas sanas, es común utilizar un modelo plano, ya que la marcha normal tiene lugar principalmente en el plano sagital. Sin embargo, la marcha de los lesionados medulares dista mucho de ser plana, por lo que en este proyecto se ha desarrollado un modelo tridimensional. El modelo consta de 18 sólidos rígidos conectados por 17 pares cinemáticos, todos ellos esféricos, para evitar el problema de la determinación precisa de los ejes en aquellos pares anatómicos asimilables a un par de revolución (como la rodilla o el codo). El modelo precisa de coordenadas que permitan describir el movimiento de esos sólidos rígidos y pares cinemáticos: en este caso se han utilizado 228 coordenadas, correspondientes a las coordenadas cartesianas de 22 puntos, las componentes cartesianas de 36 vectores unitarios, y 54 coordenadas angulares. En ambos casos, personas sanas y pacientes lesionados, pueden apreciarse los marcadores utilizados en la captura del movimiento mediante las cámaras de infrarrojos. Hay que tener en cuenta que, para lograr un buen análisis de la marcha, es necesario que el modelo computacional del individuo sea tan parecido a éste como sea posible, tanto en dimensiones como en propiedades inerciales.

Evidentemente, lo ideal sería disponer de datos precisos, sobre estas magnitudes, obtenidos por tomografía computerizada o técnica similar, pero éste no suele ser el caso. Entonces, en personas sanas, se procede como sigue: para el tren inferior, que es el más relevante en el análisis de la marcha, se recurre a tomar algunas medidas sobre el sujeto, y se aplican después unas ecuaciones de correlación obtenidas sobre muestras representativas de población; para el tren superior, se aplican

unas tablas en función del peso y la estatura del individuo [1] (J. Cuadrado Aranda, U. Lugrís Armesto, F.J. Alonso Sánchez, J.M. Font-Llagunes, Aplicación de técnicas de dinámica multicuerpo al diseño de ortesis activas para ayuda a la marcha).

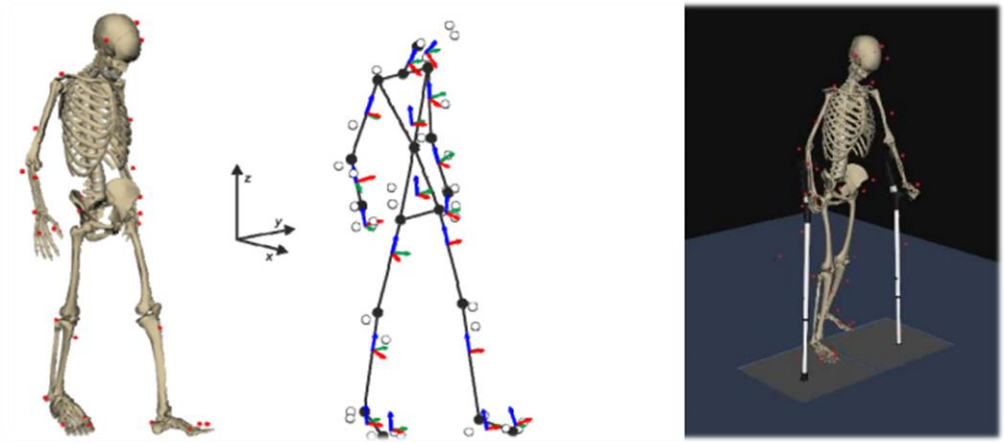

Figura 12: Modelo computacional del cuerpo humano; Modelo de sujeto lesionado con muletas.

El modelo ha de ser animado con el movimiento grabado por las cámaras. Como se ha explicado, lo que graban las cámaras es el movimiento de los marcadores (Figura 12), que van pegados al cuerpo del sujeto con cinta adhesiva de doble cara. Entonces, en el proceso de captura del movimiento se producen errores, debido a la precisión de las cámaras, a la colocación de los marcadores en los puntos anatómicos deseados y, sobre todo, al movimiento de los marcadores con respecto al esqueleto durante el movimiento (fenómeno conocido como "skin motion artifact"). En consecuencia, la información que proporcionan las cámaras sobre la historia de las coordenadas cartesianas de los marcadores debe ser procesada antes de su utilización para animar al modelo. Ello implica su filtrado (eliminación de ruido) y la imposición de la consistencia cinemática (la distancia entre dos puntos de un hueso ha de mantenerse constante durante el movimiento). Además, no sólo es necesario conocer la posición del individuo a lo largo del tiempo, sino también su velocidad y aceleración, lo que conlleva un proceso de derivación numérica que requiere de especial atención para reducir al máximo los errores inherentes a este proceso.

Una vez el modelo ha sido animado con el movimiento procesado, se aplican las ecuaciones del movimiento para obtener los esfuerzos motores que lo han generado (pares articulares y reacciones pie-suelo), esto es, se realiza un *análisis dinámico inverso*. Para marcha normal, dicho análisis sólo requiere conocer el movimiento en el caso de que haya un único pie en contacto con el suelo. Pero necesita más información cuando se produce bipedestación. Es por este motivo que también se miden las fuerzas de contacto pie-suelo, para deshacer la indeterminación que impide conocer cómo se reparten las reacciones entre ambos pies. Además, dicha medición sirve también para valorar la precisión que se está consiguiendo en el análisis de marcha.

En el caso de los lesionados medulares, el modelo ha de ser completado con las muletas y las ortesis. Cada muleta se considera como un sólido adicional que se une rígidamente a la mano del individuo, y no posee ningún tipo de conexión cinemática con el antebrazo. Se añaden al modelo 2 puntos y 2 vectores unitarios por muleta, lo que da un total de 252 coordenadas para el modelo de sujeto con

muletas. En cuanto a las ortesis, las propiedades inerciales se suman a las de los segmentos corporales correspondientes, y los pares en rodilla (motor) y tobillo (muelle de Klenzak) han de ser calculados e introducidos como actuaciones conocidas en el modelo. El cálculo del par del motor implica la programación del algoritmo de control de éste en el modelo computacional, mientras que el cálculo del par del muelle de Klenzak solamente requiere conocer las propiedades del resorte, rigidez y longitud natural.

En el caso de lesionados medulares, es preciso hacer dos consideraciones adicionales. Por un lado, la distribución de masa de estos pacientes suele ser muy diferente a la de las personas sanas, debido a la reducción de masa muscular sufrida en las extremidades afectadas, por lo que es conveniente realizar una densitometría que ofrezca más precisión en los parámetros inerciales que la conseguida por aplicación de las ecuaciones de correlación y las tablas mencionadas más arriba.

Por otro lado, la indeterminación de reacciones exteriores que se produce en el caso de marcha normal cuando ambos pies están en contacto con el suelo, se ve aquí aumentada con los contactos de las muletas con el suelo, pudiendo darse hasta cuatro contactos simultáneos. Por este motivo, se miden experimentalmente las fuerzas de reacción en las muletas. Cada paciente adopta un estilo de marcha muy distinto, por lo que tanto el movimiento como los esfuerzos muestran gran disparidad.

Resulta por ello indispensable analizar ambos aspectos para diseñar un controlador de la ortesis que se adapte a la marcha del paciente concreto, ya que el objetivo de estos dispositivos no es imponer una marcha determinada al sujeto, sino reducir el consumo energético de su marcha, facilitando así que elija caminar en lugar de emplear la silla de ruedas, cuestión clave para su rehabilitación.

2.7. Objetivos del proyecto

Como resultado del proyecto, se dispone de un software que permite la creación de un modelo tridimensional de sujeto sano o lesionado adaptado a las características del individuo concreto (incluyendo muletas para el lesionado), y sobre el que se pueden analizar movimientos reales grabados en el laboratorio montado al efecto (con cámaras, placas de fuerza para el contacto pie-suelo, y muletas instrumentadas con extensometría para medir las fuerzas de contacto con el suelo y con el antebrazo), obteniéndose los correspondientes esfuerzos, tanto a nivel articular como muscular. La discapacidad se ha considerado mediante meros factores que limitan la fuerza en el modelo de músculo de Hill, mientras que la ortesis se ha modelado de manera simplificada, con masas añadidas a los segmentos corporales y pares que actúan en las articulaciones.

Además, se ha diseñado y construido un prototipo de ortesis (derecha e izquierda) para realizar pruebas, en el que se han tenido ya en cuenta algunas de las ideas del Dr. Rodríguez Sotillo, pero no otras, tratando de facilitar la tarea a los ingenieros en este primer intento. Así, su principal propuesta, la actuación de la rodilla durante la fase de balanceo, se ha llevado a cabo con un motor eléctrico con reductora convencional, y no mediante FES como él proponía.

Para el control del motor, en una primera fase experimental, se ha implementado un controlador PI para seguimiento de ángulos de rodilla y cadera, que utiliza encoders en tobillo y rodilla, sensores plantares, y electro-goniómetros para la cadera. Para el bloqueo de la rodilla durante la fase de apoyo, se ha elegido un sistema comercial de la empresa alemana Fior&Gentz. En el proceso de diseño y construcción de la ortesis se ha dispuesto de la estrecha colaboración de Ignacio Prim, director técnico de Establecimientos Ortopédicos Prim.

Como se ha dicho, la idea es llevar a la práctica las ideas del Dr. Rodríguez Sotillo sobre cómo mejorar las ortesis comerciales actualmente existentes para ayudar a la marcha de lesionados medulares incompletos. En general, las ortesis existentes incorporan dos dispositivos básicos:

- El elemento de bloqueo de la rodilla, esencial para evitar que el paciente se caiga en la fase de apoyo del pie al carecer de fuerza suficiente en el cuádriceps, y que debe desactivarse al llegar a la fase de balanceo.
- El elemento de sujeción del pie, llamado antiequino, que evita que éste cuelgue durante la fase de balanceo, golpeando contra el suelo y dificultando el paso. El bloqueo de la rodilla puede ser por forma o por rozamiento. El bloqueo se activa cuando se detecta que el pie toca el suelo, normalmente mediante un sensor de presión on-off colocado en el talón, y se desactiva cuando se detecta la pérdida de presión en el pie con el mismo sensor. La sujeción del pie suele recaer en un resorte, siendo típico el conocido como muelle Klenzak. En definitiva, estas ortesis son esencialmente pasivas, pudiendo considerarse como único elemento semiactivo el bloqueo-desbloqueo y su básico sensor plantar.

Frente a estas características de las ortesis existentes, las ideas de mejora del Dr. Rodríguez Sotillo, en total sintonía con los resultados de recientes estudios, son las siguientes:

- Introducir, además del bloqueo, actuación en la rodilla durante la fase de balanceo para facilitar el paso, ya que muchos de estos pacientes no tienen suficiente fuerza en los flexores de cadera para lanzar el pie hacia delante de manera normal. Existe un gran problema en la flexión de la rodilla, que es el principal inconveniente en el diseño, la paciente que colabora en el proyecto hace flexión en la cadera con sus músculos habiendo ortesis que hacen la extensión de rodilla por inercia, de ahí que se haya tenido que solucionar este problema con cierta prioridad. Su idea primitiva era que esta actuación se realizara por FES, pero, inicialmente, se recurrió a un motor eléctrico por resultar más simple para los ingenieros. Posteriormente se decidió abordar la actuación por FES y la comparación y/o combinación con la actuación convencional.
- Incorporar sensores a la ortesis que permitan un control más fino del bloqueo-desbloqueo de rodilla y, una vez introducida la actuación, también el control de la misma. Los sensores plantares para gobernar el bloqueo-desbloqueo conducen a unos estilos de marcha muy mejorables, lo que se traduce en incomodidad del paciente, aumento de su gasto energético e, incluso, riesgo de caída. En el primer proyecto ya se incorporaron encoders y electro-goniómetros, y en este proyecto se va a investigar la inclusión de EMG.

Objetivos concretos del proyecto global

- Mejorar el diseño mecánico de la ortesis atendiendo a criterios de confort del usuario y consumo energético de la ortesis.
- Diseñar y construir un nuevo prototipo de ortesis activa explorando nuevos sistemas de bloqueo y actuación de la rodilla.
- Incorporar EMG como sensor para el control del bloqueo y actuación de la rodilla.
- Incorporar FES como actuación de la rodilla, sustituyendo en todo o en parte al motor.
- Desarrollar estrategias de control basadas en modelos musculares para el bloqueo y actuación de la rodilla.
- Obtener un modelo muscular de lesionado que mejore el modelo de Hill con factores de discapacidad y considere la actuación híbrida neuronal-FES.
- Introducir en el modelo computacional de sujeto lesionado con ortesis todas las novedades surgidas de los objetivos anteriores.
- Desarrollar nuevos algoritmos de predicción del movimiento de marcha.

3. Simulación movimiento MATLAB: obtención eje rotación rodilla

Para obtener un resultado visual de las capturas de movimiento, se utiliza la herramienta Matlab®. Para ello, se realiza una programación de las diferentes matrices de rotación y transformación necesarias para obtener la transformación de los datos de la captura de movimiento en vectores y puntos que cambian de posición a lo largo del tiempo optimizando los ejes de rotación para obtener la mayor precisión posible en el movimiento.

3.1. Huesos participantes

Los huesos protagonistas del movimiento que interesa en el presente Trabajo Fin de Máster son el fémur y la tibia principalmente. Para ello, se deben tener los datos principales que los sitúan en el espacio, trabajando en coordenadas relativas y absolutas durante las transformaciones, y teniendo en cuenta que el sistema de coordenadas global del esqueleto está situado próximo a la cadera, tomando esta como centro de gravedad del esqueleto completo y dimensionando cada hueso, junto con su escalado y posicionamiento, con respecto a este punto tomado como centro de gravedad absoluto del modelo.

De esta forma, se obtienen los puntos proximal y distal de cada hueso participante en el movimiento, entendiendo como proximal, el más próximo a la cadera. Por tanto, el punto distal del fémur sería el proximal de la tibia, y así sucesivamente.

A partir del archivo que contiene la matriz de datos de la captura del movimiento, se realizan las operaciones precisas para conseguir la simulación del movimiento.

3.2. Programación

Cada uno de los huesos está representado por dos puntos (el proximal y el distal) y tres vectores: u, v y w (Figura 13), que limitan la posición del hueso con respecto al resto del esqueleto y que lo hacen característico en cada uno de los movimientos representados en la matriz de movimiento de marcha recogida en el archivo *qProj.dat*. Los dos primeros son datos recogidos en la matriz de captura del movimiento, mientras que el tercero se calculará a partir de la siguiente expresión, el cociente entre la diferencia de las posiciones distal y proximal del hueso y la norma de ese vector diferencia:

$$\widehat{w} = \frac{r2 - r1}{|r2 - r1|} \qquad (1)$$

El vector w se calculará como el producto vectorial de u y v en caso de que el hueso al que representa no tenga forma alargada y tubular, como puede ser el caso de la pelvis o el pie. Tan solo se utilizará la expresión (1) en caso del fémur y la tibia.

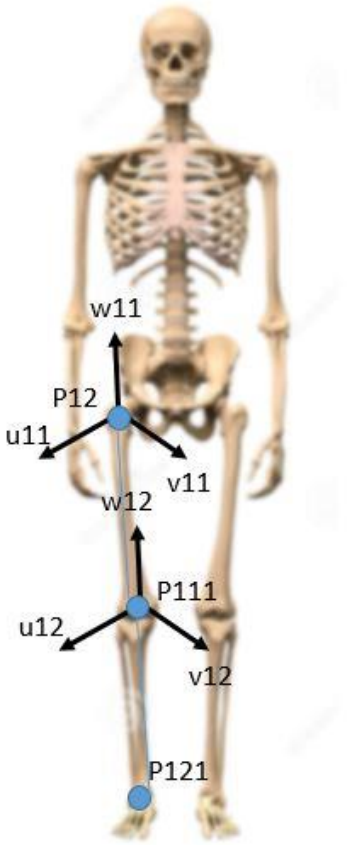

Figura 13: Vectores u, v, w de cada sólido

Se tiene como dato que el fémur es el sólido 11 y la tibia el sólido 12. A partir de aquí, se obtiene la posición de estos sólidos, en posición, y en vectores, en la matriz q de captura de movimiento. Dado que esta matriz contiene un número considerable de filas, se debe realizar un bucle que procese la información de la matriz completa y conseguir así el movimiento deseado en la simulación. Una vez obtenidos los vectores representativos de cada sólido, se puede escribir la matriz de rotación A tanto de la tibia como del fémur, donde A se completa como sigue:

$$A = \begin{bmatrix} ux & vx & wx \\ uy & vy & wy \\ uz & vz & wz \end{bmatrix} \quad (2)$$

Al tratarse de una matriz de rotación, el producto de $A^T A$ debe ser la matriz identidad.

Se realiza el mismo proceso con los datos de captura de movimiento, esta vez, derivados, obteniendo la matriz $\dot{A}$ de cada uno de los sólidos. El producto $\dot{A}A^T$ da como resultado el vector velocidad de rotación de ese sólido en forma de matriz anti-simétrica. Como lo que se desea es obtener el eje de

rotación de la rodilla, para posteriormente poder realizar mejor control de su flexión y extensión, se realiza la diferencia entre los ejes de ambos huesos obteniendo el resultante en la rodilla:

$$\widetilde{W}_R = \widetilde{W}_{12} - \widetilde{W}_{11} \quad (3)$$

Al ser una matriz anti-simétrica, resulta inmediato conocer los componentes del vector:

$$\widetilde{\omega}_R = \begin{bmatrix} 0 & -\omega z & \omega y \\ \omega z & 0 & -\omega x \\ -\omega y & \omega x & 0 \end{bmatrix} \quad (4)$$

Obteniendo además, el vector unitario en coordenadas *globales* del eje como:

$$\hat{e}_R = \frac{\omega_R}{|\omega_R|} \quad (5)$$

Siendo,

$$\omega_R = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix} \quad (6)$$

El eje de rotación global debe ser el mismo para la tibia y el fémur, es decir, está unido rígidamente a los mismos:

$$\hat{e}_R = \hat{e}_{R11} = \hat{e}_{R12} \quad (7)$$

La relación entre el vector en coordenadas locales y globales es,

$$\hat{e}_{R1x} = A_{1x}\bar{e}_{r1x} \quad (8)$$

Siendo, $\bar{e}_{r11}$ el vector en coordenadas locales, calculado a partir de la siguiente expresión:

$$\bar{e}_{r1x} = A_{1x}^{T}\hat{e}_{R} \qquad (9)$$

Se desea optimizar el resultado para evitar la acumulación de errores a lo largo del cálculo. Los vectores del eje en coordenadas locales deberían ser constantes, así que se pueden estimar calculando un promedio. Dicho promedio se pondera con el cuadrado de la velocidad angular, ya que para velocidades pequeñas la ecuación (5) da malos resultados. La fórmula utilizada es la siguiente:

$$\bar{e}_{r1x}{}^{op} = \frac{\sum_{i=1}^{n} \bar{e}_{r1x} \, |\omega_R^i|^2}{\sum_{i=1}^{n} |\omega_R^i|^2} \qquad para\ x = 1{,}2 \qquad (10)$$

Los nuevos vectores de rotación, serán el siguiente producto:

$$\hat{e}_{R1x}{}^{new} = A_{1x}\bar{e}_{r1x}{}^{Op} \qquad (11)$$

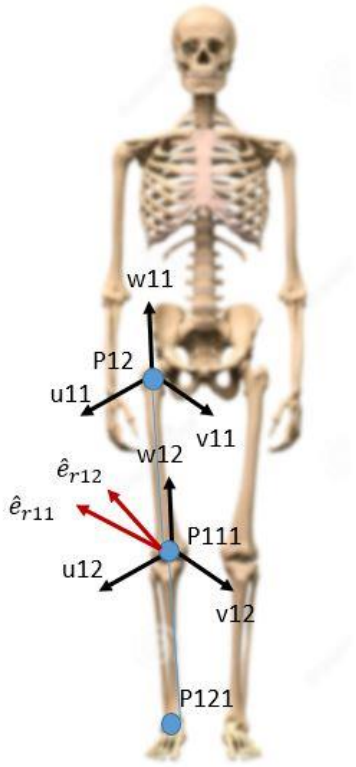

Figura 14: Ejes rotación en rodilla

Una vez realizados los cálculos para la determinación del eje de giro de la rodilla en coordenadas globales, se procede al cálculo de la matriz de transformación que representa la tibia en coordenadas locales del fémur. Para ello, se utiliza la **Fórmula de Rodríguez**, utilizando el vector que corresponde a la rodilla en coordenadas locales del fémur y además, el ángulo que describirá la rodilla a lo largo del movimiento de marcha capturado en la matriz de captura de movimiento.

Primeramente, se halla el ángulo que describe aproximadamente la rodilla durante la marcha de la siguiente forma:

$$\theta r = \tan^{-1}(norm(\overrightarrow{u1} x \overrightarrow{u2})/(u1^T.u2)) \quad (12)$$

$$\theta tob = \tan^{-1}(norm(\overrightarrow{u2} x \overrightarrow{u3})/(u3^T.u2) \quad (13)$$

Por tanto, se tienen los ángulos descritos por la rodilla y por el tobillo, que posteriormente se utilizarán para representar gráficamente (Figura 15) y observar su evolución (ángulos en grados frente a frames).

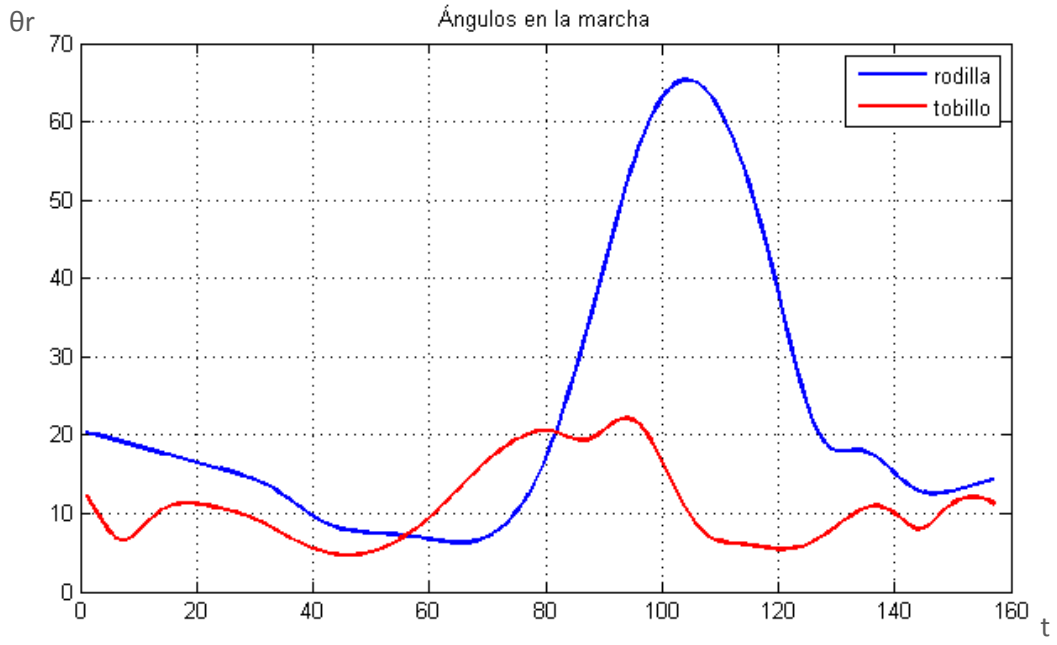

Figura 15: Rodilla derecha/Tobillo derecho

Verdaderamente, resulta más interesante la comparación de la evolución de los ángulos que describen la rodilla de una pierna frente al tobillo de la otra pierna, para así poder comprobar que la evolución de cada uno, concuerda con lo esperado de la marcha humana correcta.

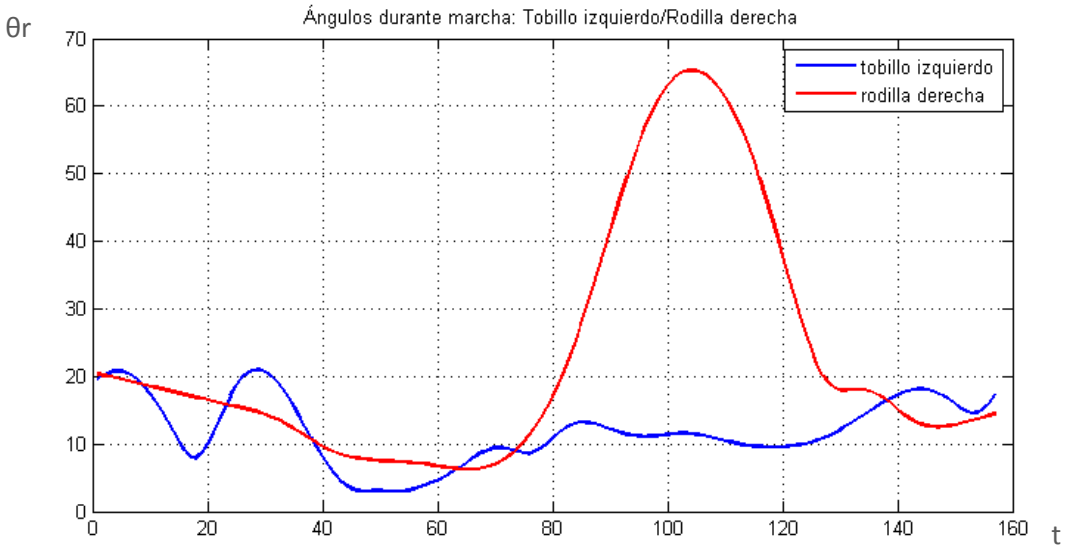

Figura 16: Rodilla derecha/ Tobillo izquierdo

Para conocer la forma que debe describir el control de la rodilla, se enfrenta el ángulo descrito por esta articulación, frente al ángulo que describe a lo largo de la marcha el tobillo de la pierna contraria. Así se podrá comprobar (Figura 17) a qué tipo de curva se debe ajustar la optimización del ángulo descrito.

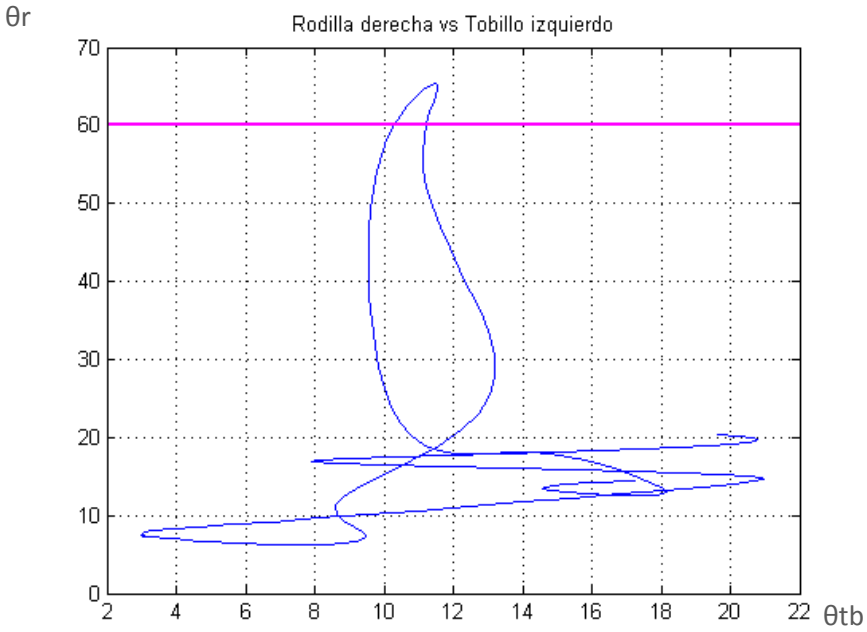

Figura 17: Representación enfrentada de rodilla y tobillo

Por último, para clarificar la interpretación de gráficas que enfrentan ángulos, se expone la representación de los ángulos que describen la rodilla y el tobillo en la misma pierna. Así se podrá interpretar de forma ligeramente más sencilla cómo evolucionan los ángulos que describen ambas articulaciones a lo largo del movimiento de marcha.

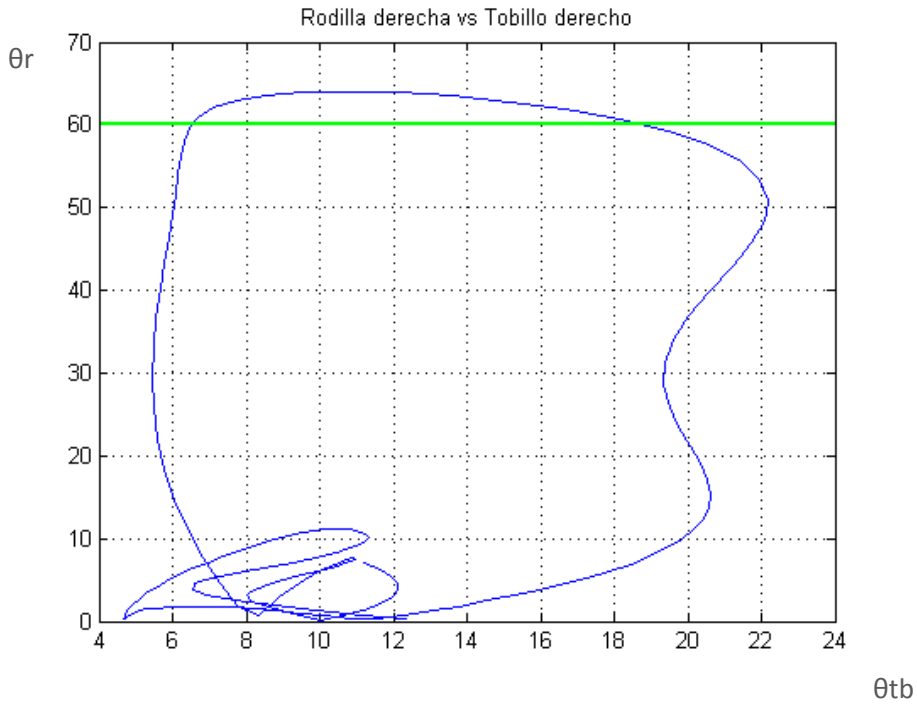

Figura 18: Representación de ángulos descritos por rodilla y tobillo derechos

Siempre se debe tener en cuenta la fuerza de ambos pies durante la marcha, como se puede observar en la Figura 19. Esto es, saber cuándo existe doble apoyo, y cuando uno de ellos está ejerciendo más fuerza, lo que implicaría directamente cuál es aquel que está sobre el suelo y cuál en el aire.

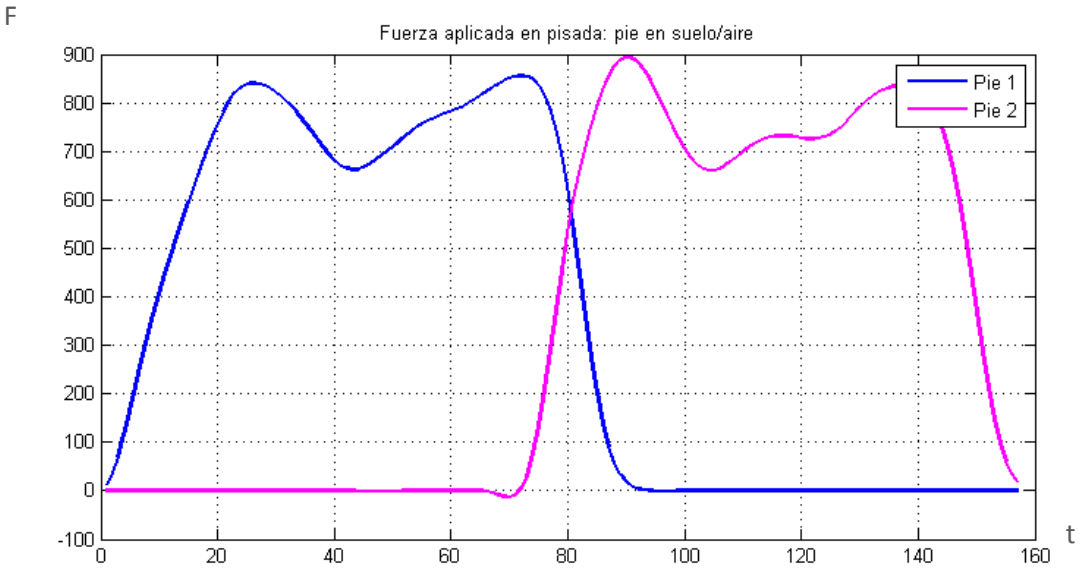

Figura 19: Fuerza aplicada en pisada

Utilizando la *fórmula de Rodríguez*, se puede hallar la matriz de transformación relativa dependiente del ángulo que describe la rodilla, y posteriormente, calcular la matriz de transformación de la tibia en coordenadas locales del fémur.

$$A2 = A1.Ar \tag{14}$$

$$[u2\ v2\ w2] = [u1\ v1\ w1][ur\ vr\ wr] \tag{15}$$

$$Ar = \left[I_3 + \widetilde{Vr}\ sin\theta r + 2\widetilde{Vr}\widetilde{Vr}\sin^2\frac{\theta}{2}\right] \tag{16}$$

Donde $\widetilde{Vr}$ será la matriz anti-simétrica correspondiente al vector $\overrightarrow{erLOptimo1}$ calculado previamente como eje de giro óptimo en coordenadas locales del fémur.

$$\widetilde{Vr} = \begin{bmatrix} 0 & erLOp1(3) & -erLOp1(2) \\ -erLOp1(3) & 0 & erLOp1(1) \\ erLOp1(2) & -erLOp1(1) & 0 \end{bmatrix} \tag{17}$$

Esta nueva matriz de transformación, vista desde el fémur, debe ser comparada con la matriz de transformación calculada previamente en coordenadas globales. En la animación 3D debe tener un aspecto parecido, si bien no idéntico puesto que los parámetros difieren en mayor o menor medida.

Llegados a este punto, se tiene un sistema que no mantiene el ángulo inicial que se desearía tener entre el fémur y la tibia, tanto inicialmente como en el transcurso de la marcha. Por ello, se decide aplicar una nueva trasformación algebraica que relacione ambos huesos mediante una proyección de los vectores representativos de cada uno de los sólidos con respecto al eje de giro de la rodilla. Así, ambos estarían proyectados sobre el mismo plano consiguiendo un movimiento más preciso debido a la falta de exactitud en los ángulos relativos entre los vectores representativos de cada hueso. De esta manera se consigue el movimiento deseado en la simulación, obteniendo una representación del ángulo durante la marcha muy real.

Se parte de que la matriz de transformación relativa será el producto de una constante por la matriz de transformación que se obtiene del ángulo de giro de la rodilla. Esta matriz constante, se obtiene de la resolución de un sistema de ecuaciones en el que se tiene como datos los vectores *u* del fémur y de la tibia y cuyas incógnitas serán las componentes precisamente de esta matriz. Se especifica mediante la consecución de ecuaciones como sigue:

$$\overrightarrow{u1} = A(\theta)A(0)\overrightarrow{u2} \quad (18)$$

$$A(\theta)^T\ \overrightarrow{u1} = A(0)\overrightarrow{u2} \quad (19)$$

Se resuelve a continuación un sistema de 10 ecuaciones y 9 incógnitas en el que se añade la décima ecuación para asegurar un término en la matriz de transformación resultado de resolver el sistema.

```
y(1) = x(1)*u2(1)+x(4)*u2(2)+x(7)*u2(3)-u1(1);
```
(20)

```
y(2) = x(2)*u2(1)+x(5)*u2(2)+x(8)*u2(3)-u1(2);
```
(21)

```
y(3) = x(3)*u2(1)+x(6)*u2(2)+x(9)*u2(3)-u1(3);
```
(22)

```
y(4) = x(1)^2+x(2)^2+x(3)^2-1;
```
(23)

```
y(5) = x(4)^2+x(5)^2+x(6)^2-1;
```
(24)

```
y(6) = x(7)^2+x(8)^2+x(9)^2-1;
```
(25)

```
y(7) = x(1)*x(4)+x(2)*x(5)+x(3)*x(6);
```
(26)

```
y(8) = x(1)*x(7)+x(2)*x(8)+x(3)*x(9);
```
(27)

```
y(9) = x(4)*x(7)+x(5)*x(8)+x(6)*x(9);
```
(28)

```
y(10) = x(3);
```
(29)

Resolviendo este sistema (tomando como condición inicial la matriz identidad 3x3), se obtiene la solución de la matriz que se deseaba en un principio obtener.

Este sistema de ecuaciones parte de busca solución a una matriz A compuesta de los términos en x de las ecuaciones anteriores:

$$A2 = A1.A(\theta).A0 \quad (30)$$

$$A = \begin{bmatrix} x1 & x4 & x7 \\ x2 & x5 & x8 \\ x3 & x6 & x9 \end{bmatrix} \quad (31)$$

$$A0 = \begin{bmatrix} u0x \\ u0y \\ u0z \\ v0x \\ v0y \\ v0z \\ \vdots \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (32)$$

Con esta matriz, se resuelve de nuevo la ecuación que da como solución el ángulo de rotación de la rodilla pero con los vectores u tanto del fémur como de la rodilla, proyectados sobre el eje de giro, mediante la ecuación:

$$\hat{e} = A1 * \overline{er} \quad (33)$$

$$u11p = u11 - (u11^T . \hat{e}). \hat{e} \quad (34)$$

Y de igual modo se calcula la proyección del vector u representativo de la rodilla u22. Finalmente, el ángulo que representa la rodilla será:

$$\theta r = \tan^{-1}(norm(\overrightarrow{u11p} x \overrightarrow{u22p})/(u11p^T . u22p)) \quad (35)$$

Este código de programación se transforma en lenguaje c++ para poder realizar la simulación del movimiento en Open Scene Graph® para poder visualizar los huesos participantes en el movimiento, lo que permite facilitar al observador el movimiento de marcha.

Se utiliza programación enfocada a Open Scene Graph®, en la que se cambia la opacidad de la extremidad inferior derecha sana, para poder comparar el movimiento durante la marcha entre una persona sana y el paciente del que se ha capturado el movimiento.

En c++ no es trivial el manejo de matrices, por lo que se ha tenido que trabajar con vectores de tamaño 3x1, 9x1 o 16x1 dependiendo de si se trabajan con vectores, matrices de 3x3 o matrices de 4x4. Para ello se debe tener en cuenta que el orden matricial es el siguiente:

$$\begin{bmatrix} ux & vx & wx \\ uy & vy & wy \\ uz & vz & wz \end{bmatrix} \gg \begin{bmatrix} ux \\ uy \\ uz \\ vx \\ vy \\ vz \\ wx \\ wy \\ wz \end{bmatrix}$$

$$\begin{bmatrix} ux & vx & wx & rpx \\ uy & vy & wy & rpy \\ uz & vz & wz & rpz \\ 0 & 0 & 0 & 1 \end{bmatrix} \gg \begin{bmatrix} ux \\ uy \\ uz \\ 0 \\ vx \\ vy \\ vz \\ 0 \\ wx \\ wy \\ wz \\ 0 \\ rpx \\ rpy \\ rpz \\ 1 \end{bmatrix}$$

Se debe tener especial cuidado durante la programación del código. El orden de los componentes de las matrices es extremadamente importante para que los resultados sean adecuados. Puede que la simulación no dé errores en cuanto a compilación del archivo, pero el resultado sería totalmente erróneo pudiendo ocasionar movimientos muy lejanos a la realidad e incitando a un control de la ortesis más severo.

Teniendo la simulación realizada, se deberá implementar el control de la ortesis para que el ángulo que se introduce en la matriz de transformación que genera el movimiento de la tibia en coordenadas globales con respecto al fémur nos dé como resultado el ángulo que se debe introducir al control para el movimiento de la pierna contraria. Este ángulo que se introducirá ahora, será el resultado del control, puesto que se necesita una continua retroalimentación con el movimiento y el ángulo que había anteriormente, en el instante inmediatamente anterior.

A la vez, será necesario conocer el ángulo del tobillo en cada momento. Esto permitirá conocer la posición de la pierna y a la vez, permitirá saber si el paciente desea comenzar la marcha o se encuentra en posición de reposo. Además, habrá que distinguir entre intención de comienzo de marcha o un simple movimiento, de giro sobre sí mismo o de cambio de rumbo durante la marcha. En estos momentos el ángulo del tobillo puede resultar equívoco para el control por software.

Será necesario conocer esas posibles adversidades del control y encontrar soluciones que permitan discernir entre unas situaciones u otras.

4. Mejora del controlador

Este apartado contiene el grueso del presente Trabajo Fin de Máster. En él se detallarán todos los aspectos referidos a la programación y utilización de diferentes modos de sensorización para calibrar el control de la ortesis. Esto es, las entradas que se deben introducir al control para que la otra pierna actúe en consecuencia.

4.1. Inputs al control

Se trata de aquellas entradas que debemos introducir al control para que a la salida del mismo se consiga el resultado esperado. Como se debe tener en cuenta la posición y movimiento que realiza o en la que está una pierna para actuar de cierta manera sobre la pierna contraria, se deben introducir cuidadosamente los inputs de tal manera que se pueda equilibrar el funcionamiento de ambas ortesis simulando la marcha habitual de un sujeto sano.

Los inputs al control serán los ángulos que describen los tobillos. El ángulo del tobillo que pertenece a la pierna que inicia el movimiento de *swing* será el input en ese momento en el control creado. Con esta entrada, se calculará el ángulo de la rodilla perteneciente a la pierna contraria para inferirle el movimiento correcto para la marcha adecuada de la persona. Además, se realizará un filtrado de la velocidad (derivada de los inputs) para conseguir evitar el ruido inherente a la toma de datos (adquisición) y que así no existan desviaciones ni mala praxis en cuanto al cálculo del ángulo adecuado impidiendo que aparezcan dos posibilidades de ángulo como input del control y por tanto se cree incertidumbre en cuanto al movimiento que debe realizar la rodilla de la pierna contraria.

El control se realizará basándose en el ángulo descrito por el tobillo de la pierna contraria, y además, jugarán un papel fundamental el estado (alto o bajo) de las placas de fuerza donde se ha realizado la captura de movimiento. Si la placa de fuerza de la pierna contraria está HIGH, y la de la pierna a la que se le va a aplicar el control está LOW, entonces es el momento justo para aplicar el control basado en el ángulo del tobillo de la pierna contraria.

4.2. Introducción de nuevos sensores

En este apartado se muestran los sensores añadidos al conjunto ortésico para poder realizar el control sobre el motor de la ortesis en lugar de la utilización de los encoders instalados previamente ni de la posterior utilización de potenciómetros.

4.2.1. IMUs

Un IMU (del inglés Inertial Measurement Unit) o unidad de medición inercial, es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un dispositivo, usando una combinación de acelerómetros y giróscopos. Las unidades de medición inercial son usadas para pilotaje de aviones, entre muchos otros usos. Recientes desarrollos han permitido la producción de dispositivos GPS protegidos contra la interferencia electromagnética.

Los datos recolectados por los sensores de una IMU permiten a un computador seguir la posición del dispositivo, usando un método conocido como navegación por estima. Una IMU trabaja detectando

la aceleración instantánea del dispositivo utilizando uno o más acelerómetros, y detectando a su vez, cambios en atributos rotacionales (cabeceo, alabeo y guiñada) mediante el uso de uno o más giróscopos.

Los IMUs, tal como se ha indicado, se utilizan ampliamente para propósitos de navegación y calibración de vehículos autónomos. Se procede a continuación a realizar una comparativa y evaluación de IMUs low-cost centrándose en los dos posibles empaquetados de los sensores y las soluciones software disponibles. La comparativa de IMUs suele basarse en la utilización de navegación inercial, o bien cuando se les añade funcionalidades de GPS. La dirección futura de los IMUs de bajo coste se encuentra en la inclusión de una solución que se encauce una estrecha colaboración entre los IMUs y el flujo óptico, utilizando igualmente la navegación inercial propia de los IMUs e introduciendo, según coste del dispositivo, funcionalidad de GPS, magnetómetro y giróscopo.

Introducción

Una IMU es un dispositivo para medir estados relativos de una unidad estática o móvil con respecto a una referencia inercial. Recientemente, numerosos sistemas microelectrónicos (MEMS) han salido al mercado por un precio que asciende a cientos de dólares. Estos IMUs low-cost pueden ser utilizados en vehículos de navegación o bien pueden combinarse con sensores para trabajos de georeferencia.

La precisión de una IMU resulta una parte importante del coste total del dispositivo en el que se vaya a utilizar. De ahí que haya comenzado a emerger una demanda de IMUs low-cost que hace posible el uso en muchas aplicaciones de control en tiempo real, que si no necesitan una gran precisión de trabajo, resultan de gran utilidad. Con la nueva tendencia hacia la modularización y estandarización en el diseño de sistemas de control, los investigadores pueden utilizar IMUs de bajo coste como parte del sistema de navegación o implementar su propio sistema con sensores inerciales de bajo coste.

Se procede a realizar una comparativa en IMUs de bajo coste, sin embargo no se debe perder de vista que este tipo de dispositivos, aunque son muy usados en dispositivos pequeños o micro dispositivos, y por tanto son pequeños, ligeros y aun así bastante potentes, poseen grandes errores de medida o ruido en comparación con sistemas de navegación de mayor coste. Resulta interesante para el diseño y el testeo de dispositivos, la implementación de IMUs para sistemas de navegación basados en sensores inerciales de bajo coste.

Los IMUs se utilizan normalmente para medir estados de dispositivos como orientación, velocidad y posición. La medida de la orientación es especialmente importante para misiones que requieren una navegación precisa. Sin embargo, la orientación no es directamente medible con sensores de corriente. Debe ser estimada a partir de una serie de estados correlativos con medidas angulares diferentes (giros), aceleraciones lineares (acelerómetros) y campos magnéticos (magnetómetros).

Por consiguiente, la estimación de la precisión de una IMU está altamente relacionada con el sensor utilizado y por tanto, con su algoritmo de optimización. Muchos investigadores han preferido adentrarse en el problema de estimación de estados utilizando filtros no lineares.

Existen diferentes filtros Kalman que son muy utilizados para este tipo de estimaciones. Sin embargo, muchos de estos algoritmos son desarrollados para sensores inerciales de alta precisión. Además, pueden tener un coste computacional elevado, incompatible con la idea de IMUs de bajo coste.

Si se realiza una pequeña búsqueda de posibles filtros de corriente para este tipo de sensores de bajo coste, se encontrará que existen filtros complementarios, filtros Kalman y otros filtros no lineares.

Existe un problema para establecer una relación entre los diferentes sensores que componen una IMU. Este problema se conoce como problema de filtrado no lineal o de estimación de estados que se resuelve normalmente con un filtro de Kalman o complementarios.

Sensores incluidos en la IMU

El desarrollo de sensores inerciales MEMS de bajo coste puede postularse hacia los años 70. Una IMU general se compone de los siguientes sensores:

- Giróscopo: un giróscopo es un sensor que se utiliza para medir cambios angulares en torno a unos ejes predefinidos observando siempre desde el sistema de coordenadas terrestres del cuerpo. El error de un giróscopo puede ser expresado mediante:

$$\widehat{w} = (1 + sg) * w + bg + \mu g \qquad (36)$$

Donde $\widehat{w}$ es el valor medido, sg es el error escalado, w es el valor verdadero, bg es la tendencia del giróscopo y finalmente μg es el ruido aleatorio.

Este tipo de sensores puede ser integrado para conseguir estimar el ángulo. Sin embargo, la estimación del ángulo basada únicamente en giróscopos tiene grandes derivas.

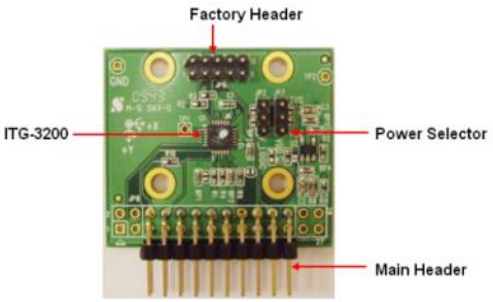

Figura 20: Giróscopo

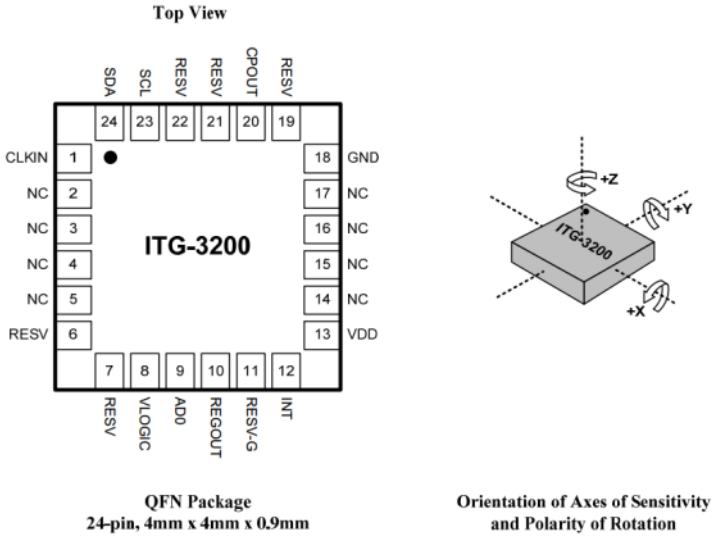

Figura 21: ITG 3200

- Acelerómetros: utilizados en IMUs de bajo coste para medir aceleraciones lineales. De hecho, los acelerómetros miden la aceleración bajo la acción del vector de gravedad. Por ejemplo, la salida por defecto de un acelerómetro es -1 cuando el eje apunta hacia el centro de la tierra.

$$\hat{a} = (1 + sa) * \mathrm{a} + ba + \mu a \qquad (37)$$

Donde $\hat{a}$ es el valor medido, sa es el error escalado, a es el valor real, ba es la tendencia del acelerómetro y μa es el ruido inducido. El acelerómetro puede ser utilizado para medir la tendencia del vehículo en relación a un acelerómetro de tres ejes que pueda medir el vector de gravedad bajo aceleración nula. Sin embargo, el ángulo estimado por un acelerómetro sufre ruido de alta frecuencia cuando comienza el movimiento de los vehículos sin navegación inercial.

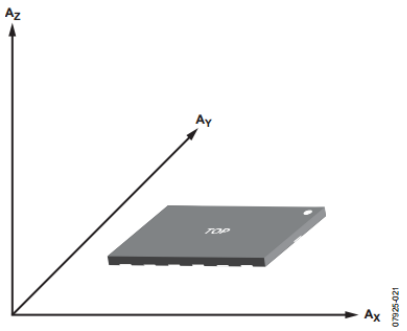

Figura 22: Acelerómetro. Ejes

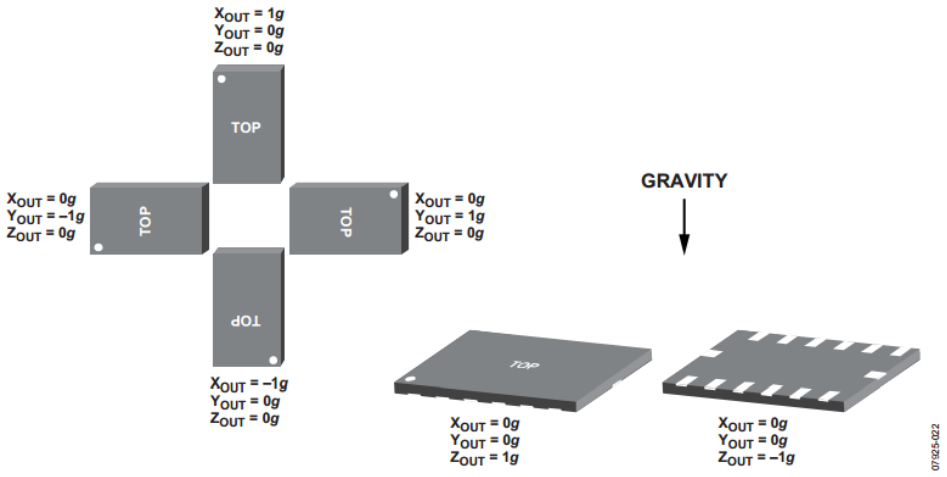

Figura 23: Salida frente a orientación

- Magnetómetro: los magnetómetros son utilizados para medir campos magnéticos de la Tierra, que se pueden simplificar con un valor absoluto asumiendo que el vehículo no se está moviendo a alta velocidad. Los de tres ejes son utilizados para estimar el cabeceo y la tendencia del giróscopo a la compensación del movimiento.

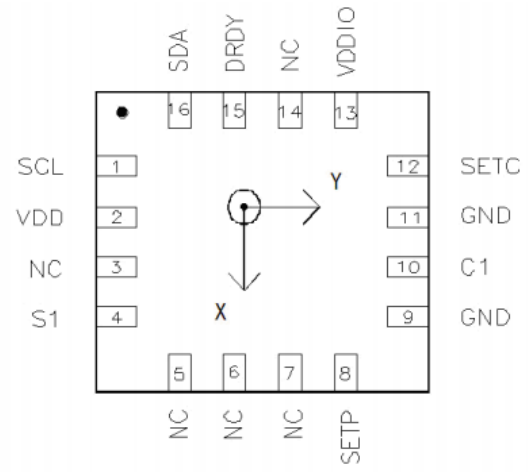

Figura 24: Distribución patillas magnetómetro

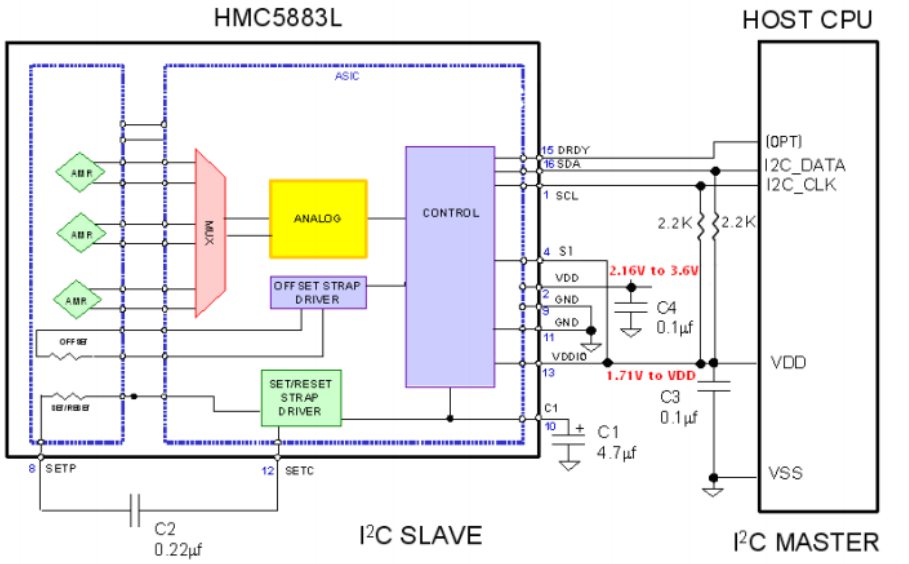

Figura 25: Esquema interno

- GPS: los sensores GPS pueden aportar medidas de posición absoluta, velocidad y trayectoria del ángulo. La posición de los paquetes pueden ser tanto de latitud, longitud como altura, o bien x, y, z expresados en referencia ECEF. Las velocidades que se incluyen (Vn, Ve, Vd) están todas expresadas con respecto a un sistema de referencia inercial. El cambio de ángulo será definido como relativo con respecto al norte (sistema girando en sentido de las agujas del reloj). Las medidas del GPS tienen ciertas ventajas en cuanto a limitación de errores, que puede ser usado para resetear el error del sistema.

- Sensores de presión: estos sensores incluyen sensores de presión relativa y absoluta. El principal se usa para medir la presión del aire y estimar la altitud. El último de ellos, se utilizará para medir la velocidad del aire. Resulta interesante que la mayoría de IMUs industriales usados en sensores MEMS inerciales reducen el tamaño y peso de estos dispositivos.

Figura 26: Ejemplo sensor de presión

Comportamiento de los algoritmos de estimación

Con todos los sensores expuestos anteriormente, se hace necesario un algoritmo que fusione las funcionalidades de cada uno eficientemente para así conseguir una estimación óptima del comportamiento del dispositivo en el que vaya a ser integrado. Los filtros de Kalman son muy utilizados en problemas de estimación no lineales, especialmente en cuerpos rígidos. Este filtro puede estimar los diferentes estados del sistema recursivamente. Posee ciertas ventajas en cuanto a la utilización del giróscopo y el acelerómetro dado que poseen derivas de ruido Gaussiano.

- Filtro Kalman extendido: asume que un sistema discreto no lineal puede ser modelado como:

$$\mathrm{xk} = \mathrm{f}(\mathrm{xk}-1, \mathrm{uk}) + wk, s.t. w \sim N(0, Q) \quad (38)$$

$$yk = g(xk) + vk, s.t. v\tilde{\mathrm{N}}(0, R) \quad (39)$$

Donde x_k es un vector mx1, y_k es un vector nx1, f (xk-1, uk) y g (x_k) son funciones no lineares. La primera ecuación es llamada ecuación de propagación y la segunda es conocida como la ecuación de medida. Teniendo los valores iniciales P0, la medida de la covarianza R y el estado de covarianza Q, el óptimo de estimación de Kalman para los estados puede ser actualizado utilizando los siguientes pasos:

I. Extrapolación del estado estimado: $\widehat{xk}|\mathrm{k}-1 = \mathrm{f}(\widehat{xk-1}|\mathrm{k}-1, \mathrm{uk})$
II. Extrapolación del error de covarianza: $\mathrm{Pk}|\mathrm{k}-1 = \mathrm{FkPk}-1|\mathrm{k}-1\mathrm{F\ T\ k} + \mathrm{Qk}$
III. Ganancia filtro de Kalman: Pk|k-1G T k(GkPk|k-1G T k+Rk)-1
IV. Estado estimado actualizado: $\widehat{xk}|\mathrm{k} = \widehat{xk}|\mathrm{k}-1 + \mathrm{Kk}\left(\mathrm{yk} - \mathrm{g}\left(\widehat{xk}|\mathrm{k}-1\right)\right)$
V. Actualizar error de covarianza: Pk|k=(I-Kk Gk)Pk|k-1

Donde la matriz F_k es el Jacobiano de f (xk-1, uk), y la matriz G_k es el Jacobiano de g (xk).

Cuaternión basado en el filtro de Kalman extendido

Los cuaterniones unitarios tienen muchas aplicaciones en problemas de estimación de estados por su enorme simplicidad. Un cuaternión basado en el filtro de Kalman extendido fue propuesto originalmente por MNAV IMU de Crossbow Technology. Los desarrolladores de UAV han usado esta aproximación en plataformas de uso específico. El sistema de variables de estado incluye ambos cuaterniones unitarios (q) y la evolución del giróscopo (b_p, b_q y b_r).

Las medidas tomadas como observador del sistema son las aceleraciones (a_x, a_y, a_z) y el ángulo de guiñada ψ derivado de los datos que se pueden leer del magnetómetro.

Resulta de especial interés señalar que la ecuación de medida asume que la aceleración medida solo se proyecta a través del vector de aceleración de la gravedad. Sin embargo, asumir esta premisa, puede no ser cierta para pequeños UAVs (vehículos aéreos no tripulados). El filtro de Kalman extendido utilizando ángulos de Euler, puede ser elegido como filtro aplicado a estados del sistema para los problemas de estimación del comportamiento del dispositivo supuesto. Asumiendo que el estado del sistema es un vector x que representa el ángulo modificado, y la salida del sistema es un vector "y" que representa los errores leídos del acelerómetro.

El sistema puede ser modelado como:

$$
\begin{aligned}
x &= \begin{bmatrix} \phi \\ \theta \end{bmatrix}, \qquad \hat{y} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \\
\dot{x} &= \begin{bmatrix} p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\ q\cos\phi - r\sin\phi \end{bmatrix} + v_w, \\
\hat{y} &= \begin{bmatrix} \dot{u} - rv + qw - g\sin\theta \\ \dot{v} + ru - pw - g\cos\theta\sin\phi \\ \dot{w} - qu + pv - g\cos\theta\cos\phi \end{bmatrix} + v_k,
\end{aligned}
\tag{40}
$$

Donde $V_w \sim N(0, Q)$, $V_k \sim N(0, R)$.

De hecho, las velocidades (u, v, w) no son fácilmente medibles a altas frecuencias. La velocidad del aire puede ser utilizada para así simplificar las ecuaciones de medida. Estas simplificaciones pueden ser, por ejemplo, que las derivadas de las velocidades (aceleraciones) sean nulas, que la velocidad v sea también nula, y finalmente:

$$u = Va * cos\theta \tag{41}$$

$$w = Va * sen\theta \quad (42)$$

Donde Va es la velocidad del aire medida en un tubo de pitot.

Tabla 1: categorías IMUs

IMU Type	Navigation Grade	Tactical Grade	Industrial Grade	Hobbyist Grade
Cost ($)	$>$ 50k	10-20k	0.5-3k	$<$ 500
Weight	$>$ 5 lb	about 1 lb	$<$ 5 oz	
Gyro Bias	$<$ 0.1 deg/h	0.1-10 deg/h	$\leq$ 1 deg/sec	$>$ 1 deg/sec
Gyro Random Walk Error	$<$ 0.005 deg/$\sqrt{h}$	0.2-0.5 deg/$\sqrt{h}$		
Accel Bias	5-10 μg	0.02-0.04 mg		
Example	Honeywell HG9848	Honeywell HG1900	Microstrain GX2	ArduIMU

La ecuación de medida se puede simplificar en:

$$\hat{y} = \begin{bmatrix} qV_a \sin\theta + g\sin\theta \\ rV_a\cos\theta - pV_a\sin\theta - g\cos\theta\sin\phi \\ -qV_a\cos\theta - g\cos\theta\cos\phi \end{bmatrix} + v_k \quad (43)$$

Donde $V_k \sim N(0, R)$. El comportamiento puede ser estimado utilizado un filtro de Kalman extendido siguiendo los pasos descritos anteriormente.

Aggie EFK: GPS con filtro de Kalman extendido

Aggie EKF es una extensión para filtros de Kalman añadidos a funcionalidades de GPS propuesta gracias a sus consideraciones por parte de ambos filtros diseñados en secciones anteriores.

Un filtro de Kalman extendido es muy similar a los desarrollados. Sin embargo, la ecuación de medida es reemplazada por una estimación mucho más precisa del vector de gravedad gracias a las medidas de velocidad realizadas con el GPS. El sistema de ecuaciones que se muestra a continuación enfoca su solución a V_g, velocidad medida por el GPS.

$$\dot{q} = \frac{1}{2}\left[\begin{array}{ccccccc} 0 & -\hat{p} & -\hat{q} & -\hat{r} & 0 & 0 & 0 \\ \hat{p} & 0 & \hat{r} & -\hat{q} & 0 & 0 & 0 \\ \hat{q} & -\hat{r} & 0 & \hat{p} & 0 & 0 & 0 \\ \hat{r} & \hat{q} & -\hat{p} & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] q + w_k$$

$$\left[\begin{array}{c} \hat{a}_x \\ \hat{a}_y \\ \hat{a}_z \end{array}\right] = \left[\begin{array}{c} 2g(q_1 q_3 - q_0 q_2) \\ 2g(q_2 q_3 + q_0 q_1) \\ g(q_0^2 - q_1^2 - q_2^2 + q_3^2) \end{array}\right] + v_k, \qquad (44)$$

Donde,

$$\left[\begin{array}{c} \hat{a}_x \\ \hat{a}_y \\ \hat{a}_z \end{array}\right] = \left[\begin{array}{c} a_x \\ a_y - rV_g + g\sin\phi_{t-1} \\ a_z + qV_g + g\cos\phi_{t-1} \end{array}\right] \qquad (45)$$

Y Vk >> N (0, R). El comportamiento estimado del estado puede ser calculado usando los pasos descritos anteriormente.

Filtros complementarios

Aparte de las aproximaciones de los filtros Kalman, existen otros filtros no lineares que trabajan como filtros complementarios. Un ejemplo de ellos se expone a continuación:

$$\dot{\hat{R}} = [(R\Omega)_\times + k_{est}\,\pi_a(\tilde{R})\hat{R}^T]\hat{R},$$
$$\pi_a(\tilde{R}) = \frac{1}{2}(\tilde{R} - \tilde{R}^T), \quad \tilde{R} = \hat{R}^T R, \qquad (46)$$

Donde W es la velocidad angular del cuerpo, R es la matriz de rotación estimada de los acelerómetros, y K_{est} es la ganancia del sistema. El principal reto es fusionar la estimación que proviene de los giróscopos y la proveniente de los acelerómetros.

Ejemplos IMUs low-cost

En este apartado se van a comparar algunos de los ejemplos más representativos de IMUs de bajo coste, centrándose en sensores hardware, con precisión aceptable y fácilmente modificable por software. Las especificaciones funcionales de los dispositivos hardware se compararán al final del apartado.

- Comportamiento esperado de IMUs

  El Microstrain 3DM-GX2 IMU tiene un sensor inercial común incluyendo un giróscopo de tres ejes, un acelerómetro de tres ejes y un magnetómetro de tres ejes igualmente. Este sensor puede tener a su salida medidas de ángulo basadas en ángulos de Euler o bien matrices de rotación a una frecuencia superior a 250 Hz trabajando con un sensor con un ancho de banda de 100Hz. El 3DMGX2 IMU puede conectarse fácilmente con otras unidades a través de comunicación serie RS232/422 o bien mediante USB. Otra ventaja de este sensor es su resistencia a las interferencias (por golpes). Existen IMUs similares como el 3DM-GX4, una opción muy recomendable, también de Microstrain pero con mejores prestaciones y VN100 de la marca Vector Nav Technologies.

- GPS añadido a IMUs

  El GPS o sistema de navegación inercial puede ser el tipo de dispositivo que contenga un conjunto de todas las unidades o bien, estar compuesto de unidades separadas según los sensores utilizados e incluidos en él. Desafortunadamente, cuando se integra GPS a una IMU el precio resulta demasiado elevado. Por ello, es imprescindible estudiar a fondo los requisitos del proyecto previa compra del sensor, dado que en muchas aplicaciones no resulta ventajosa la inclusión del GPS. El MTi-G IMU de Xsens es una buena opción para este tipo de sensores. Realiza estimaciones de comportamiento y posición a una frecuencia superior a 120Hz. Sin embargo, este IMU cuesta alrededor de $3,000. Al agregar un GPS en un sensor inercial, este puede ser combinado como dos unidades aisladas (GPS e IMU) y una central de procesado, razón por la cual el coste aumenta considerablemente.

- Hobbyist Grade IMUs.

  Recientemente, muchos IMUs han comenzado a estar disponibles con tres ejes para giróscopo y acelerómetro. Este modelo trabaja a una frecuencia superior a 50Hz. Un buen ejemplo sería Sparkfun Razor IMU.

Tabla 2: Comparativa IMUs Low-cost

Specifications	Microstrain 3DM-GX2	Microstrain 3DM-GX3	Xsens Mti-g (w. GPS)	ADIS16405	Ardu-IMU	VectorNAV VN-100
Size (mm)	$41 \times 63 \times 32$	$44 \times 25 \times 11$	$58 \times 58 \times 33$	$23 \times 23 \times 23$	$28 \times 39 \times 12$	$75 \times 75 \times 17$
Wight (g)	39	18	68	16	6	36
Orientation Accuracy (static)	$\pm 0.5°$ typical	$\pm 0.5°$ typical	$< 0.5°$(roll/pitch)	Only raw data	N/A	$< 0.5°$(roll/pitch)
(dynamic)	$\pm 2.0°$ typical	$\pm 2.0°$ typical	$1°$ RMS			$< 2.0°$ typical
Update Rate	$< 100Hz$	$< 1000Hz$	$< 120Hz$	$< 330Hz$	$< 50Hz$	$< 200Hz$
Gyro Bias (°/sec)	$\pm$ 0.2	± 0.2	1	± 3(initial) 0.007 (in-run)	N/A	<0.028 $(25°C)$
Gyro Range(default)	$\pm 300°$/sec	$\pm 300°$/sec	$\pm 300°$/sec	$\pm 300°$/sec	$\pm 300°$/sec	$\pm 500°$/sec
Gyro Nonlinearity	0.2%	0.2%	0.1%	0.1%	1%	$< 1\%$
Gyro Random Walk Error	N/A	N/A	$0.05°/\text{sec}/\sqrt{}\text{Hz}$		N/A	N/A
Accel Bias	± 0.005g	± 0.005g	± 0.002g	± 0.05g(initial) 0.0002g(in-run)	N/A	± 0.0005g(X,Y) ± 0.0016g(Z)
Accel Range(default)	± 5g	± 5g	$\pm 50 m/s^2$	± 18g	± 3.6g	± 2g
Accel Nonlinearity	0.2%	0.2%	0.2%	0.1%	0.3%	$< 0.5\%$
Mag Bias (Gauss)	± 0.01	± 0.01	0.0001	± 0.004	N/A	± 0.000125
Mag Range (Gauss)	± 2.5	± 1.2	± 0.75	± 2.5	N/A	± 6
Mag Nonlinearity	0.4%	0.4%	0.2%	0.5%	N/A	$< 1\%$

Direcciones futuras

Hoy en día, los sensores inerciales y los sistemas de tecnologías no navegables están sujetos a grandes avances y cambios continuos cada año. Existen todavía muchas nuevas investigaciones que introducen ideas innovadoras y que se incluyen en la metodología IMUs.

IMU basado en flujo óptico: se basa en la idea de la capacidad de algunos seres vivos para poder planear, que es la idea básica del flujo óptico. Con la mejora de la precisión de los chips de flujo óptico, un IMU basado en este tipo de tecnología será altamente recomendable.

Red de trabajo acelerómetro-giróscopo: sería de pequeño tamaño y utilizaría capacidades de tecnología Wireless. Hace posible el empleo de un giróscopo o acelerómetro en vehículos que no necesitan navegación. Estos sensores, pese a parecer redundantes en este tipo de vehículos, pueden combinarse para una estimación del movimiento altamente precisa.

GPU: la mayoría de IMUs suelen estar más o menos limitados por la capacidad computacional. Los gráficos procesados unitariamente pueden exponerse en condiciones de un intenso coste computacional, por ejemplo, el tan usado procesado de imágenes en tiempo real para estimaciones de comportamiento.

FOKF: Fractional Order Kalman Filter. Este filtro puede producir diferentes aproximaciones de algoritmos de estimaciones no lineares. La razón de la utilización del filtro de Kalman viene de asumir el ruido Gaussiano del sensor. Sin embargo, el ruido no Gaussiano puede ser mejor adaptado gracias a que se asumirán cálculos con infinitas dimensiones.

4.2.2. IMU SEN 10736 ROHS

Se ha decidido la compra de este sensor para la realización de las pruebas en el presente Trabajo Fin de Máster. Se trata de una IMU que incorpora tres sensores: un giróscopo de tres ejes, ITG-3200, un acelerómetro de tres ejes, ADXL345 y un magnetómetro, también de tres ejes, HMC5883L. Se obtiene un total de 9 grados de libertad en la medida inercial de la IMU.

Las salidas de cada uno de los sensores pasa por una placa de Arduino® ATmega328 procesada por puerto serie. La placa se programa con un Arduino® de 8MHz (stk500v1) conectándolo a puerto serie mediante los pines de envío y recepción (Tx y Rx) alimentados a 3,3V. Se utilizará una velocidad de transmisión de 57600 baudios. El tipo de Arduino® es el Pro o el Pro Mini.

Figura 27: IMU SEN 10736 ROHS

Características

- 9 Grados de libertad:
  - ITG-3200 giróscopo de 3 ejes
  - ADXL345 acelerómetro de 3 ejes, ±16g. 13 bits de resolución
  - HMC5883L magnetómetro digital de 3 ejes.
- Salidas procesadas mediante ATmega328 y comunicación por puerto serie
- Pines de salida combinados con FTDI
- 3,5-16 VDC input
- Interruptor ON/OFF y función RESET
- Dimensiones: 28x41 mm

Preparación Hardware

Para poder utilizar el dispositivo se debe soldar una tira de pines para que ayudados de pines acodados, se pueda tener conexión entre la placa que contiene los sensores, y la que contiene el conector de puerto serie. Además, para que la alimentación sea de 3,3V se debe poner un punto de soldadura, por recomendación del fabricante, en la placa que contiene el conector puerto serie.

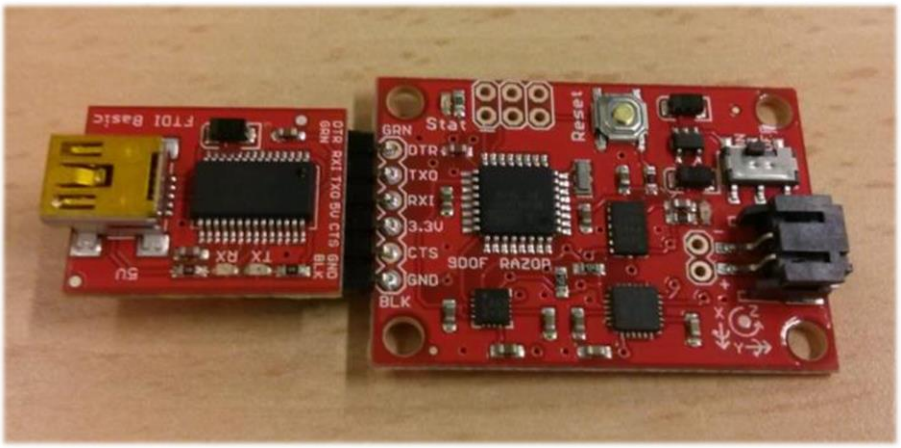

Figura 28: IMU sensores y comunicación serie

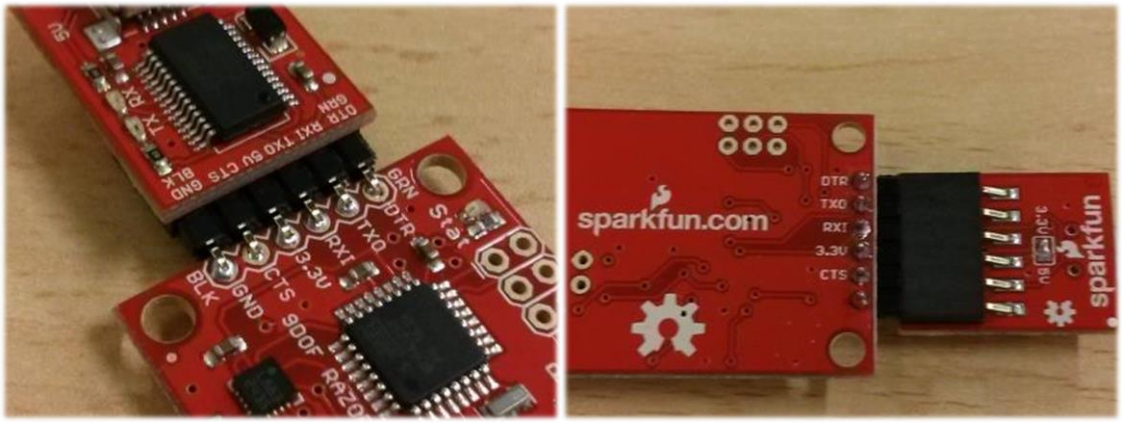

Figura 29: Conexión por pines

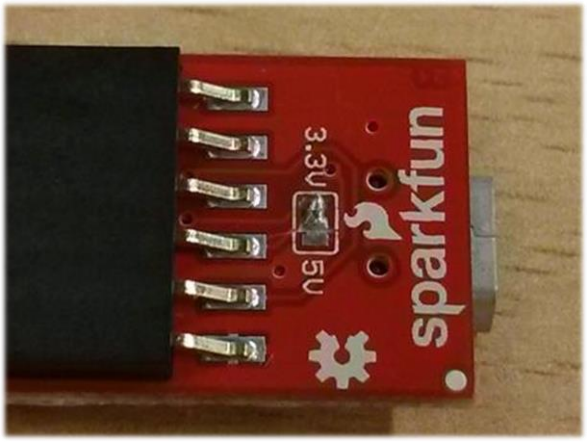

Figura 30: Elección tensión de 3.3V

Preparación Software

Para poder instalar y calibrar la IMU se necesita tener instalado el software de Arduino® y el Firmware Razor AHRS.

Aunque se desee trabajar con Bluetooth más adelante, se instala utilizando el USB conectándolo directamente al ordenador.

Utilizando Arduino®, se abre el archivo Razor_AHRS.ino. Se debe *descomentar* en la parte del código titulada "USER SETUP AREA" la opción hardware que corresponde al IMU seleccionado. Se debe elegir el tipo de Arduino® que posee el sensor y además, especificar el puerto serie utilizado para la comunicación. Se ejecuta el programa con todas las opciones anteriores seleccionadas.

Calibración de sensores

Con el software de Arduino® se debe ejecutar el archivo Razor_AHSRS.ino. En el código, nos situamos en la parte de calibración y ahí se pondrán los diferentes valores de calibración que se consigan en el monitor serie. Para ello, se abre el monitor de puerto serie y se escribe el comando *#oc* para poder comenzar con *output mode to calibration*. Se comienza calibrando el acelerómetro. Para ello se debe mover en torno al eje X, eje Y además de en torno al eje Z el dispositivo de manera muy lenta, para que la acción de la gravedad pueda ser medida con propiedad. Se debe escribir cada cierto tiempo el comando *#oc* para asegurar que los resultados mostrados tengan cierta convergencia. Con todo ello, se escribirán los valores mínimo y máximo de los tres ejes en el código abierto en Arduino®.

Lo siguiente será calibrar el magnetómetro. Para ello, se escribe en la ventana de comandos la orden *#on* para pasar al siguiente sensor. Se realizan los diferentes movimientos en torno a los 3 ejes pero esta vez no es necesario manejar con sumo cuidado la IMU en cuanto a los movimientos que se le imponen, sino que, simplemente cada una de las pruebas debe comenzar en el Norte y terminar el movimiento en el Sur, girando el dispositivo en torno al eje formado entre Este y Oeste. Se toman esos valores y se proyectan en el código de Arduino®.

Por último, se vuelve a escribir *#on*, es decir, continuar con el siguiente sensor, para poder calibrar el giróscopo. Esta vez, se debe poner el Razor sobre la mesa, y sin moverlo esperar a que los valores de x, y, z se estabilicen. Se trascriben esos valores al código y se tendrá la IMU Razor calibrada.

El magnetómetro necesita una calibración adicional. Se realiza con el software "*Processing*". Se actúa sobre el código *Processing/Magnetometer_calibration* y se procede a mover el Razor en todos los ejes posibles. Se podrá ver una imagen parecida a la siguiente:

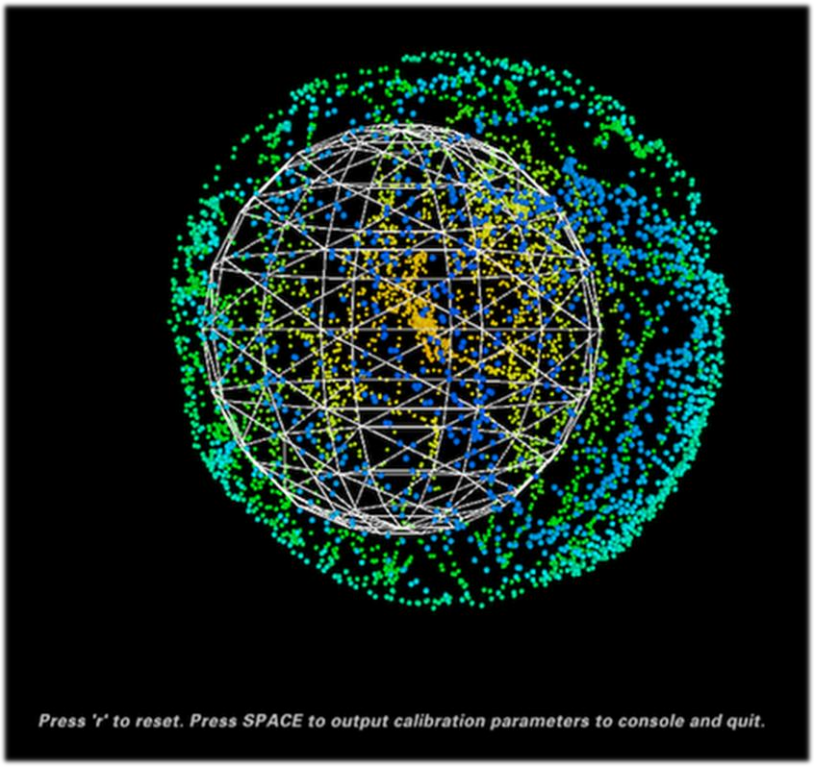

Figura 31: Calibración extendida Magnetómetro

De esta calibración se obtienen una serie de datos que serán los que se deben incluir en el código Razor de Arduino® en la parte correspondiente indicada por el fabricante. La calibración estará, por tanto, completa con este último paso.

4.3. Software: pruebas de control

Para la realización de las pruebas de control con los diferentes inputs se realiza un código en lenguaje c++ que permite la visualización del ángulo descrito por los sensores, tanto encoder como IMU, según las diferentes entradas y transformaciones algebraicas realizadas para llevar a cabo la adquisición de datos en ambos sensores.

Se decide realizar este código para comprobar la diferencia existente en la medición de los ángulos que se describen durante la marcha entre el tobillo, la tibia y el fémur, tanto con el encoder como con la IMU. El primer cambio significativo ha sido la sustitución del encoder por un potenciómetro más resistente al ruido que pueda haber durante la adquisición de los datos.

Aunque se ha comprobado que, efectivamente, el ruido es menor, la pretensión máxima es sustituir estos sensores por la IMU, incluso poder sustituir los sensores de presión plantares, utilizando IMUs en diferentes posiciones sobre la ortesis que permita realizar el control de igual modo al que se estaba realizando hasta el momento, disminuyendo el número y variabilidad de sensores y obteniendo resultados más exactos y precisos.

Para ello se utiliza una tarjeta de adquisición de datos NI®DAQmx-6212 que permite adquirir datos de entrada analógica y la lectura del encoder en tiempo real. Para ello se conecta el sensor a la tarjeta de adquisición y se obtiene la lectura del mismo a través de la programación en c++.

Figura 32: Encoder conectado a tarjeta adquisición

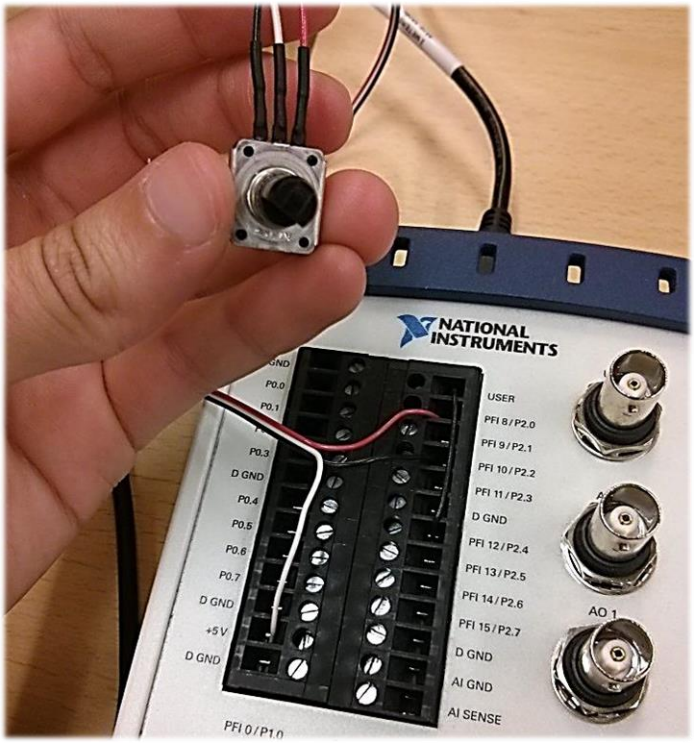

Figura 33: Conexión patillas encoder

A la vez, se codifica para la lectura de la IMU, obteniendo el cuaternión que describe el giro de los sensores que componen el dispositivo inercial. Estos datos guardados en forma de cuaternión serán descompuestos y tratados en Matlab® para conocer los ángulos principales del sensor: yaw, pitch, roll.

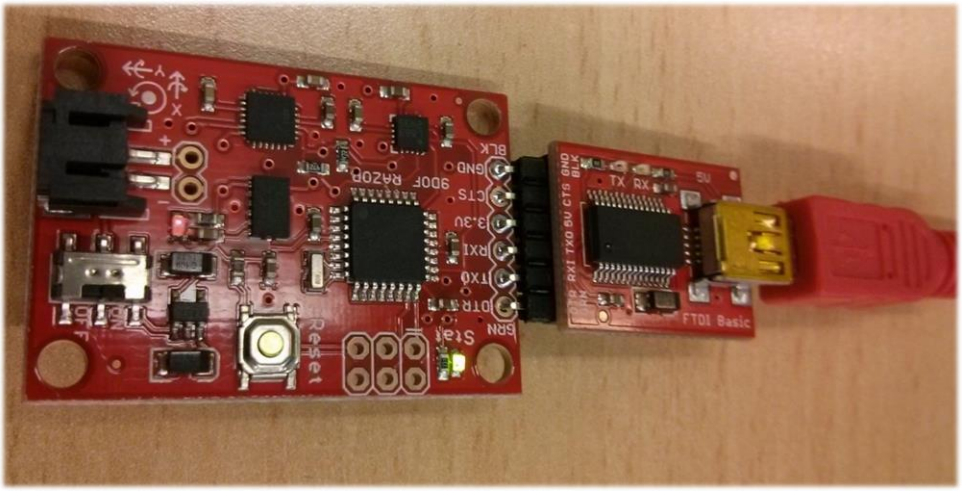

Figura 34: IMU conexión microUSB

Se programa una clase que permite la conversión del cuaternión en una matriz de rotación. De esta matriz de rotación, el ángulo que resulta de especial interés es el resultante del giro alrededor del eje Y, concretamente la componente en z de este ángulo que se obtiene a partir de la siguiente expresión:

$$\widehat{\theta pitch} = arcsen(-2 * (qx.qz - qy.qw)) \quad (47)$$

Se ha creído conveniente no utilizar finalmente esta clase, por temas de precisión en la conversión, y realizarla posteriormente en Matlab® donde además de obtener mediante una función que contiene el propio software, se podrá plotear el resultado obteniendo la comparativa visual que se perseguía desde el principio.

4.3.1. Entorno de software

Se ha utilizado el software de programación Microsoft® Visual Studio® 2013 utilizando la aplicación de ventana de comandos, en lugar de programación orientada a objetos. Este software permite crear código con una serie de ventajas, facilitando funciones para creación de vectores, listas, puntos, etc. añadiendo la ventaja de una programación muy completa y concurrente, sin falta de la secuencialidad a la que se acostumbra en otros lenguajes de programación y softwares. Se ha decidido utilizar lenguaje de programación c++ puro, para poder realizar la extensión a cualquier otro tipo de software sin necesidad de realizar grandes cambios en el código.

4.3.2. Statechart

El dispositivo en el que se basa el presente TFM posee un funcionamiento continuo, sin necesidad de interacción con el usuario, dado que la pretensión es minimizar la sensación de control remoto hacia el paciente. Por ello, se rehúsa utilizar botonera o similar para evitar externalizar al paciente cualquier sensación de monitorización. A tal efecto, se introduce en un ordenador embebido todo el software necesario, inputs, control y procesamiento de señales. El siguiente esquema muestra el funcionamiento general de la ortesis, un funcionamiento lo más cercano a la sencillez posible.

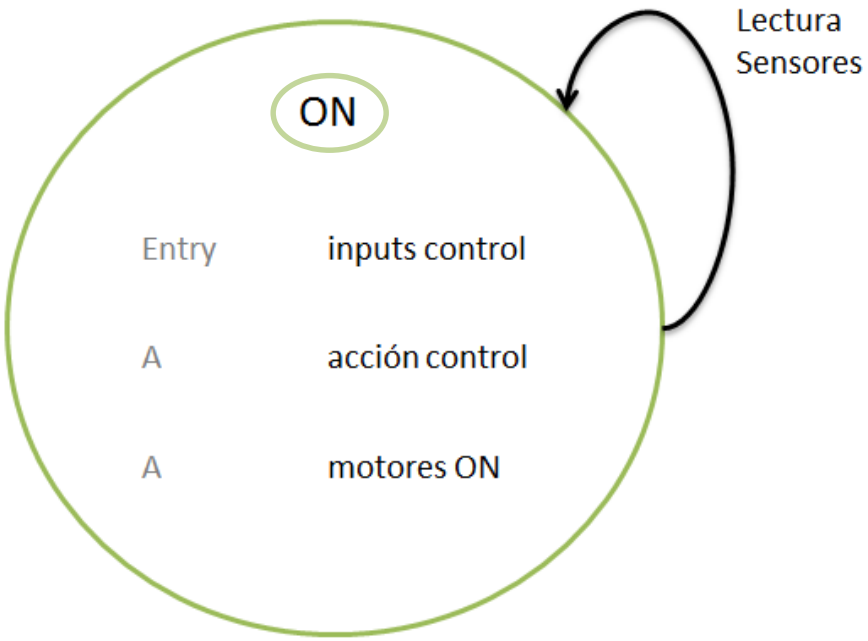

Figura 35: Funcionamiento general del sistema

A continuación se representa el diagrama de estados correspondiente a una sola ortesis. Se debe tener en cuenta que el control se basa precisamente en que una de las ortesis haga un determinado movimiento en función de la posición y acción de control que esté efectuando o en la que se encuentre la ortesis de la pierna contraria.

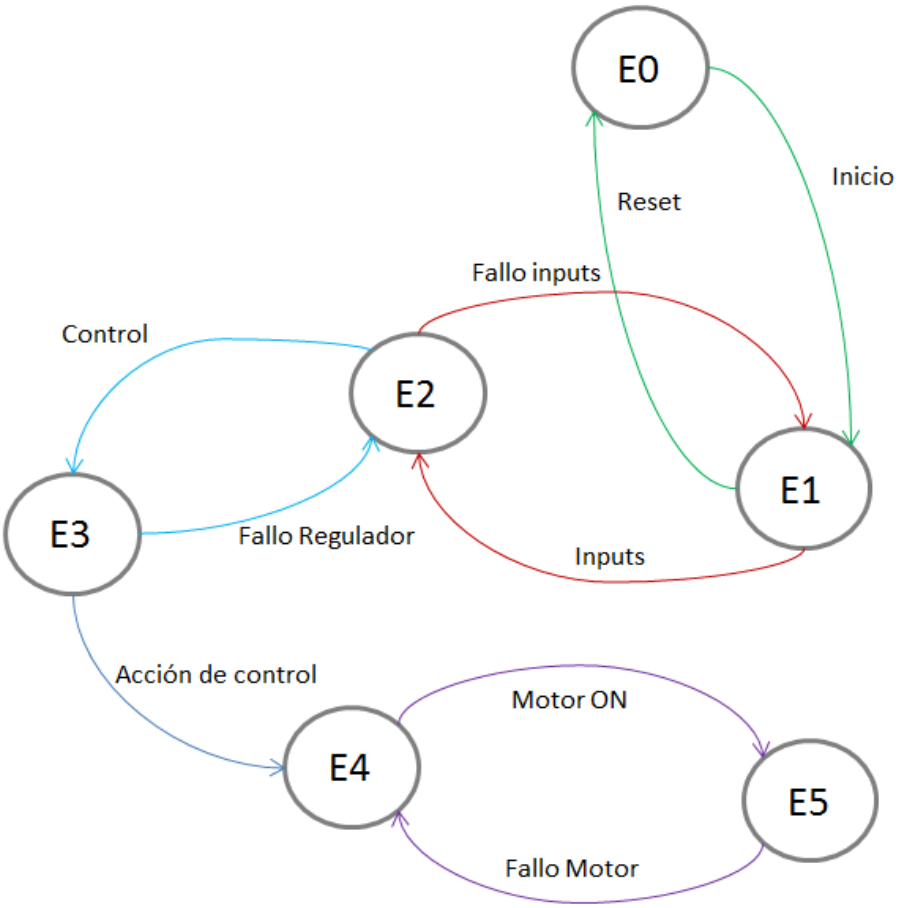

Figura 36: Diagrama de estados de una ortesis

4.3.3. Lógica de funcionamiento

En este subapartado se dará una explicación del funcionamiento de la ortesis objeto de estudio del presente Trabajo Fin de Máster. Se trata de un funcionamiento que se basa en la entrada de inputs al control para conseguir el movimiento de ambas piernas.

Las ortesis poseen un motor de tipo EPOS 2 con tarjeta controladora Maxon EC45. EPOS2 situado a la altura de la rodilla. Las entradas al control serán los ángulos que describen los tobillos. De esta forma, al control le entra como input el ángulo que describe uno de los tobillos, por ejemplo, el izquierdo, de tal manera, que ese ángulo será clave para que la rodilla derecha describa un determinado ángulo impuesto por el control de la ortesis.

Esto permitirá que según el ángulo descrito por el tobillo de la pierna contraria, se podrá conocer el ángulo que debe describir la rodilla de la otra pierna para conseguir el movimiento de marcha buscado.

De ahí la importancia de la exactitud de los inputs al control. Se debe tener en cuenta que la consecución de movimientos depende totalmente de los inputs. Con el ángulo de tobillo correctamente

medido, se podrá implementar el control correctamente para que el movimiento sea adecuado. Los ángulos introducidos deberán ser correctamente medidos.

Otra fuente de diversificación de resultados se encuentra en la posición de inicio de la marcha. Cuando se comienza a caminar, existe un ángulo característico entre el pie y la tibia. Sin embargo, se debe tener en cuenta que este ángulo puede deberse a un movimiento de giro del cuerpo sobre esa pierna o incluso simple balanceo sin necesidad de tener intención de emprender marcha para comenzar a caminar.

Por tanto, es importante que si se forma ese ángulo en uno de los pies, no se active el motor de la pierna contraria para la realización de una contracción de la rodilla de esa pierna contraria, si no que exista cierta indeterminación que pueda ser resuelta mediante software para que el movimiento de marcha no sea inmediato al tener como input un determinado ángulo.

Un punto de reflexión importante es cómo conseguir filtrar este tipo de movimientos, sin que el control confunda los inputs, es decir, se desea que el control consiga "interpretar" la intención del paciente a la hora de realizar cualquier tipo de movimiento, que no emprenda marcha sin cumplir una serie de condiciones además de la esencial del ángulo del tobillo de la pierna con la que se comienza el movimiento de *swing*.

La posición de los sensores provoca una alta sensibilidad ante el funcionamiento. Si lo que se desea medir es el ángulo del tobillo, debe tener el eje perfectamente alineado con lo que se quiere medir, movimientos de rotación alrededor de un eje imaginario sobre el que gira la propia articulación. En el momento en que se desea incluir la IMU, se debe alinear de manera bastante precisa perpendicularmente al eje descrito por la tibia. Es decir, se desea medir un ángulo que oscila alrededor de un eje paralelo a la tibia, que permitirá la medida del ángulo "pitch" en el caso de la situación cómo se ha posicionado en la ortesis.

4.3.4. Código

En este subapartado se van a explicar los diferentes códigos de programación que se han desarrollado para el correcto funcionamiento de los sensores en combinación con la simulación que se ha realizado para obtener resultados sin necesidad de probar las diferentes formas de control en el conjunto mecánico físico.

Relación entre sensores: Encoder e IMU

Se ha programado un código para relacionar el comportamiento entre ambos sensores, encoder e IMU. Originalmente, el ángulo que describe cada uno de los tobillos era medido mediante encoders de 4096 pulsos por revolución de resolución. Una primera evolución tecnológica fue el cambio de encoders a potenciómetros de medida. Carecen de una precisión tan elevada como los encoders instalados, sin embargo para esta aplicación, un error de medida de una décima parte de un grado, se puede considerar despreciable, dado que el movimiento será relativamente parecido dependiendo del ángulo medido en el tobillo y del control ajustado para conseguir el movimiento adecuado de la rodilla de la pierna contraria en función del ángulo descrito en el tobillo de la pierna que inicia el movimiento.

Este código consiste en la lectura simultánea de ambos sensores y la recogida en archivos con extensión *.dat de las salidas que aportan cada uno de ellos. Los archivos de salida son tratados en Matlab® mediante un código que permite la lectura de archivos externos y el posterior graficado de los resultados.

En el código principal, escrito en lenguaje c++, recoge los ángulos medidos por la IMU en forma de cuaterniones. El encoder es leído en grados sexagesimales. La transformación de la lectura de la IMU se realiza mediante una función implementado en Matlab® que permite la transformación de los cuaterniones en los ángulos de pitch, yaw y roll. Dependiendo de la orientación de la IMU, será uno u otro ángulo el que interese para el estudio. En este caso, se ha colocado la IMU en una posición vertical, con el cableado USB enfocado hacia la rodilla de la pierna en la que está colocado. Por tanto, el ángulo que interesa conseguir es el "Pitch".

Figura 37: Posiciones sensores en ortesis

```
function sensor_lecture

%Conversión de Quaternion leído por la IMU a ángulo de rotación
%El ángulo interesante es el arcsen(Yz)


for i=1:1:size(IMU,1)

    [yaw(i), pitch(i), roll(i)] =
                quat2agle([IMU(i,1),IMU(i,2),IMU(i,3),IMU(i,4)]);
```

Las pruebas se realizarán mediante el movimiento de la ortesis emulando el movimiento de la marcha humana al caminar. Se recogen los ángulos durante este ciclo de funcionamiento y se importan los resultados en Matlab® para su posterior ploteado.

Los resultados se exponen en el siguiente apartado de base de datos de pruebas. En las gráficas se puede ver una gran concordancia en los resultados obtenidos de uno y otro sensor, de lo que se puede deducir que el cambio de los encoders por IMUs resultará exitoso y con resultados de igual grado de precisión.

La programación de este código parte de la creación de objetos para la lectura del puerto serie, la lectura del encoder con una resolución de 4000 pasos por revolución y un objeto de tipo analógico para la creación de un reloj que marcará la frecuencia de adquisición de datos. Se prepara el código para el entendimiento entre el propio código y la tarjeta de adquisición de datos.

```
encoder.addChannel ("Orthosis/ctr0", 4000, true, "/Orthosis/PFI0", "/Orthosis/PFI1",
"/Orthosis/PFI2");                //(encoder q)
std::cout << "Adding encoder channel" << std::endl;


clk.addChannel ("Orthosis/ai0"); //solo se utiliza un canal analógico (orthosis)
std::cout << "Adding analog channel" << std::endl;
```

Los datos se irán almacenando en un buffer que se ha creado dentro de una static union para, en combinación con una estructura, conseguir la correspondencia entre el espacio dedicado al guardado de los datos y la consecución de los mismos.

Existe un gran problema y es que el encoder no lee a una velocidad tan alta como la IMU, por lo que existen valores que no pueden rellenarse en las tablas de datos, dado que el oscilador del microcontrolador de la IMU es poco preciso, y su frecuencia es un poco superior a los 100 Hz requeridos. A estos espacios vacíos en la tabla, c++ les asigna por defecto unos valores arbitrarios que poco tiene que ver con los datos que se están recogiendo. Para evitar elementos indeseables en la tabla de valores de medida de los sensores, se procede a implementar una posible solución, recogiendo previamente los datos del encoder mediante la función *peek*.

```
/*… Parte de código de lectura y almacenamiento en el buffer… */

jd = encoder.peek ( &encj );
temp[jd] = encj;

if ( jd - ( jd - 1 ) > 1 )
    temp[j] = temp[jd - 1];
else
    temp[j] = encj;
```

Código de movimiento

Este código se basa en la lectura de los archivos de captura de movimiento, tratado de cada uno de los datos para representar el movimiento de fémur, tibia y pie de cada una de las piernas.

Se crea correlación con el programa Open Scene Graph para la programación de la simulación del movimiento con este software. Se codifica la iluminación y las cámaras necesarias para la correcta visualización de la simulación.

Para conseguir vislumbrar con mayor claridad la diferencia entre el movimiento correcto de la pierna y el conseguido mediante la captura de movimiento de la paciente, se transforma una de las piernas (la derecha) en un material transparente, que permite ver la diferencia entre una y otra y realizar una comparativa visual más directa.

Explicación bucle principal

Se leen los archivos exportados de Blender® que son los huesos que intervienen en el movimiento. En este caso interesa la pelvis, el fémur, la tibia y el pie. Estos archivos, se deben convertir en matrices de transformación para su posterior tratamiento. Se trasladan según las medidas del modelo de cuerpo entero de Blender®. Todas las medidas están referidas a la cadera, como centro de gravedad y origen del esqueleto. Con todos los huesos leídos y transformados en matrices, se crea un grupo de todos ellos, de tipo *switch*, que permitirá ocultar aquellos huesos que no sean necesarios en la simulación, mediante la adicción de comandos de tipo true/false.

```
osg::ref_ptr<osg::Switch> root = new osg::Switch;

  root->addChild ( mat_trans0.get ( ) , true );
```

Se crea un objeto de la clase manipulador. Este objeto permitirá el cambio de vista isométrica y vistas principales de planta, alzado y perfil durante la visualización de la simulación. El cambio de vistas se conseguirá presionando diferentes teclas que deberán ser explicadas a modo de instrucciones de uso de la simulación. Normalmente, se utilizan los cuatro primeros números naturales.

Se realiza la lectura de los archivos que contienen los datos de la capturas de movimiento. Estos archivos son de tamaño 157x174 (filas, columnas). Existen, por tanto, unos archivos de datos que permiten conocer la ubicación de cifras que interesan para cada uno de los sólidos y la relación entre los vectores de posición y los vectores u y v que representan cada uno de los huesos participantes.

Por tanto, estos datos son almacenados en matrices, tanto las matrices de movimiento como sus derivadas en el tiempo, lo que permitirá su posterior utilización en el código de simulación del movimiento de marcha humana.

Se hace, además, la lectura de los archivos que almacenan los resultados de las placas de fuerza, para tener constancia de cuál es el pie que está en contacto con el suelo para posteriormente aplicarle el control necesario a la pierna contraria en función de que sea el izquierdo o el derecho.

A partir de este momento, la simulación se implementa en un bucle *for* que permite la computerización de las órdenes pertinentes a lo largo de toda la matriz de movimiento.

Para cada uno de los huesos participantes se realiza la búsqueda de sus vectores en la matriz *q* de movimiento y se compone la matriz de transformación de cada uno de ellos como un vector de 16x1 que conserva el siguiente orden:

{Vector u; 0; vector v; 0; vector w; 0; vector posición proximal; 1}

De la misma forma, se llevan a cabo idénticos cálculos matemáticos para la pelvis y para el fémur, tibia y pie de la pierna izquierda y la pierna derecha. Se añaden los pasos necesarios para el cálculo computerizado necesario para obtener las matrices de fémur y tibia derivada que intervendrán en el cálculo del eje de rotación de la rodilla (en este caso solo se ha calculado en la pierna derecha) derecha, puesto que la izquierda sería el mismo eje de rotación con efecto espejo en el plano Y.

```
int pointf_i = 141;//punto FEMUR de la tabla de ip para obtener correspondencia en q
size_t posf_i = std::distance ( ip[0].begin ( ) , std::find ( ip[0].begin ( ) , ip[0].end
                                                              ( ) , pointf_i ) );
double rPointf_i[3] = { 0 };

for ( int i = 0; i < 3; i++ )
       rPointf_i[i] = q[filas][ip[1][posf_i] + ( i - 1 )];

int vector_uf_i = 141;              //punto FEMUR vector u de la tabla de iv para obtener
                      correspondencia en q. El femur es el 111, se busca su correspondiente
int vector_vf_i = vector_uf_i + 1; //vector v correspondiente al FEMUR
size_t vec_uf_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin ( ) , iv[0].end
                                                            ( ) , vector_uf_i ) );
size_t vec_vf_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin ( ) , iv[0].end
                                                              ( ) , vector_vf_i ) );
double uVectorf_i[3] = { 0 };
double vVectorf_i[3] = { 0 };

for ( int j = 0; j < 3; j++ )
{
       uVectorf_i[j] = q[filas][iv[1][vec_uf_i] + ( j - 1 )];//para que empiece justo en
                                            la posición que indica la tabla (añadir j-1)
       vVectorf_i[j] = q[filas][iv[1][vec_vf_i] + ( j - 1 )];
}


//se calcula el punto proximal del fémur para poder calcular la norma entre el punto distal
y el proximal

int point_proxf_i = 13;                         //punto FEMUR de la tabla de ip para obtener
                                                              correspondencia en q
size_t pos_proxf_i = std::distance ( ip[0].begin ( ) , std::find ( ip[0].begin ( ) ,
                                                    ip[0].end ( ) , point_proxf_i ) );
double rPoint_proxf_i[3] = { 0 };

for ( int i = 0; i < 3; i++ )
       rPoint_proxf_i[i] = q[filas][ip[1][pos_proxf_i] + ( i - 1 )];


//Vectores para calcular la norma con la que se forma el vector W

double U_i[3] = { 0 };
```

```
double Ut_i[3] = { 0 };
double Utb_i[3] = { 0 };
double Up_i[3] = { 0 };

double norma_distal_f_i = 0;
double norma_distal_t_i = 0;
double norma_distal_tb_i = 0;
double norma_distal_p_i = 0;

//Vector intermediario para cálculo de la norma

for ( int j = 0; j<3; j++ )
	U_i[j] = rPointf_i[j] - rPoint_proxf_i[j];

norma_distal_f_i = norma.Norm (U_i);
//vector W
double wVectorf_i[3] = { 0 };
for ( int i = 0; i < 3; i++ )
	wVectorf_i[i] = U_i[i] / (-norma_distal_f_i);


//la matriz de 16x1 tendrá los elementos, "comenzando por 0", [3], [7] y [11] como ceros y
el elemento [15] como un 1 EN TODOS LOS SÓLIDOS

m1_i [3] = m1_i [7] = m1_i [11] = 0;
m1_i [15] = 1;

//Entre [0] y [3] es el vector u; entre [4] y [7] es el vector v; entre [8] y [11] es el
vector w; entre [12] y [15] es el vector del punto proximal

for ( int i = 0; i < 3; i++ )
{
	m1_i [i] = uVectorf_i[i];
	m1_i [i + 4] = vVectorf_i[i];
	m1_i [i + 8] = wVectorf_i[i];
	m1_i [i + 12] = rPoint_proxf_i[i];
}
```

En aquellos huesos cuya forma y definición no es de tipo alargado (tibia, fémur), es decir, la pelvis y el pie, no se puede calcular el vector *w* como la diferencia entre las posiciones proximal y distal dividida entre su norma. En estos casos, el vector w resulta del producto vectorial entre el vector u y el vector v del sólido que representan.

El siguiente paso es calcular el eje de rotación de la rodilla. Para ello es necesario calcular el ángulo que describen ambos tobillos. Este ángulo, como se ha explicado anteriormente, se conoce a partir del cálculo del arco cuya tangente es el cociente entre la norma del producto vectorial de los vectores *u* y *v* de cada uno de los tobillos y el producto escalar de ambos vectores.

```
//Producto vectorial = sen (theta tobillo izquierdo)
temp_tob_iz [0] = uVectort_i [1] * uVectortb_i [2] - uVectort_i [2] * uVectortb_i [1];
temp_tob_iz [1] = uVectort_i [2] * uVectortb_i [0] - uVectort_i [0] * uVectortb_i [2];
temp_tob_iz [2] = uVectort_i [0] * uVectortb_i [1] - uVectort_i [1] * uVectortb_i [0];

//Cálculo de la norma del producto vectorial
norma_temp_tobillo_iz = norma.Norm (temp_tob_iz);
```

```
//Cálculo del ángulo del tobillo izquierdo
theta_tobillo_izq = atan2 (norma_temp_tobillo_iz, (uVectortb_i [0] * uVectort_i [0] + uVec-
tortb_i [1] * uVectort_i [1] + uVectortb_i [2] * uVectort_i [2]));
```

Estos ángulos son necesarios para posteriormente poder aplicar el control al movimiento de la ortesis. Con todos estos datos, se procede a calcular la matriz antisimétrica *W* del fémur, resultado de multiplicar la matriz de transformación del fémur derivado por la transpuesta de la matriz de transformación del fémur sin derivar. Igualmente, se calcula la matriz antisimétrica W correspondiente a la tibia. La diferencia entre ambas (tibia-fémur) da como resultado la matriz antisimétrica global de la que se obtiene el vector representativo del eje de giro de la rodilla. A partir de este vector, se puede conocer el eje de giro en coordenadas globales del sistema, se trata simplemente de hacerlo unitario al dividirlo por su norma. Con este último resultado, se obtendrá el vector de eje de giro en coordenadas locales para el fémur y para la tibia, obteniendo finalmente el eje de giro óptimo y las nuevas coordenadas ejes globales.

Se calculan las proyecciones de los vectores *u* de la tibia y del fémur para conocer el ángulo que describe la rodilla durante el movimiento. Estos vectores serán normalizados para que sean unitarios. La explicación de por qué se utilizan los vectores u radica en que el ángulo que describen entre ellos durante el movimiento debe conservarse entre el sólido que representa el fémur y el que representa la tibia de forma constante.

Con estos vectores proyectados sobre el eje de giro de la rodilla, se puede calcular el ángulo que describe la rodilla durante el movimiento:

theta_rodilla = (atan2 (norma_temp , (u11_pn[0] * u12_pn[0] + u11_pn[1] * u12_pn[1] + u11_pn[2] * u12_pn[2])));

Finalmente, se aplica la ***fórmula de Rodrigues*** para calcular la matriz de transformación relativa de la rodilla, para posteriormente aplicando la formulación explicada en los cálculos de Matlab®, obtener la matriz representativa de la tibia en coordenadas locales del fémur.

```
identity [0] = identity[4] = identity[8] =  1 ;
identity [1] = identity[2] = identity[3] = identity[5] = identity[6] = identity[7] =  0 ;

antisimetrica rodilla[0] = antisimetrica rodilla[4] = antisimetrica_rodilla[8] =  0 ;
antisimetrica rodilla[1] = -erLOptimo11[2];
antisimetrica_rodilla[2] = erLOptimo11[1];
antisimetrica_rodilla[3] = erLOptimo11[2];
antisimetrica_rodilla[5] = -erLOptimo11[0];
antisimetrica_rodilla[6] = -erLOptimo11[1];
antisimetrica_rodilla[7] = erLOptimo11[0];

//Rodrigues Formula

for ( int i = 0; i < 3; i++ )

   square antisim[i] = antisimetrica rodilla[i] * antisimetrica rodilla[0] + antisime-
   trica rodilla[i + 3] * antisimetrica_rodilla[1] + antisimetrica_rodilla[i + 6] * antisi-
   metrica_rodilla[2];
```

```
for ( int i = 3; i < 6; i++ )

   square antisim[i] = antisimetrica rodilla[i - 3] * antisimetrica rodilla[3] + antisime-
   trica rodilla[i] * antisimetrica_rodilla[4] + antisimetrica_rodilla[i + 3] * antisime-
   trica_rodilla[5];

for ( int i = 6; i < 9; i++ )

  square antisim[i] = antisimetrica rodilla[i - 6] * antisimetrica rodilla[6] + antisime-
  trica rodilla[i - 3] * antisimetrica_rodilla[7] + antisimetrica_rodilla[i] * antisime-
  trica_rodilla[8];


for ( int i = 0; i < 9; i++ )
  {
       mat trans relativa rodilla[i] = identity[i] + antisimetrica rodilla[i] * sinf (
       theta_rodilla ) + 2 * (square_antisim[i])*pow ( sinf ( theta_rodilla / 2 ) , 2 );
   }
```

Con estos datos, se puede conocer la matriz de transformación de la tibia vista desde el fémur. Tras este cálculo, se procede a definir la posición del nuevo tobillo y su correspondiente nueva matriz de transformación.

Para la correcta visualización de la simulación, se deben aplicar escalas diferentes a cada uno de los huesos participantes, teniendo en cuenta que para conseguir cualquier hueso de la pierna izquierda, basta con aplicar escala negativa en el eje Y a las matrices de transformación de la pierna derecha.

Se convierte cada una de las matrices de transformación a lenguaje OSG:

```
osg::Matrixd mpel ( m0 );
mat_trans0->setMatrix ( mpel );
```

Se actualiza la visualización mediante la orden *viewer.frame ()*, y se le aplica un *delay* de 10 ms, dado que la frecuencia de muestreo es de 100 Hz, es decir, cada 10 ms se implementará todo el bucle en el que se cambia de fila en la matriz q de movimiento (qProj.dat) y en la que se implementan los nuevos parámetros representativos de posición y vectores, nuevos ejes de rotación y por tanto un movimiento diferente, para finalmente obtener la marcha del paciente al que se le ha realizado la captura de movimiento.

4.3.5. Control de la ortesis

Dentro de este último código mencionado, se realiza el control de la ortesis para que la salida, o ángulo de la rodilla de la pierna contraria a la que se inicia el movimiento, sea calculado a partir del ángulo del tobillo de la pierna en swing.

Para ello, se crea una clase llamada “Control” en la que se incluye el cálculo de este ángulo, la posición de los motores “imaginarios” en los que se le hace actuar a la pierna como si tuviese unos motores que le implicasen el movimiento, simplemente por facilitar la tarea de control en cuestión de conocimiento ingenieril.

Se añade además un filtro de velocidad para filtrar los ángulos de entrada y evitar que se produzcan balanceos indeseados durante la fase de swing. Para cada ángulo de rodilla existe una relación directa con el ángulo que forme el tobillo contrario. Por tanto, para que debido al ruido en la toma de datos, no se distorsione el cálculo de los ángulos correspondientes, se crea este filtro de velocidad, un filtro pasa bajos de primer orden, que evitará que existan varias medidas de un cierto ángulo para el cálculo del correspondiente de la rodilla contraria y por tanto se permite llevar a cabo un control del movimiento más exhaustivo.

Filtro pasa bajos de primer orden. Filtrado de velocidad.

Un filtro pasa bajos se utiliza para suavizar altas frecuencias en el ruido que se produce en la medida de diferentes señales. Un filtro muy común es el que se basa en el control de sistemas discretos de primer orden- o mediante constante de tiempo. Se puede derivar como un filtro discretizado mediante la función de transferencia de Laplace. Se suele discretizar mediante el método de diferenciación de Euler. En este caso, se ha utilizado la derivación mediante filtro (discrete-time) usando el método de diferenciación de Euler. La función de transferencia usada mediante la transformada de Laplace es:

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{T_f s + 1} \quad (48)$$

Donde T_f [s] es la constante de tiempo. U(S) será el input del filtro que se va a realizar y por tanto y(s) será el output de dicho filtro.

Multiplicando en cruz, se obtiene:

$$(T_f s + 1) y(s) = u(s) \quad (49)$$

Y resolviendo el paréntesis:

$$T_f s y(s) + y(s) = u(s) \quad (50)$$

Aplicando la antitransformada de Laplace a ambos lados de la ecuación, se obtiene la siguiente expresión diferenciada (dado que la multiplicación por s implica diferenciación temporal en el dominio del tiempo).

$$T_f \dot{y}(t) + y\,(t) = u(t) \quad (51)$$

Se procede a utilizar la expresión definida en función de t_k, para representar el instante actual de tiempo, o tiempo discreto:

$$T_f\dot{y}(t_k) + y\,(t_k) = u(t_k) \qquad (52)$$

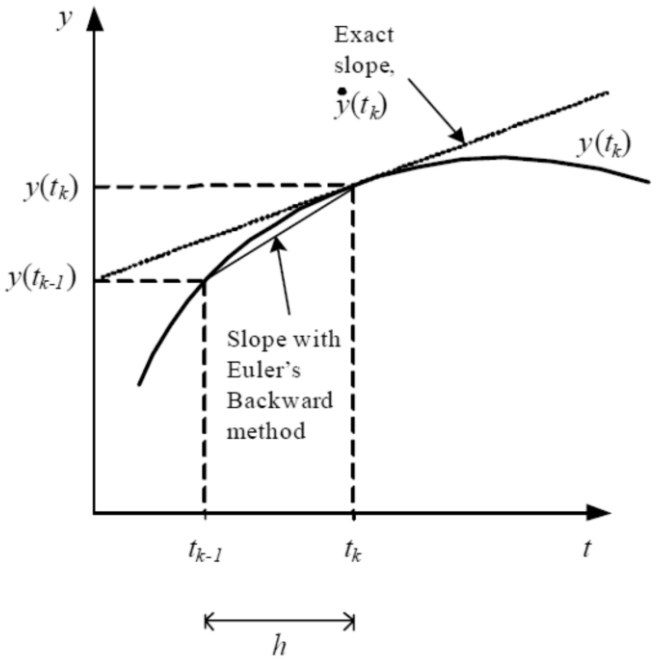

Figura 38: Aproximación de Euler. Diferenciación temporal

Se procede a la sustitución de la expresión de Euler en a términos de diferenciación temporal:

$$\dot{y}(t_k) \approx \frac{y(t_k) - y(t_{k-1})}{h} \qquad (53)$$

Y como muestra la Figura 38:

$$T_f \frac{y(t_k) - y(t_{k-1})}{h} + y(t_k) = u(t_k) \qquad (54)$$

Despejando $\mathrm{y}(\mathrm{t_k})$ se tiene:

$$y(t_k) = \frac{T_f}{T_f + h} y(t_{k-1}) + \frac{h}{T_f + h} u(t_k) \quad (55)$$

Normalmente escrito como:

$$\mathrm{y}(\mathrm{t_k}) = (1 - \mathrm{a})\mathrm{y}(\mathrm{t_{k-1}}) + a\ \mathrm{u}(\mathrm{t_k}) \quad (56)$$

Donde a es el parámetro del filtro descrito por:

$$a = \frac{h}{T_f + h} \quad (57)$$

Que posee un determinado valor cuando se fija la frecuencia de corte o la constante de tiempo. Se debe tener en cuenta que para la implementación en lenguaje de programación de este filtro, hay que guardar el valor anterior y el actual de los inputs y los outputs, dado que son valores clave en el cálculo del valor actual cada vez que se realiza.

Es importante tener en cuenta que el valor de *h* debe ser inferior a la constante de tiempo, de otra manera, el filtro se comportaría de forma diferente a la deseada, distando del comportamiento del filtro continuo original del que ha sido derivado.

Habitualmente se considera que,

$$h \leq \frac{T_f}{5} \quad (58)$$

Codificación del filtro de velocidad

Con el fin de asegurar que cada ángulo de rodilla provenga de un solo ángulo de tobillo, y no exista posibilidad de distorsiones debido al ruido que pueda generar la captación de la señal, se realiza el filtrado dela velocidad (Figura 39) en lugar de la posición (o ángulo en este caso).

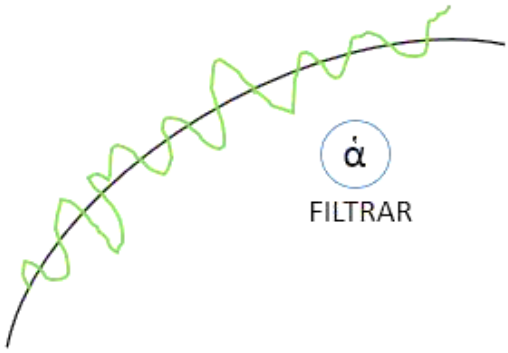

Figura 39: Ruido toma de medidas

Los inputs del filtro serían los ángulos del tobillo que se van capturando en el movimiento. Los outputs, por su parte, serán las diferentes velocidades ya filtradas.

La ecuación queda de la siguiente forma:

$$V_i^f = a\left(\frac{x_i - x_{i-1}}{h}\right) + (1-a)V_{i-1}^f \qquad (59)$$

```
float sampleRate = (5.0*2.0*pi*cutoff_);


if (cutoff_ > 0) //si la frecuencia de corte es mayor que 0
       fltPar = 1.0 / (sampleRate / (2.0*pi*cutoff_) + 1.0); //Cuanto más cercana
esté a de 1, más se mayora la evaluación de entradas y menos la velocidad en el ins-
tante anterior


control_input [1] = control_input [0];
control_input [0] = oppAnk;



control_output [1] = control_output [0];
control_output [0] = fltPar*(sampleRate*(control_input [0] - control_input [1])) +
                                                (1- fltPar)*control_output [1];
```

De manera que se actualicen en cada paso del bucle de movimiento. Para llevar a cabo el control de velocidad, se debe añadir en la evaluación de movimiento (*swing o stance*), las siguientes líneas de código, de tal forma que represente el caso de movimiento de marcha: si el ángulo actual es mayor que el anterior, implica que se está emprendiendo marcha hacia delante, o lo que es lo mismo, la velocidad es positiva. En caso contrario, el movimiento será diferente al de la marcha, y no se debe transmitir movimiento a la rodilla de la pierna contraria, permaneciendo por tanto, en posición estática.

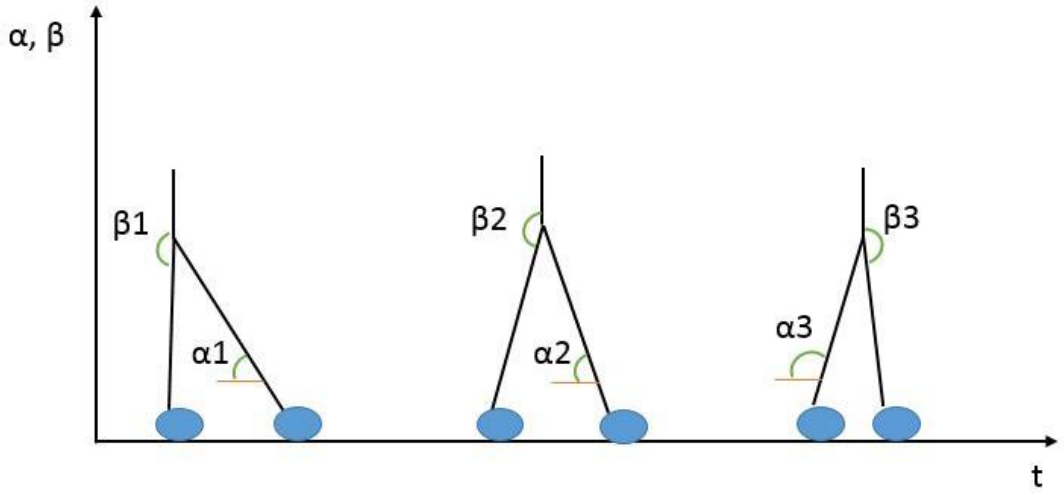

Figura 40: Ángulos durante la marcha

En la Figura 40, en la que α es el ángulo que representa el grado de apertura de la pierna entre fémur y tibia, de la derecha por ejemplo, y β representa el ángulo que forma la pierna contraria con el eje longitudinal del tronco de la persona, se puede ver la tendencia de los ángulos durante la marcha. Esto se refleja en el control de velocidad, donde:

$$\alpha 1 < \alpha 2 < \alpha 3 \quad (60)$$

Y por otro lado, los ángulos de la pierna contraria serán:

$$\beta 1 = \beta 3 = 0 \quad (61)$$

$$\beta 2 = \beta máx \quad (62)$$

Dado que el ruido en la medida puede hacer que no se corresponda un cierto α con su respectivo β, se hace necesario el filtrado de velocidad explicado hasta el momento.

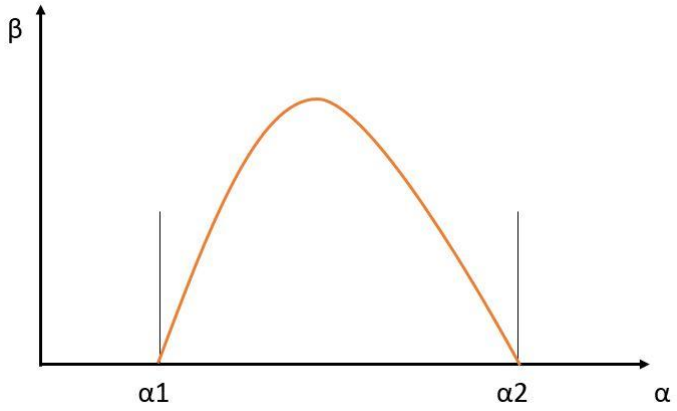

Figura 41: Representación ángulos piernas contrarias

Finalmente, el ángulo de la rodilla contraria se calculará mediante la siguiente función englobada dentro de esta misma clase "*Control*".

```
double motorAngle(const double a) //the input value is the ankle angle
	{
		double knee = 0.0;
		double w = 4.0*atan(1.0)/(fa_-ia_);

		// Phase shift for tuning curve shape
		double phase = ks_*sin(w*(a-ia_)) + kw_*sin(2*w*(a-ia_));

		// Sine function with phase shift
		if (a > ia_ && a < fa_)
			knee = kr_*(sin(w*(a-ia_) - phase));

		return (knee); //Calculated in radians
	}
```

Dado que el control depende fundamentalmente de qué pie esté en contacto con el suelo y por tanto se realice el cálculo a partir del ángulo descrito por uno u otro tobillo, se implementa el código dependiendo de los datos recogidos en los archivos FPL1.dat y FPL2.dat que almacenan en una matriz de tamaño 157x8, los datos de fuerza de cada uno de los pies.

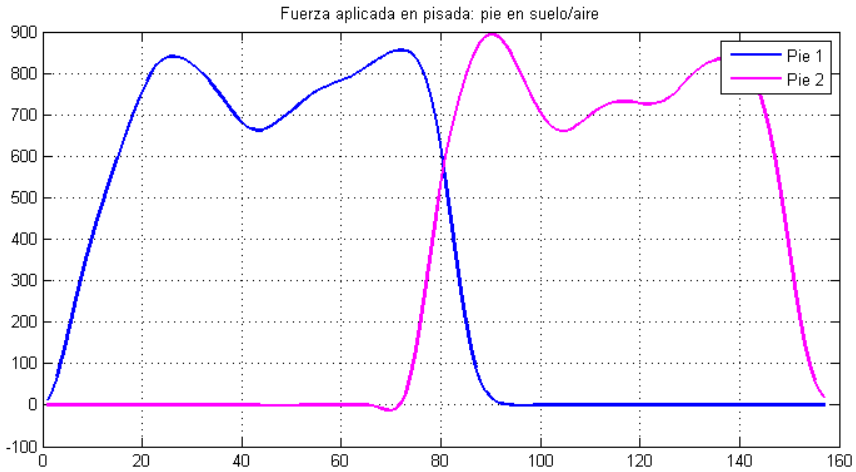

Figura 42: FPL1 en azul; FPL2 en magenta

Existe un problema inherente a esta codificación, y es que no se desea que el control actúe intermitentemente dependiendo del disparo de estas dos matrices (activación y desactivación de los sensores de presión de las placas de fuerza que se utilizaron para la captura de movimiento). Por este motivo, se crea una clase denominada *Schmitt* que permite emular un disparador Schmitt eléctrico delimitando dos franjas: un límite superior que indica si la placa de fuerza está activa (o en estado HIGH), y un límite inferior, a partir del cual (en cadencia descendente) se desactivaría la placa (estado LOW) y significaría que el pie se encuentra en el aire.

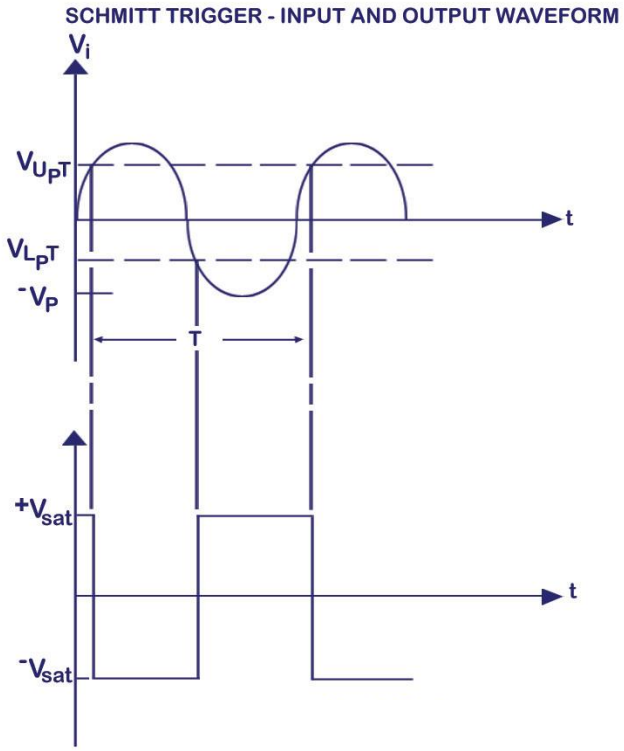

Figura 43: Funcionamiento disparador Schmitt

El código del *Schmitt* consiste en fijar unos límites de estados HIGH y LOW con los que se pretende fijar los límites en los cuales se entenderá que la placa de fuerza está siendo activada por el pie correspondiente o por el contrario, está liberada, y por tanto el pie se encuentra en el aire.

```
#ifndef SCHMITT_H
#define SCHMITT_H

enum status_t {LOW, HIGH};

// Schmitt trigger emulation functor
class Schmitt
{
private:
	status_t status_;
	double ht_, lt_;

public:
	Schmitt(const double ht, const double lt) : status_(LOW)
	{
		ht_ = ht; // Set HIGH transition value
		lt_ = lt; // Set LOW transition value
	}
```

```
        status_t operator()(const double input)
        {
                if (input > ht_ && status_ == LOW) status_ = HIGH;
                if (input < lt_ && status_ == HIGH) status_ = LOW;
                return status_;
        }

        void setLow(const double input)
        {
                lt_ = input;
        }

        void setHigh(const double input)
        {
                ht_ = input;
        }
};

#endif // SCHMITT_H
```

4.3.6. Modificaciones tras fase de pruebas

Las primeras pruebas del control codificado demostraron que el funcionamiento no era todo lo preciso como se deseaba. Comenzaba con cierto retraso y no conseguía alcanzar el ángulo que la pierna realiza en marcha normalmente.

A partir de estos resultados, se decide cambiar la programación del control, en la que las placas de fuerza (pisadas o no) tengan menos importancia a la hora de poner en marcha el funcionamiento del sistema. Primeramente, se añade una doble condición para el comienzo del control de las ortesis. Esta doble condición parte de calcular el centro de presiones durante la fase de apoyo del pie. Para tener un punto que posea cierta horizontalidad durante la marcha, se decide calcular el centro de presiones en la parte delantera del pie, punto proximal de la punta del pie.

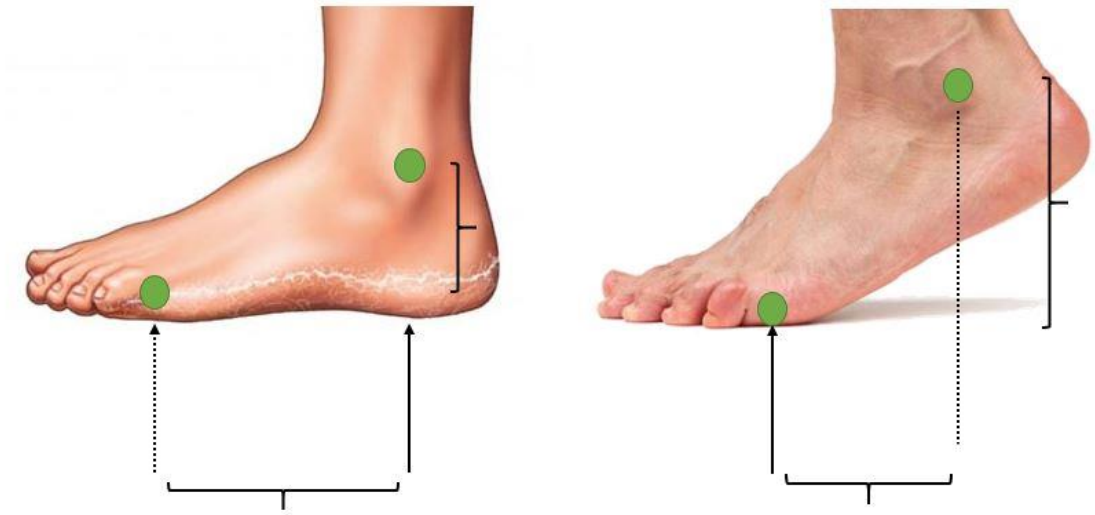

Figura 44: Centros de presión durante la marcha

Durante la marcha (Figura 44) se crea una fuerza sobre el suelo y su reacción sobre el pie del mismo valor y signo contrario. Además se crea un momento que será resultado de multiplicar esa fuerza vertical por la distancia al centro de presiones.

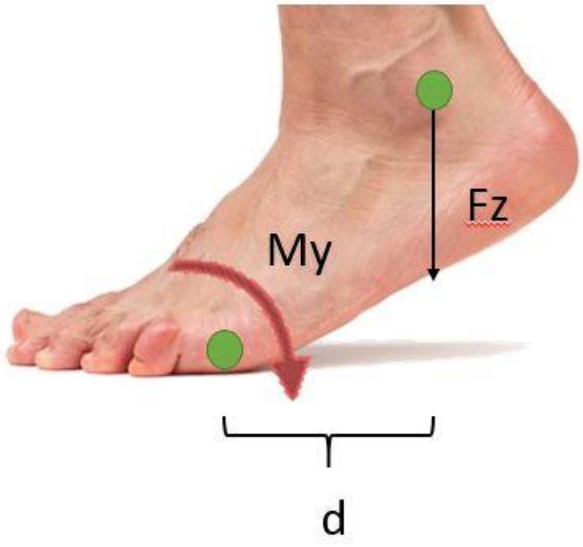

Figura 45: Fuerza y momento resultantes

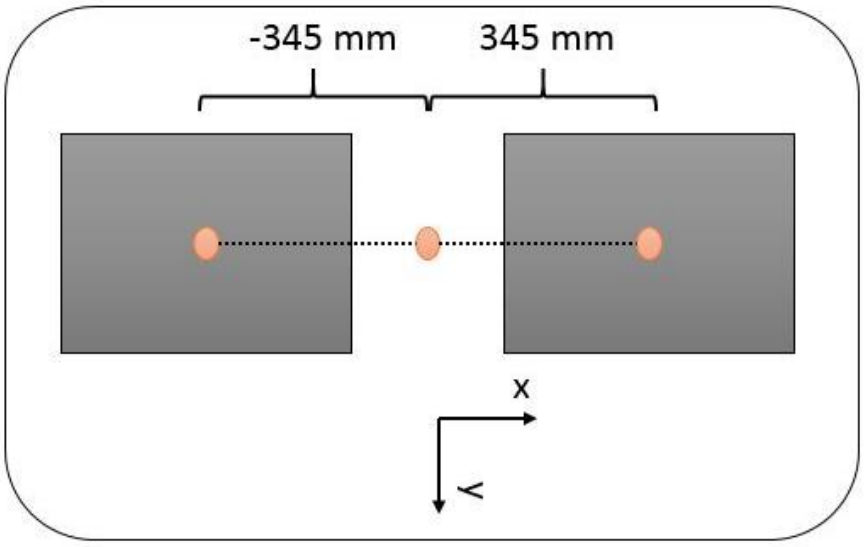

Figura 46: Distancias cálculo centro de presiones

Por tanto las coordenadas del centro de presiones de cada placa de fuerza ubicadas en el suelo del laboratorio de Ingeniería Mecánica en el que se realizan las capturas de movimiento será:

$$rc1 = \begin{bmatrix} -0{,}345 \\ 0 \\ 0 \end{bmatrix}$$

$$rc2 = \begin{bmatrix} 0{,}345 \\ 0 \\ 0 \end{bmatrix}$$

Finalmente, el cálculo en el eje X de los centros de presión de cada una de las placas serán:

$$X_{COP1} = -0{,}345 - \frac{My}{Fz1} \quad si\ Fz1 > umbral \tag{63}$$

$$X_{COP2} = 0{,}345 - \frac{My}{Fz2} \quad si\ Fz2 > umbral \tag{64}$$

En cuanto a las distancias que se fijarán como condiciones, debe ser la diferencia entre el centro de presiones de cada una de las placas y la coordenadas x correspondiente al tobillo. Si esta distancia es mayor que un umbral fijado, la placa estaría al aire (pie en el aire). En caso contrario, la placa estaría pisada por el pie que se está midiendo en ese momento.

Con esto, se pretende detectar si el talón está en el aire, aunque la punta del pie esté en contacto con el suelo. Este hecho emularía mejor al FSR ya que éste está situado en el talón

La placa FSR estaría en estado alto (ON) si y solo si se cumple la doble condición:

$$\delta \leq \delta^* \quad \&\& \quad Fz \geq Fz^* \tag{65}$$

El código quedará de la siguiente forma:

```
delta1 = rPoint_proxp [0] + 0.345 + (FPL1 [filas][4] / FPL1 [filas][2]);
delta2 = rPoint_proxp_i[0] - 0.345 + (FPL2[filas][4] / FPL2[filas][2]);


std::cout << std::endl << delta1 << std::endl;

if (delta1 >= umbral_delta && FPL1 [filas][2] >= umbral_fuerza)
	rFSR= HIGH;

else
	rFSR = LOW;

if (delta2 >= umbral_delta && FPL2 [filas][2] >= umbral_fuerza)
	lFSR = HIGH;
else
	lFSR= LOW;

if (FPL1 [filas][2] >= umbral_fuerza)
	std::cout << rFSR << lFSR << std::endl;
```

Sin embargo, como ya se había comentado, el resultado no es el que se esperaba, debido principalmente a que la acción de control resulta extremadamente dependiente de las placas de fuerza. Por lo que se decide realizar otro tipo de control, en el que intervengan solo las distancias en los centros de presiones.

Se decide, además, utilizar el ángulo de la tibia y no el de tobillo que se utilizaba hasta el momento. Este ángulo describe un mejor funcionamiento del control, si se demuestra que el error utilizando el ángulo de la tibia es menor que el utilizado anteriormente, se concluirá que la utilización de los sensores inerciales o IMUs será la mejor opción para este sistema. El ángulo de la tibia se estimará como el ángulo cuyo seno es la coordenada z del vector u representativo del hueso en cuestión, es decir:

```
theta_tibia_iz = asin (uVectort_i [2])
```
(66)

Para saber qué ángulo resulta óptimo, se programa una función objetivo de la que se minimizará el error entre la referencia y el ángulo calculado a lo largo de toda la captura de movimiento. Se exportarán los datos de los ángulos que interesan en el estudio de la simulación realizada en c++ y posteriormente se trabajará con estos datos en Matlab®, donde se utilizará una función propia del software, "ga", y se programará el control de las ortesis como función objetivo a minimizar. El código es el siguiente:

```
function x = Optimizacion_angulos

x0 = [0.8; -0.15; -0.45; 0.20]; %valores iniciales
x0 = [0; 0; -1; 1];             %valores iniciales
lb = [-2.0 -2.0 -1.0 -1.0];     %valores límite inferior
ub = [2.0 2.0 1.2 2.2];         %valores límite superior

figure (1)

options= gaoptimset('generations',2000,'populationSize',800,'TolFun',1e-
8,'TolCon',1e-8,'HybridFcn',{@fmincon},'PlotFcns',@gaplotbestf);

x = ga(@funcion_objetivo,4,[],[],[],[],lb,ub,[],options);


load('datos_angulos_plot.dat')

theta_tibia = datos_angulos_plot(:, 1);
theta_rodilla = datos_angulos_plot(:, 2);
theta_pie_iz = datos_angulos_plot(:,4);
theta_rodilla_control = Funcion_angulo(-theta_tibia-theta_pie_iz, 1.117,
x(1), x(2), x(3), x(4));

theta_rodilla_control = Funcion_angulo(-theta_tibia, 1.117, x(1), x(2),
x(3), x(4)); %Comprobación de IMU o encoder


figure (2)
plot(theta_rodilla,'r','LineWidth', 2)
hold on
plot(theta_tibia, 'k:')
plot(theta_rodilla_control,'LineWidth', 2)
plot(datos_angulos_plot(:,3),'g','linewidth',2)
hold off
title('Ángulos de rodilla de referencia, tibia iz y rodilla controlada')
legend('rodilla de referencia','tibia izquierda','rodilla controlada','ro-
dilla c++')
grid on
```

```
function theta_rodilla= Funcion_angulo(a,kr,ks,kw,ia,fa)

theta_rodilla=zeros(size(a));
w= pi/(fa-ia);

aon = a(a > ia & a < fa);

phase = ks*sin(w*(aon-ia)) + kw*sin(2*w*(aon-ia));

theta_rodilla(a > ia & a < fa) = kr*(sin(w*(aon-ia) - phase));



function f =funcion_objetivo(x)

load('datos_angulos_plot.dat')

frames = 80:130;

for i=frames

theta_tibia(i) = datos_angulos_plot(i, 1);
theta_pie_iz(i) = datos_angulos_plot(i,4);

theta_control(i) = Funcion_angulo(-theta_tibia(i)-theta_pie_iz(i), 1.117,
x(1), x(2), x(3), x(4));

theta_referencia(i) = datos_angulos_plot(i,2);

end

%La minimización del error interesará entre los frames 80 y 125, donde se
%alcanza el ángulo máximo de marcha

sumatorio= sum((theta_control(frames)-theta_referencia(frames)).^2);

f=sqrt(sumatorio/length(frames)); %125-80 son el rango de frames donde in-
teresa el cálculo
```

Con esta función se obtienen los valores óptimos que debemos introducir al control para que sea lo más efectivo posible y parecido a la marcha humana de una persona sana. Estos parámetros son los que se han denominado bajo las siglas kr, ks, kw, ia y fa, y corresponden, respectivamente, al máximo ángulo que alcanza la rodilla durante la marcha, la desviación de la curva en el eje del tiempo y su ensanchamiento o estrechamiento, y finalmente los ángulos inicial y final de elemento que se está introduciendo como input (el ángulo que describe).

En las siguientes gráficas se puede ver el error que se comete cuando el input es el ángulo de la tibia y cuando es la diferencia entre el ángulo de la tibia y el que del pie de la pierna en movimiento.

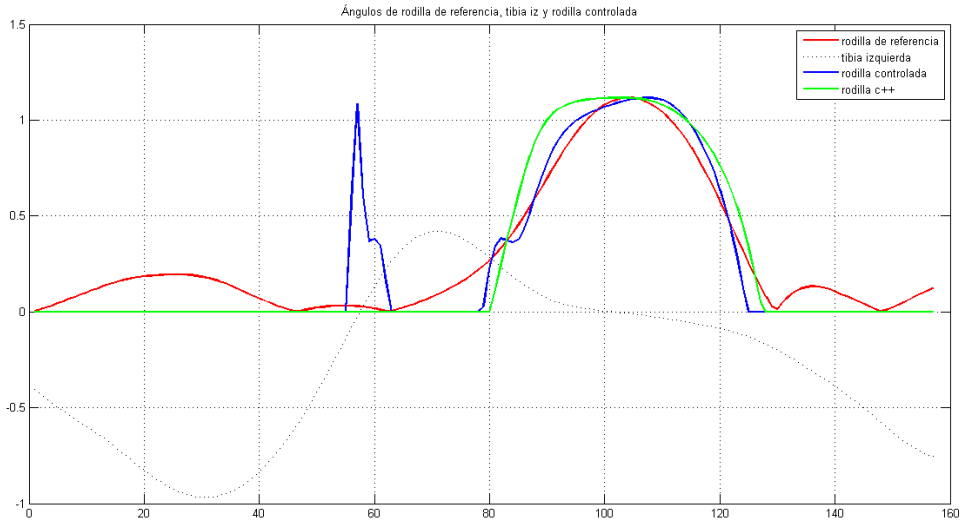

Figura 47: Optimización con diferencia de ángulos

Los errores máximos y mejor optimizados con la diferencia entre el ángulo descrito por la tibia y el descrito por el pie serán:

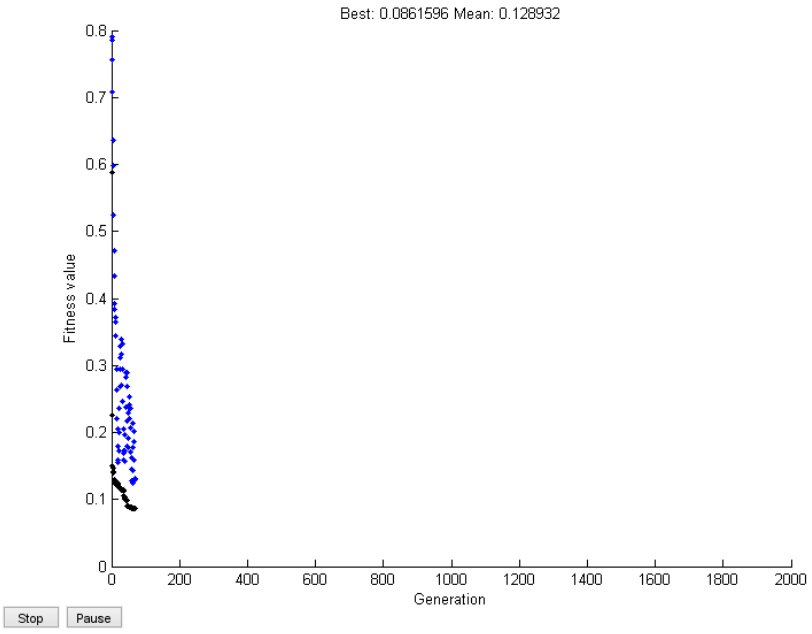

Figura 48: Generación de optimización diferencia de ángulos

Donde el error máximo ronda los 0,68 radianes y se establece en un valor óptimo de 0,09.

Cuando se realiza la misma optimización solamente con el ángulo descrito por la tibia se obtienen los siguientes resultados:

El error máximo no alcanza los 0,38 rad mientras que el mejor valor del ángulo descrito por la tibia será de alrededor de 0,08 rad.

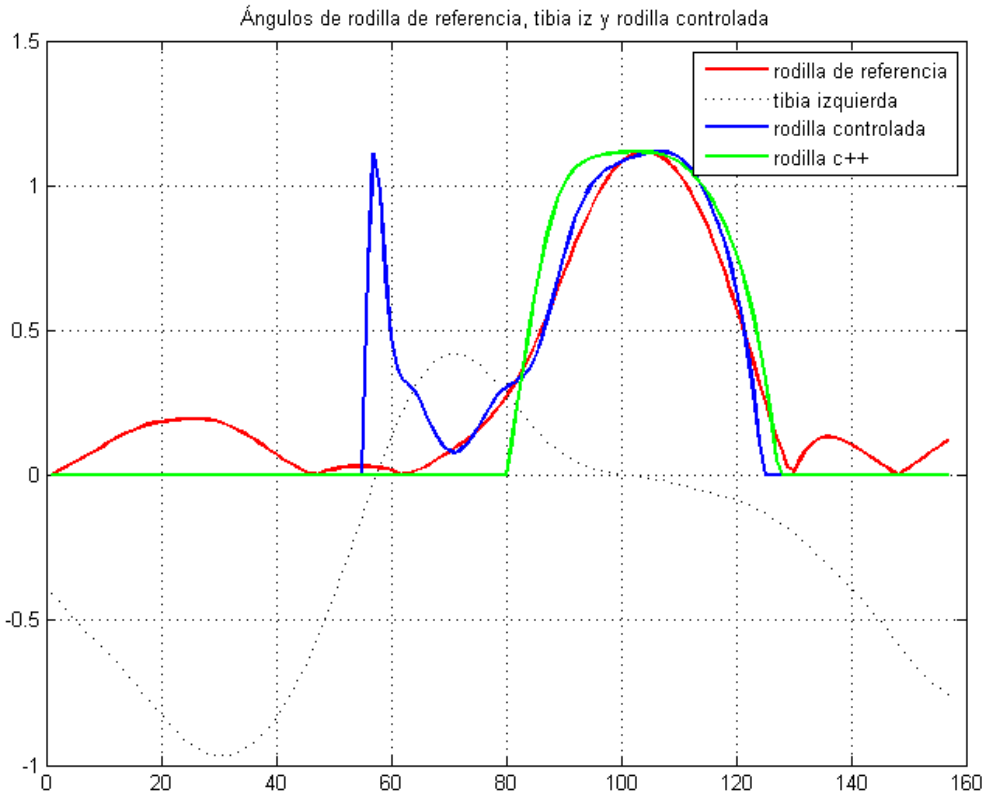

Figura 49: Optimización con ángulo de tibia

El error calculado durante la optimización se representa en la siguiente gráfica:

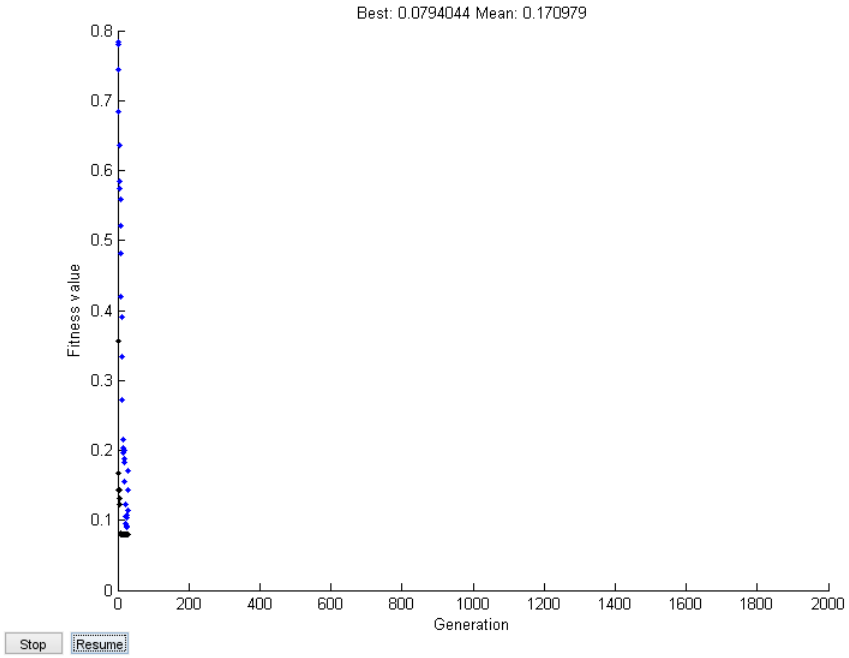

Figura 50: Error en la optimización con ángulo de tibia

Dado que el error calculado utilizando únicamente el ángulo de la tibia es bastante inferior al calculado con la diferencia entre el ángulo descrito por la tibia y el descrito por el pie, se concluye que la utilización de un sensor inercial tipo IMU es lo más conveniente para esta aplicación. Se colocaría de tal forma que midiese durante el movimiento el ángulo de la tibia y daría como resultado un control optimizado mucho más próximo a la marcha humana correcta.

Con esta demostración, tan solo queda explicar de dónde surge la idea de utilizar el ángulo de la tibia. Al plotear los ángulos que se utilizaban primeramente, se podía comprobar cómo los ángulos poseen demasiado ruido y cambian a lo largo del tiempo sustancialmente. Sin embargo, si se plotea el ángulo de la tibia y del pie, en vez de las articulaciones, se comprueba cómo poseen una evolución más lineal y progresiva.

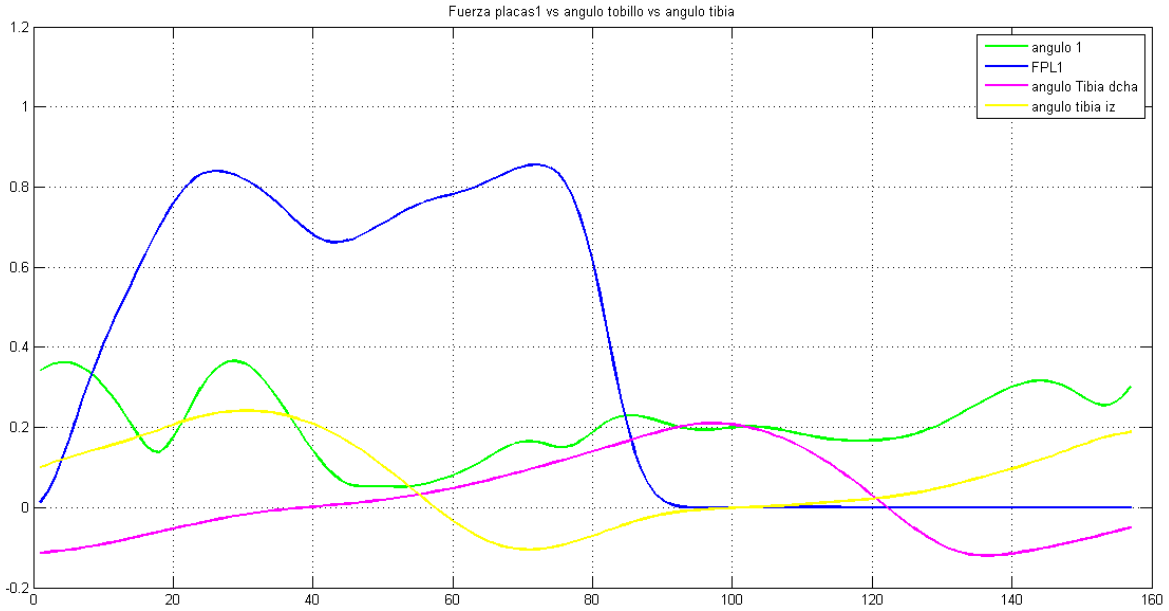

Figura 51: Evolución de ángulos de control

La línea verde representa el ángulo de la rodilla y la magenta y la amarilla representan los ángulos de la tibia derecha e izquierda respectivamente. Se puede probar la evolución progresiva y lineal de la que se hablaba anteriormente, comparado con la inestable evolución de la línea verde.

En la siguiente figura se observa la evolución del ángulo descrito por el pie derecho en relación con las fuerzas obtenidas de las placas de fuerza relativas a dicha pierna. Esto es, se describe la evolución del ángulo del pie derecho cuando esa misma pierna está en contacto con el suelo o en el aire.

Se puede ver cómo la evolución parece razonable de acuerdo con la marcha humana característica.

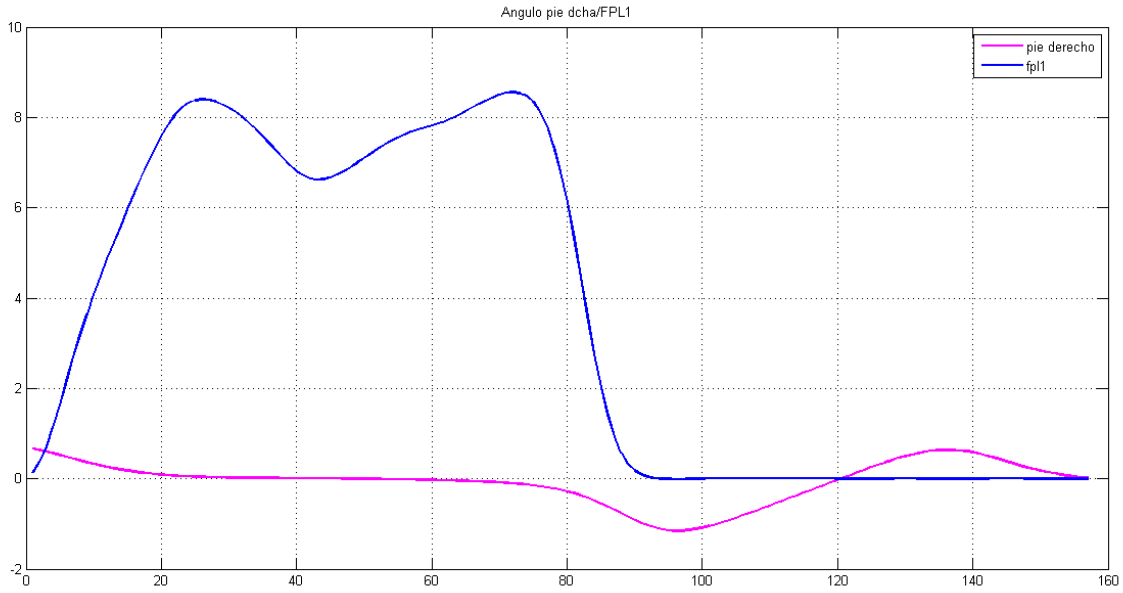

Figura 52: Evolución ángulo pie derecho con FPL1

Finalmente se decide, para mejorar el control aplicado a la marcha humana, evitar la utilización de las placas de fuerza a la hora de analizar el estado del movimiento durante el movimiento de marcha. Se utilizarán las distancias verticales (eje Z) del tobillo al suelo durante este movimiento.

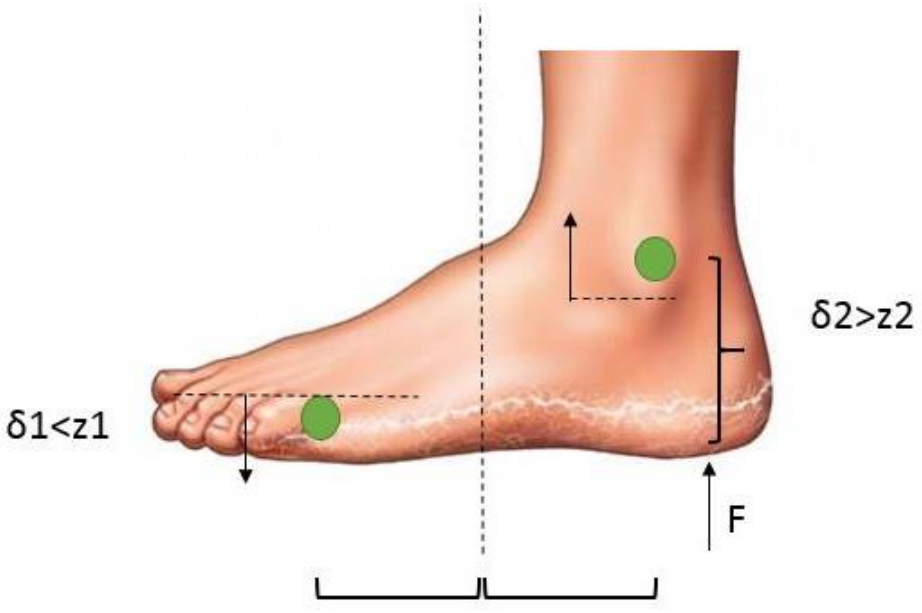

Figura 53: Nuevo control de ortesis

Se aplicará este control a la rodilla contraria al pie que está en contacto con el suelo, sabiendo si está en estado alto o bajo solo con las distancias del talón al suelo del pie contrario y añadiendo el ángulo que describe la tibia de la pierna en contacto con el suelo.

```
delta1 = rPoint_proxtb [2]; //coordendada z del punto

delta1_iz = rPoint_proxtb_i [2];

if (delta1 >= 0.115 )
       rFSR = LOW;
else
       rFSR = HIGH;

if (delta1_iz >= 0.11)
       lFSR = LOW;
else
       lFSR = HIGH;


theta_rodilla_dcha = rMotorControl (-theta_tibia_iz, rFSR, lFSR);
```

Habiendo introducido los valores óptimos de los parámetros del control en la clase Control, obtenidos de la optimización realizada en Matlab®.

```
double kr = 1.117,           // Knee rotation range (degrees). Now radians
double ks = 1.2035,          // Peak translation (0 = symmetric)
double kw = -0.3262,        // Peak width (0 = normal)
double ia = -0.4515,  // Initial angle w.r.t. neutral position (degrees). Now radians
double fa = 0.1261,       // Final angle w.r.t neutral position (degrees). Now radians
```

4.4. Capturas de movimiento

Se realizan capturas de movimiento para la realización de pruebas del software codificado para la realización del control de la ortesis objeto del presente Trabajo Fin de Máster. Las primeras capturas se han realizado para obtener resultados comparativos del funcionamiento de la IMU con respecto al encoder que se encuentra instalado en la articulación del tobillo de la ortesis. Lo que se desea, es comprobar si el cambio que se pretende realizar entre los sensores, puede llevar a un resultado mejor o al menos no modificar la adquisición de datos que se tenía anteriormente. En pruebas posteriores, se comprobará la validez de la programación realizada con diferentes inputs al control, utilizando, también, la simulación que se ha realizado para la comprobación de posibles cambios en el control.

4.4.1. Bases de datos para pruebas

Se realizan una serie de pruebas para la obtención de resultados del código programado y de la funcionalidad conjunta del equipo. Con estas pruebas se obtendrán resultados del funcionamiento de la ortesis con los cambios realizados en el presente Trabajo Fin de Máster.

Prueba IMU-encoder

La primera prueba se ha realizado en base a la comparativa entre sensores. Se ha instalado la IMU de manera que el ángulo medido sea el que se encuentra perpendicularmente alineado con la tibia (en este caso pitch, aunque siempre depende de la orientación de la IMU) y además formando 90˚ con el eje del encoder instalado en la ortesis en el plano longitudinal de la tibia. En esta primera fase de pruebas se pretende conocer la simultaneidad existente en el funcionamiento de ambos sensores.

Se realizan los cambios oportunos en el código para obtener los datos del encoder, con una resolución de 4000 pulsos por revolución y conectado a una tarjeta de adquisición del tipo NI DAQ 6212, de manera que la lectura del ángulo que describe el tobillo y que lee el encoder sea inmediata a través de la ventana de comandos.

A la vez, con este mismo código se realiza la lectura de la IMU, colocada estratégicamente para que mida el mismo ángulo que el encoder, siempre con ligeras modificaciones. No se debe olvidar que se está en una fase de pruebas, por lo que la colocación de la IMU, en este caso, no es definitiva. La medición de datos de la IMU se extrapola al cuaternión que describe. Ambos datos se exportan a archivos *.dat para su posterior lectura en Matlab®. De esta forma se consigue plotear los resultados a través de un código de este último software.

Una vez obtenidos los datos, en Matlab se transforma el cuaternión en los ángulos que mide la IMU: yaw, pitch, roll. El ángulo que interesa en este momento es el de pitch, y por tanto será el ploteado junto con el ángulo medido por el encoder.

A continuación se muestran las gráficas que representan ambos ángulos durante la fase de movimiento de la ortesis, primeramente con movimientos alternativos y a continuación con un movimiento semejante al de marcha.

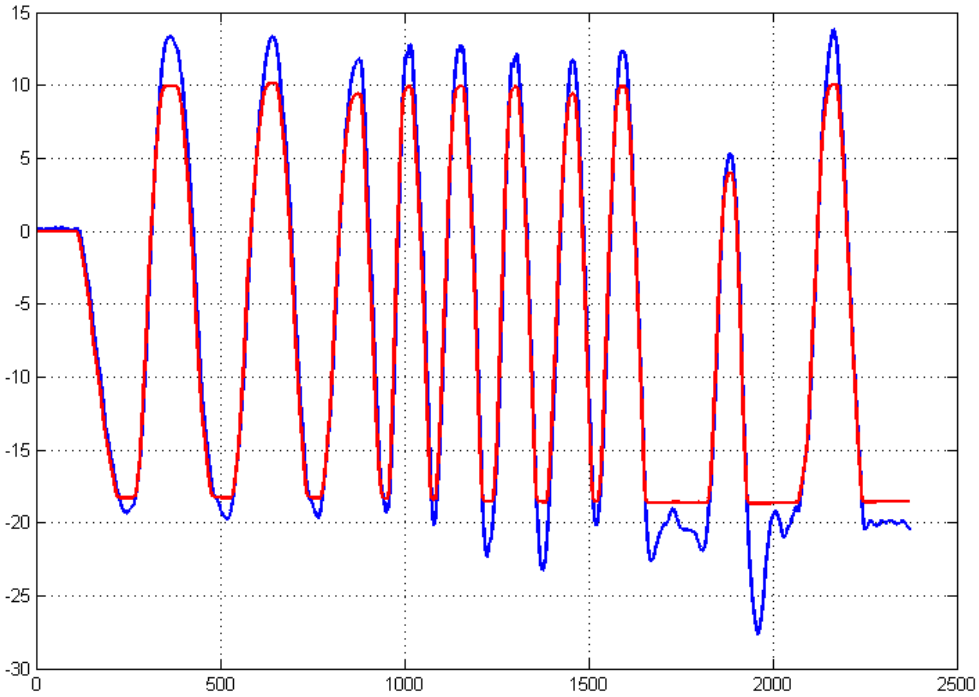

Figura 54: Primera prueba. Movimiento alternativo

En la gráfica anterior, donde el **encoder** está representado por una línea **roja** y la **IMU** por una línea de color **azul**, se puede comprobar como la lectura del ángulo descrito en el movimiento es bastante similar en ambos sensores. Se trata de una gran ventaja a la hora de poder realizar los cambios que se pretendían. Las diferencias de amplitud entre uno y otro se deben principalmente al movimiento que se le ha impartido para obtener los datos. Llega un momento en que el encoder satura, dado que el tobillo no está sufriendo ninguna modificación de giro debido a que la articulación tiene un tope mecánico, mientras que la colocación de la IMU permite una mayor flexibilidad, llegando a obtener ángulos ligeramente mayores.

Una segunda prueba se realiza mediante la imitación del movimiento de marcha. Los cambios angulares que se pueden observar son los que se producen en el tobillo durante la marcha humana.

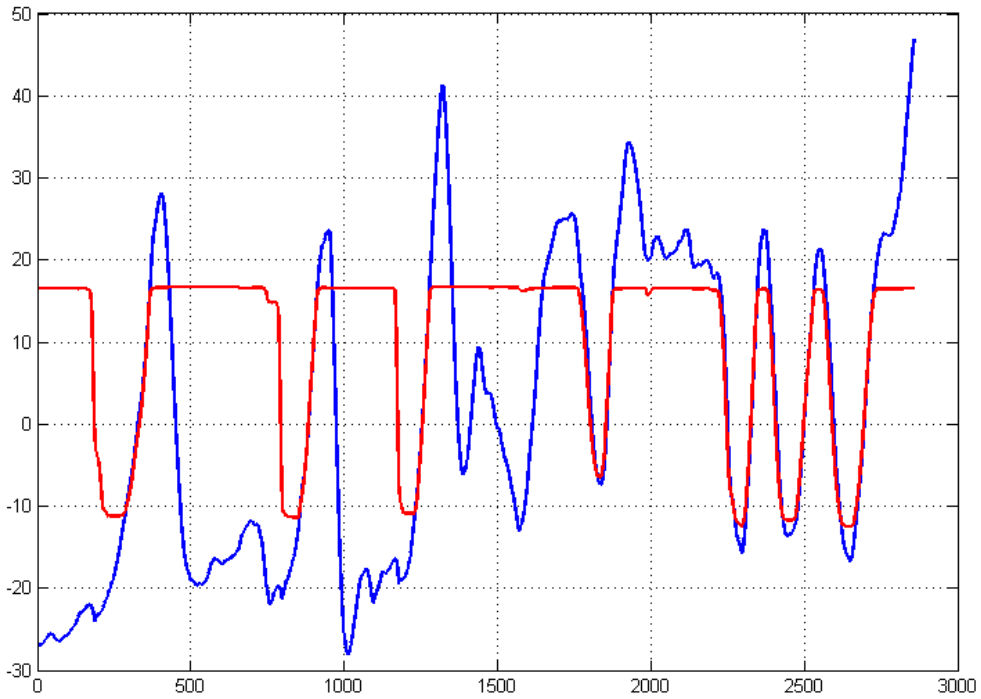

Figura 55: Segunda prueba. Movimiento de marcha

En la Figura 55, se puede observar cómo el error entre ambos sensores en cuanto al movimiento del tobillo, que sería el momento en que ambas curvas ascienden, es muy similar, con un error sorprendentemente pequeño.

Las curvas coinciden cuando el pie está horizontal, de modo que el ángulo absoluto de la tibia y el relativo al pie son iguales

Si se procede a una instalación menos temporal de la IMU, se debe alinear lo más perfectamente posible los ejes, haciendo que la medición sea mucho más precisa de lo que se ha ploteado, y obteniendo unos resultados más próximos a los deseados.

4.5. Otras posibles estrategias de control

Existen otras posibilidades de control, dependiendo de los inputs que se le deseen introducir al mismo. Estas diferentes estrategias son ahora posibles gracias a haber probado que la utilización de IMUs resulta más ventajosa que los sensores utilizados hasta el momento. Por ello, con la facilidad de montaje que deriva del uso de IMUs, se pueden colocar en diferentes partes del tronco inferior, para la medición de los ángulos de marcha. Además, con el simulador que se ha implementado en c++ sería más fácil comprobar las diferentes estrategias, realizando capturas de movimiento con los sensores colocados en los puntos de interés sobre la ortesis.

Una de las estrategias que parecen más ventajosas, sería medir el ángulo del fémur y de la tibia, y conseguir controlar su velocidad, para obtener una señal de menor ruido del mismo modo que se

hizo con los ángulos de rodilla y tibia utilizados anteriormente. Realizando estos cálculos en Matlab® se obtienen los siguientes resultados:

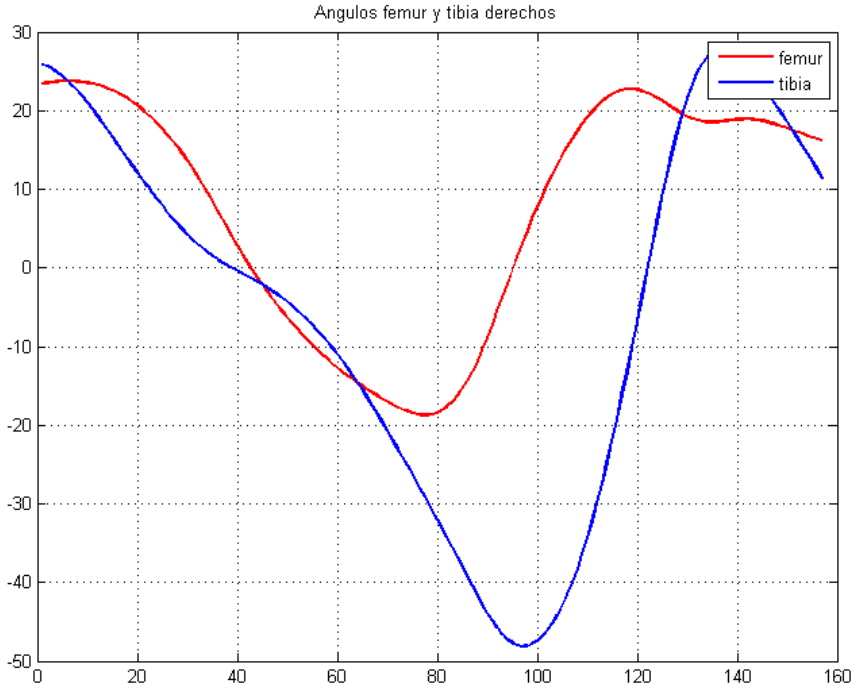

Figura 56: Ángulos fémur y tibia

Y por tanto, si se representa la derivada de estos ángulos se obtiene:

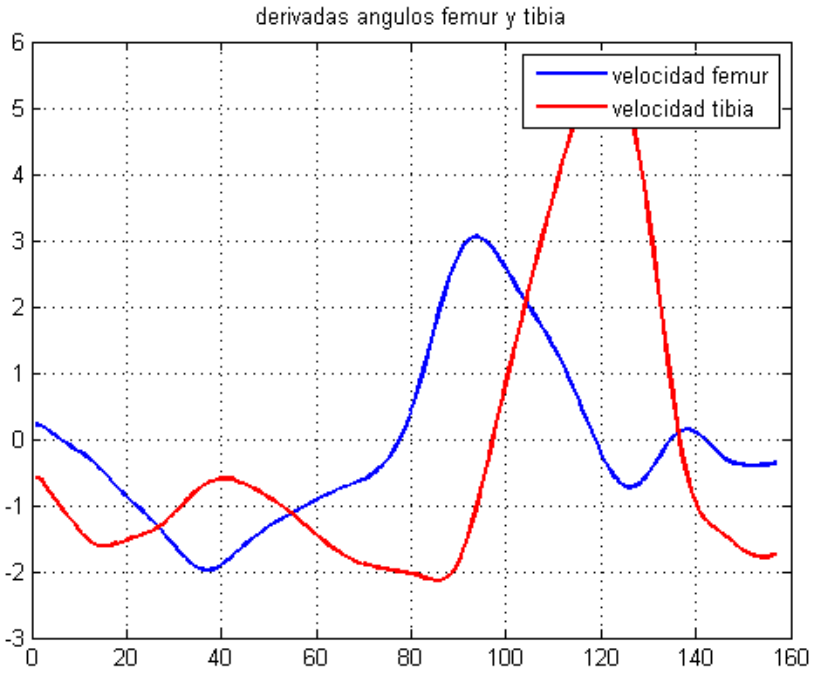

Figura 57 Derivada angular de fémur y tibia

Se debe señalar que no se han probado sobre las ortesis físicas estos inputs de control debido, en gran parte, a la falta de tiempo para realizarlo, y por otro lado, a la fase de pruebas con la paciente en cuestión a la que se está sometiendo al dispositivo, y por tanto, la imposibilidad de realizarle grandes cambios al mismo.

5. Test y resultados

A lo largo del presente documento se han ido mostrando los diferentes resultados obtenidos a partir de las pruebas realizadas sobre el control programado. Se han llevado a cabo distintos casos combinando ángulos de entrada, simulando sensores, e incluso modificando la forma en la que los diferentes agentes externos puedan afectar y en qué medida al control aplicado a la ortesis y al paciente.

5.1. Protocolo de test e indicadores cuantitativos

Cada una de las fases de test se ha llevado a cabo sobre el software de simulación creado en el presente Trabajo Fin de Máster. Este simulador es una herramienta de gran potencial. Su mayor ventaja es que evita la imperiosa necesidad que existía hasta el momento de realizar todas las pruebas, por pequeño que fuese el cambio o modificación sobre el control, la ortesis o cualquier otro elemento del sistema, sobre el paciente en cuestión. Este hecho implicaba a su vez cierto riesgo dado que no se conocía el efecto que el cambio podía tener sobre el conjunto o el control del sistema.

Ahora, con esta herramienta de simulación, se podrá experimentar todo tipo de cambios sobre el sistema y probar de forma virtual el efecto que tendrían sobre la marcha del paciente que haya realizado la captura de movimiento (del que se dispongan los datos).

Además de una herramienta visual, se han programado diferentes funciones en Matlab® que ofrecen gráficas sobre la estimación de ciertos inputs al control así como la relación entre distintos ángulos y el paso (durante la marcha) del paciente en cuestión.

5.2. Fase de pruebas con ortesis original

A lo largo de esta fase en la que se encuentra el proyecto original de investigación se han realizado pruebas con la paciente voluntaria. En colaboración con los miembros de la facultad de INEF (A Coruña) que participan en el proyecto, se han realizado una serie de pruebas consistentes en ejercicios adaptados a la paciente para fortalecer el tronco inferior afectado por la lesión medular además de las pruebas pertinentes con las ortesis.

En cuanto a las pruebas realizadas con las ortesis, se han basado en el cambio de sensores. La evolución de los sensores ha partido de la utilización de encoders situados en el tobillo para medir el ángulo relativo entre la tibia y el pie. Estos encoders tienen una alta resolución, que en principio no sería necesaria para esta aplicación. Existe otro gran inconveniente, y es que han dejado de fabricarse, por lo que la solución para seguir utilizando estos sensores sería buscar unos de parecidas prestaciones. Dado que se tendría que modificar la disposición física sobre la ortesis, se decide proba con otro tipo de dispositivos: los potenciómetros de medida. Poseen la precisión y exactitud suficientes para la aplicación, y estarían situados en la misma posición sobre la ortesis.

Se decide sin embargo, establecer una investigación y fase de experimentación virtual sobre otro tipo de sensores, que de verificar su buen funcionamiento sobre el resto de sensores, reducirían tanto el presupuesto como el peso de la ortesis. Se trata de las IMUs. En el estudio que se ha expuesto en apartados anteriores, se publican los resultados de prueba comparando el funcionamiento

de las IMUs respecto a los encoders originales. Además, queda totalmente reflejado en la optimización de los parámetros de control, que el error que se comete utilizando las IMUs es bastante inferior al cometido utilizando los encoders, únicamente basándose en la situación de cada uno de los sensores.

5.3. Fase de pruebas control mejorado

Las pruebas con los cambios realizados sobre el control se han realizado de forma virtual sobre el simulador creado en el presente Trabajo Fin de Máster. Se debe tener en cuenta que la experimentación real con la paciente colaboradora en el proyecto de investigación está sujeta a un proceso evolutivo más lento. Por un lado, se depende de la disponibilidad del paciente, y por otro, no se pueden realizar un número excesivo de pruebas en la misma sesión, dado que se produciría una fatiga sobre la paciente poco recomendable.

En posteriores pruebas, se introducirán las IMUs en las ortesis para probar su funcionamiento, además de la portabilidad a la BeagleBone Black®, que reducirá enormemente el peso del conjunto hardware y permitirá las mismas prestaciones de comunicación que la tarjeta de adquisición de datos y el computador anteriormente utilizados.

6. Direcciones futuras: control de velocidad

Se realiza una nueva captura de movimiento en la que se utilizan las placas de fuerza para medir la presión de la pisada sobre las mismas.

Con esta captura de movimiento se pretende realizar un nuevo control, o al menos encaminar el proyecto hacia direcciones futuras. Este control será un control de velocidad por lo que los inputs al mismo serán las velocidades sobre la marcha o movimiento que esté realizando el paciente.

Para ello, la captura de movimiento está compuesta por un compendio de marcha normal más levantamiento de piernas laterales manteniendo la rigidez de la rodilla y un segundo movimiento de flexión de la articulación de la rodilla.

Con estos movimientos, se realizará un cambio en el control, adaptándolo a la nueva captura de movimiento y sus matrices representativas.

Una de las fases de mejora será detectar el tipo de movimiento para aplicar o no el control a la pierna contraria, es decir, se desea conocer el estado de movimiento: marcha o levantamiento de pie. Para ello, se crea un temporizador en el código de lenguaje c++, en el momento en que uno de los pies deja de estar en contacto con el suelo. De esta forma se medirá, bien el tiempo que tarda la persona en dar un paso o bien si sigue en el aire el mismo pie. Así se realizará la distinción entre ambos tipos de movimiento a estudiar.

Al control se le añade una restricción de velocidad para que el movimiento sea de SWING únicamente si la velocidad medida es superior a un umbral fijado, además de unas condiciones inicial y final de velocidad que se incluirán como parámetros del nuevo control.

Una fase posterior es la conversión del control de posición a un control de velocidad. Se pretende medir el ángulo máximo y mínimo descrito por el tobillo, o la tibia, dependiendo del control que se prefiera realizar. Se medirá, con un temporizador, el tiempo que tarda en realizar el movimiento. A partir de estos datos se calcula la velocidad media como el cociente entre la amplitud (diferencia de ángulos) y el tiempo que tarda en cambiar a ciclo STANCE desde el modo SWING. Por tanto, lo que se realizará será una corrección de los ángulos iniciales y finales, expresándolos como un porcentaje del intervalo medido.

Para igual tiempo, si se aumenta la amplitud (diferencia de ángulos máximo y mínimo), la velocidad aumenta. Para igual amplitud, si se disminuye el tiempo, la velocidad aumentará de igual modo. El control se basará en estas dos premisas.

Al cambiar a un control de velocidad, se debe modificar la programación de la clase Control, de tal manera que para cierto ángulo de entrada, que seguirá siendo el ángulo descrito por la tibia, se calculará la velocidad asociada al ángulo de salida que anteriormente se utilizaba para el control de posición. En el bucle principal se aplicará el control únicamente cuando la temporización esté en proceso (hasta los tres segundos que tarda en dar un paso completo). Esta temporización, comienza cuando uno de los pies se pone en contacto con el suelo y termina cuando es el pie contrario el que está en contacto con el suelo. Se utiliza la clase de c++ *time.h* para llevarla a cabo.

La salida ahora, también habrá que modificarla para que igualmente sea un ángulo pero con la velocidad asociada calculada para una determinada entrada en el control.

```
switch (mode)
{
	case SWING:

	// Check whether the state is compatible with walking

	if (ownSt == HIGH)		//if foot is on the floor
	{
		mode = STANCE;
		motor_.gotoPosition(0);
		outAngle = 0;   //angulo de la rodilla contraria será 0
		break;
	}

	if (control_output [0] > 0)
	{
		motor_.gotoPosition (motorAngle (oppAnk));
		outAngle = motorAngle (oppAnk)*((100 / (2 * pi * 200) + 1)/200); //a la
rodilla contraria se le pasa como entrada el ángulo del tobillo que pisa

	}
	break;
```

Y en el bucle principal,

```
clock_t start;  //variables para crear temporización en segundos
float temporizacion;

start = clock (); //se inicia la temporización

temporizacion = (std::clock () - start) / CLOCKS_PER_SEC; //se detiene y se convierte
en segundos, unidad temporal elegida para la temporización

cout << "temporizacion en segundos: " << temporizacion << endl; //TARDA 3segundos en
dar un paso completo

if (temporizacion >= 1 && temporizacion <3)
{
	theta_rodilla_dcha = rMotorControl (-theta_tibia_iz, rFSR, lFSR);
}
```

7. Migración control a BeagleBone® Black

Se desea mejorar la portabilidad del conjunto total de la ortesis para poder disminuir el peso que los usuarios han de soportar. Por ello, en lugar del computador portable del que se disponía en un principio, se ha decidido la utilización de un computador embebido de la marca BeagleBone®, en concreto, el modelo Black. Con esta decisión se reduce significativamente el peso total del conjunto manteniendo la utilización de puertos de E/S, conexión WiFi y la facilidad de entendimiento con el lenguaje de programación elegido en la realización del software: c++. En este caso, se ha decidido utilizar el software comercial QtCreator® que permite una excelente comunicación con el dispositivo bajo el sistema operativo Ubuntu® (depuración con Linux®).

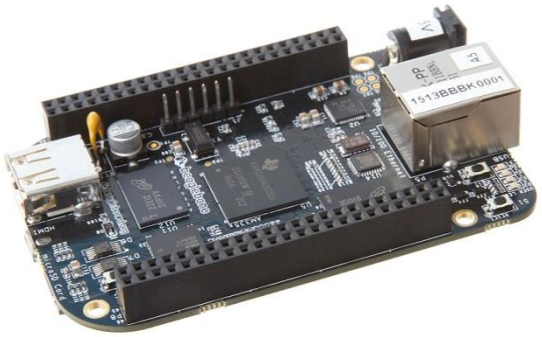

Figura 58: BeagleBone® Black

7.1. BeagleBone® Black

La BeagleBone® Black es el último miembro de la familia BeagleBoard®. Es un dispositivo de bajo coste que cuenta con un procesador A8 ARM de Texas® Instruments®. Posee características similares a la BeagleBone® clásica, sin embargo, se han mejorado ciertos aspectos y se han eliminado otros que se han considerados superfluos. Las principales diferencias se pueden ver en la siguiente tabla:

Tabla 3: Comparativa BeagleBone

	BeagleBone Black $55	BeagleBone $89
Processor	AM3358BZCZ100, 1GHZ	AM3359ZCZ72, 720MHz
Video Out	HDMI	None
DRAM	512MB DDR3L 800MHZ	256MB DDR2 400MHz
Flash	4GB eMMC, uSD	uSD
Onboard JTAG	Optional	Yes, over USB
Serial	Header	Via USB
PWR Exp Header	No	Yes
Power	210-460 mA@5V	300-500 mA@5V

A continuación se muestran las características principales de la BeagleBone® Black.

Tabla 4: Características principales BeagleBone® Black

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

7.1.1. Conectores

Los conectores disponibles en la placa son los siguientes:

- Alimentación DC: acepta 5V
- Botón de encendido
- Red Ethernet 10/100 en conexión a LAN
- Depurador por puerto serie
- Conexión miniUSB para conectar al PC
- Ranura tarjeta microSD

- Cuatro LEDs de color azul que pueden ser utilizados por el usuario
- Botón de RESET del procesador
- Conector microHDMI
- USB Host: diferentes interfaces como WiFi, BT, teclado, etc.

7.1.2. Especificaciones de alto nivel

A continuación se muestra el diagrama de bloques de alto nivel para la BeagleBone® Black.

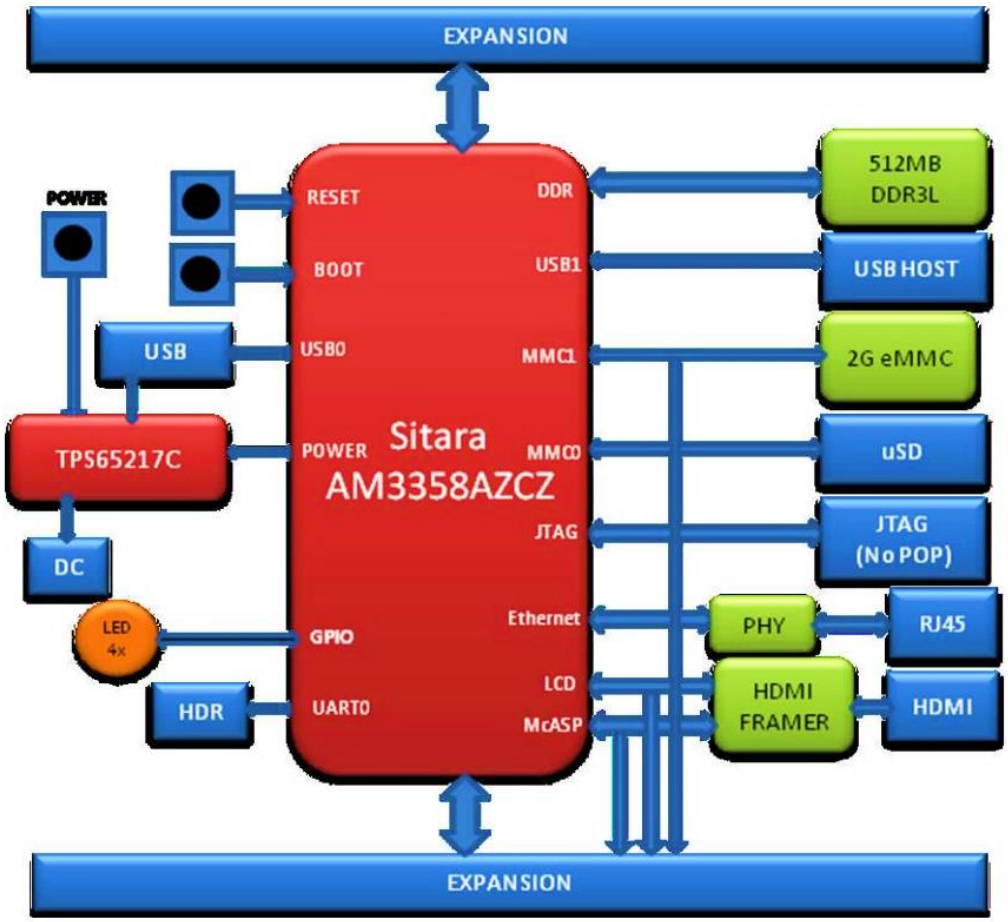

Figura 59: Diagrama de bloques BBblack®

Utiliza un procesador Sitara® AM3358AZCZ empaquetado en un bloque de 15x15 mm, operando a 1GHz de RTC. Además posee una memoria de 256 Mb x16 DDR3L 4Gb (512 Mb). Opera a una frecuencia de reloj de 303 MHz con una frecuencia efectiva de 606 MHz en el bus DDR3L permitiendo un ancho de banda de 1,32 Gb/s.

Cuenta con una memoria EEPROM de 32Kb con conexión I2C0 que contiene la información de la placa (nombre, número de serie, información de revisiones, etc.). Un dispositivo de memoria embebida de 2Gb de capacidad (MMC) que conecta con el puerto MMC1 del procesador, permitiendo un acceso de 8bits.

Finalmente, cuenta con la memoria de la tarjeta microSD.

Se debe tener especial precaución con los niveles de tensión de entrada y salida que soporta el dispositivo. A continuación se muestra el esquema conexiones para el TPS65217.

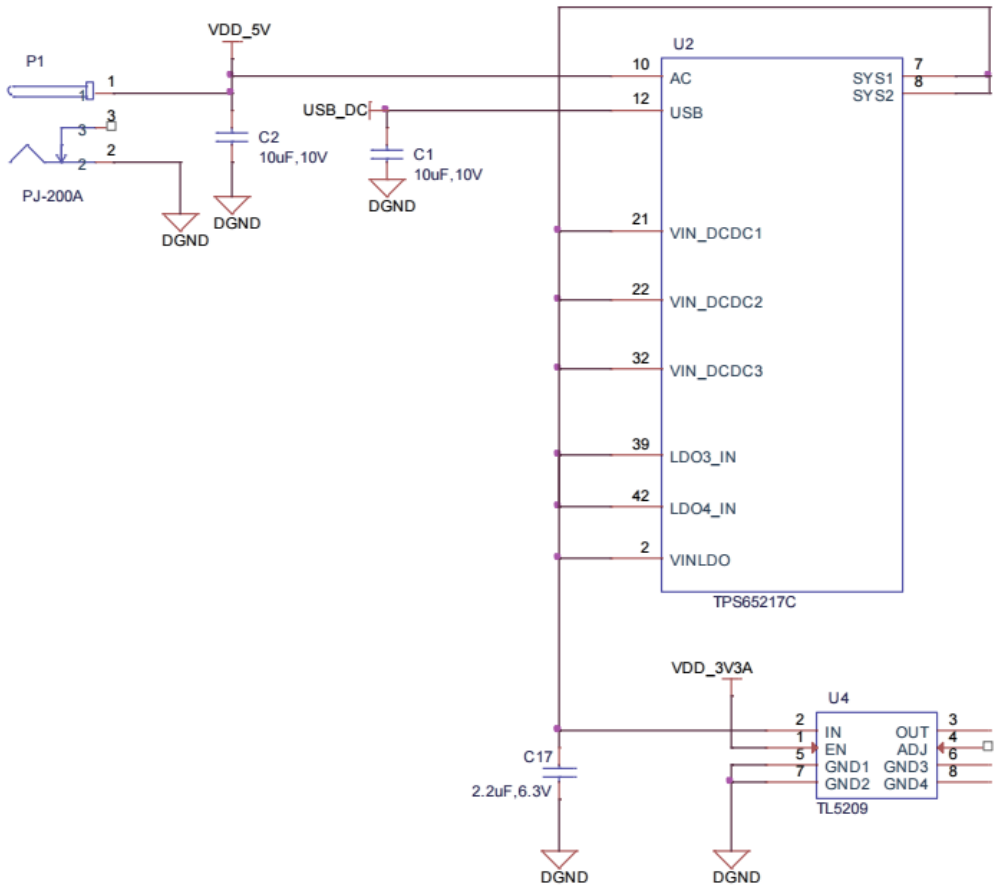

Figura 60: Niveles de tensión E/S

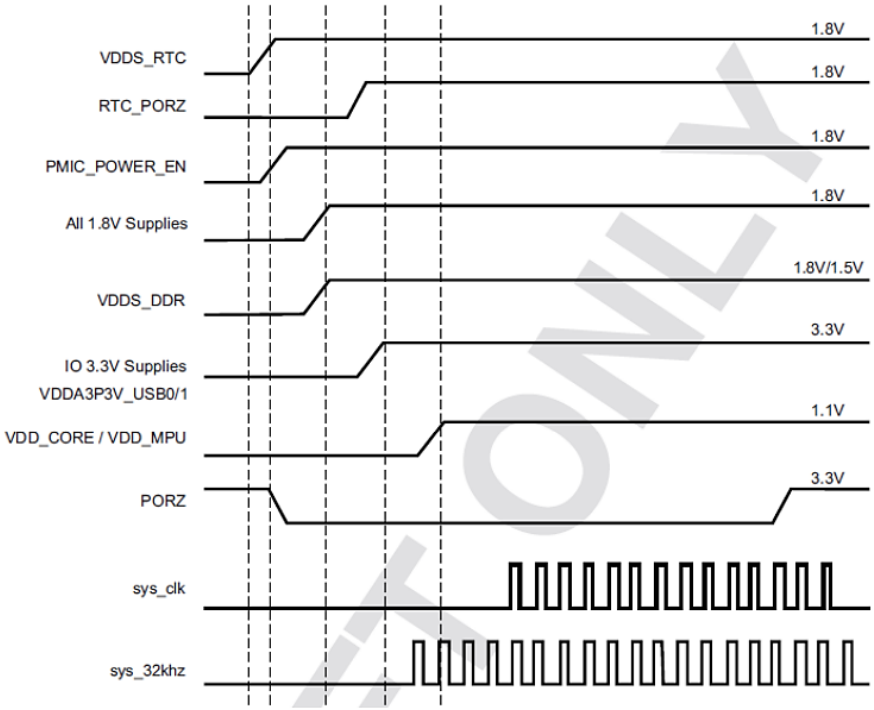

Figura 61: Salidas respecto a diferentes combinaciones de entrada

Si el usuario desea utilizar los leds proporcionados para determinados avisos luminosos de referencia, debe tener en cuenta el siguiente esquema de conexionado:

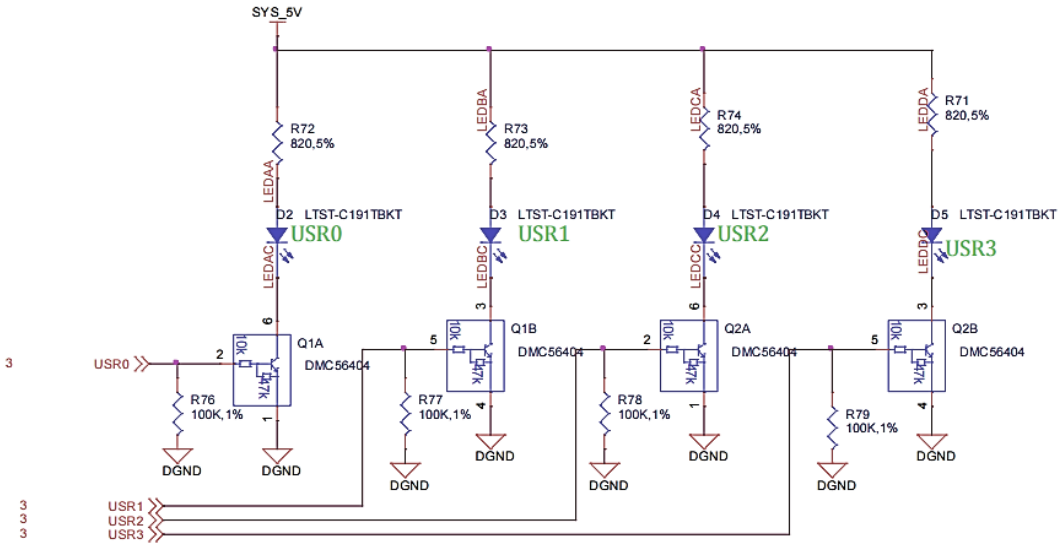

Figura 62: Conexionado LEDs

7.2. Modificaciones software

Se realizan las modificaciones oportunas, manteniendo lenguaje c++, obteniendo la equivalencia con el software QtCreator® que permite, sin realizar grandes cambios, el paso del mismo código de un sistema operativo a otro, en este caso, de Windows® a Linux® Ubuntu®. De esta forma, el código realizado para Windows® se puede pasar fácilmente a Ubuntu® manteniendo la misma estructura y proveyendo la posibilidad de comunicación con la BeagleBone® Black.

En el apartado de anexos, se adjunta el código de programación realizado.

7.3. Modificaciones hardware

La modificación hardware más relevante es el espacio utilizado por la BeagleBone® Black. Es altamente notorio su reducido volumen y su permisividad para realizar las mismas funciones que anteriormente se realizaban con un ordenador portátil de reducidas dimensiones pero peso considerable. Además, permite la sustitución de la tarjeta de adquisición de datos de National® Instruments®. Por tanto, se facilita a los pacientes el transporte del conjunto completo, debiendo mantener durante la marcha un peso mucho más reducido con las mismas ventajas software, sin eliminar ninguna funcionalidad anteriormente aplicada a la ortesis.

Otra ventaja que se entrelaza con las modificaciones software, es la falta de necesidad de comunicación con sistema operativo Windows® lo que reduce enormemente la complejidad del código y los posibles fallos de incompatibilidades entre ordenador embebido y codificación del software/hardware.

El cableado de la BeagleBone® Black es reducido y no necesita cable de gran sección puesto que las tensiones a las que está sometido no son de valores elevados (3,3V en las salidas). Se reduce, por tanto, la aparatosidad de la instalación.

8. Conclusiones

A continuación se exponen las conclusiones del presente Trabajo Fin de Máster, surgidas de la experimentación e investigación en sus diferentes fases del proyecto, con los distintos parámetros que establecen la base del control de la ortesis.

1. Se ha computerizado un simulador que permite la prueba de diferentes cambios al control y la experimentación con diferentes pacientes, con el simple cambio de las matrices que representan el movimiento capturado con las cámaras.

2. Se ha calculado el eje de rotación de la rodilla en coordenadas globales del sistema y locales con respecto al fémur para conseguir una simulación real del movimiento humano representado en las matrices de captura de movimiento con los respectivos vectores de cada hueso.

3. Se ha utilizado la Fórmula de Rodríguez para obtener una matriz de rotación que permite el cambio de coordenadas y la simulación del movimiento real de la persona objeto de estudio.

4. Se ha realizado un filtro de velocidad angular que permite la obtención de una señal limpia de ruido que dificulta el control del movimiento de la pierna con ortesis.

5. Se ha realizado un estudio de posibles sensores que puedan mejorar la situación actual.

6. Se ha verificado la mejora del funcionamiento conjunto con la utilización de sensores inerciales o IMUs en lugar de los encoders actuales mediante la optimización de los parámetros de control, obteniendo un error muy inferior en el caso del uso de IMUs.

7. Se han llevado a cabo diferentes estrategias de control, modificando los inputs al mismo.

8. Se han comenzado nuevas líneas de actuación al control, redireccionando el mismo hacia un control por velocidad en lugar de por posición (ángulo de entrada).

9. Se ha abierto un nuevo lazo de control en el que se pretende corregir los ángulos iniciales y finales, expresándolos como un porcentaje del intervalo entre ángulos máximos y mínimos, en el que la velocidad media de la marcha (o en otro orden, levantamiento de pie), sea calculada como el cociente entre la amplitud y el tiempo que tarda en realizar el movimiento.

9. Bibliografía

https://www.sparkfun.com/products/12636

http://www.hobbyking.com/hobbyking/store/__26911__Kingduino_9DOF_ArduIMU_Controller_ATmega328_ACCEL_MAG_GYRO_.html

http://www.murata.com/

[1] J. Cuadrado Aranda, U. Lugrís Armesto, F.J. Alonso Sánchez, J.M. Font-Llagunes, Aplicación de técnicas de dinámica multicuerpo al diseño de ortesis activas para ayuda a la marcha

[2] H. Herr. Exoskeletons and orthoses: classification, design challenges and futuredirections, Journal of NeuroEngineering and Rehabilitation, 6(21) (2009), 1-9.

[3] Y. Xiang, J.S. Arora, K. Abdel-Malek. Physics-based modeling and simulation of human walking: a review of optimization-based and other approaches, Structural Multidisciplinary Optimization, 42 (2010), 1-23.

[4] M. Ackermann, A.J. van den Bogert, Optimality principles for model-based prediction of human gait, Journal of Biomechanics, vol. 43, pp. 1055-1060, 2010.

[5] F.C. Anderson, M.G. Pandy, Dynamic optimization of human walking, Journal of Biomechanical Engineering, vol. 123, pp. 381-390, 2001.

[6] M.L. Audu, C.S. To, R. Kobetic, R.J. Triolo, Gait evaluation of a novel hip constraint orthosis with implication for walking in paraplegia, IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 18, no. 6, pp. 610-618, 2010.

[7] J. Bae, K. Kong, M. Tomizuka, Gait phase-based control for a rotary series elastic actuator assisting the knee joint, Journal of Medical Devices, vol. 5, 031010, pp. 1-6, 2011.

[8] J. Blaya, H. Herr, Adaptive control of a variable-impedance ankle-foot orthosis to assist drop-foot gait, IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 12, no. 1, pp. 24-31, 2004.

[9] G.P. Braz, M. Russold, G.M. Davis, Functional electrical stimulation control of standing and stepping after spinal cord injury: A review of technical characteristics, Neuromodulation vol. 12, no. 3, pp. 180-190, 2009.

[10] S.M. Cain, K.E. Gordon, D.P. Ferris, Locomotor adaptation to a powered ankle-foot orthosis depends on control method, J. Neuroeng Rehabil, vol. 4, art. 48, 2007.

[11] A.M. Dollar, H. Herr, Lower extremity exoskeletons and active orthoses: challenges and state-of-the-art, IEEE Transactions on Robotics, vol. 24, no. 1, pp. 1-15, 2008.

[12] W.K. Durfee, A. Rivard, Design and simulation of a pneumatic, stored-energy, hybrid orthosis for gait restoration, J. Biomech. Eng., vol. 127, pp. 1014-1019, 2005.

[13] K.A. Ferguson, G. Polando, R. Kobetic, R.J. Triolo, E.B. Marsolais, Walking with a hybrid orthosis system, Spinal Cord, vol. 37, pp. 800-804, 1999.

[14] D.P. Ferris, G.S. Sawicki, A. Domingo, Powered lower limb orthoses for gait rehabilitation, Top Spinal Cord Inj. Rehabil., vol. 11, pp. 34-49, 2005.

[15] C. Fleischer, G. Hommel, EMG-driven human model for orthosis control, In: G. Hommel, S. Huanye, editors, Human Interaction with Machines, Netherlands:Springer; pp. 69-76, 2006.

[16] J.M. Font-Llagunes, R. Pàmies-Vilà, J. Alonso, U. Lugrís, Simulation and design of an active orthosis for an incomplete spinal cord injured subject, Proc. IUTAM, vol. 2, pp. 68-81, 2011.

[17] S. Gharooni, B. Heller, M.O. Tokhi, A new hybrid spring brake orthosis for controlling hip and knee flexion in the swing phase, IEEE Trans. Neural. Syst. Rehabil. Eng., vol. 9, pp.106-107, 2001.

[18] M. Goldfarb, K. Korkowski, B. Harrold, W. Durfee, Preliminary evaluation of a controlled-brake orthosis for FES-aided gait, IEEE Trans. Neural Syst. Rehabil. Eng., vol. 11, pp. 241-248, 2003.

[19] J. Hausdorff, W. Durfee, Open-loop position control of the knee joint using electrical stimulation of the quadriceps and hamstrings, Med. Biol. Eng. Comput., vol. 29, pp. 269-280, 1991.

[20] J. Hsu, J. Michael, J. Fisk, AAOS. Atlas de ortesis y dispositivos de ayuda, Elsevier, Barcelona, España, 2009.

[21] A.S-L. Hung, H. Guo, W-H. Liao, D.T-P. Fong, K-M. Chan, Experimental studies on kinematics and kinetics of walking with an assistive knee brace, Proc. IEEE Int. Conf. on Information and Automation, pp. 45-50, Shenzhen, China, 2011.

[22] M.S. Huq, Analysis and control of hybrid orthosis in therapeutic treadmill locomotion for paraplegia. Ph.D. Thesis, Automatic Control and System Engineering, The University of Sheffield, 2009.

[23] E. Isakov, R. Douglas, P. Berns, Ambulation using the reciprocating gait orthosis and functional electrical stimulation, Paraplegia, vol. 30, pp. 239-245, 1992.

[24] R. Jailani, M.O. Tokhi, S. Gharooni, Hybrid Orthosis: the technology for spinal cord injury, Journal of Applied Sciences, vol. 10, no. 22 , pp. 2785-2792, 2010.

[25] R. Jiménez-Fabián, O. Verlinden, Review of control algorithms for robotic ankle systems in lower-limb orthoses, prostheses, and exoskeletons, Medical Engineering and Physics, in press, 2011.

[26] P.C. Kao, C.L. Lewis, D.P. Ferris, Invariant ankle moment patterns when walking with and without a robotic ankle exoskeleton, Journal of Biomechanics, vol. 43, no. 2, pp. 203-209, 2010.

[27] K. Kaufman, S. Irby, J. Mathewson, R. Wirta, D. Sutherland, Energy-efficient knee-ankle-foot orthosis: A case study, Journal of Prosthetiscs & Othotics, vol. 8, no. 3, pp. 79-85, 1996.

[28] H. Kawamoto, Y. Sankai, Power assist system HAL-3 for gait disorder person, Lec. Notes Comp. Science, vol. 2398, pp. 19- 29, 2002.

[29] R. Kobetic, C.S. To, J.R. Schnellenberger, M.L. Audu, T.C. Bulea, Development of hybrid orthosis for standing, walking and stair climbing after spinal cord injury, J. Rehabil. Res. Dev., vol. 46, pp. 447-462, 2009.

[30] C.L. Lewis, D.P. Ferris, Invariant hip moment pattern while walking with a robotic hip exoskeleton, Journal of Biomechanics, vol. 44, no. 5, pp. 789-793, 2011.

[31] K.K. Mankala, S.K. Banala, S.K. Agrawal, Novel swing-assist un-motorized exoskeletons for gait training, Journal of Neuroengineering and Rehabilitation, vol. 6, no. 24, pp. 1-13, 2009.

[32] Č. Marinček, T. Bajd, Surface FES-assisted walking, Critical Reviews in Physical and Rehabilitation Medicine vol. 19, no. 4 , pp. 275-294, 2007.

[33] M. Millard, J. McPhee, E. Kubica, Multi-step forward dynamic gait simulation, C.L. Bottasso (ed.), Multibody Dynamics: Computational Methods and Applications, Springer, 2009.

[34] V. Nekoukar, A. Erfanian, Adaptive terminal sliding mode control of ankle movement using functional electrical stimulation of agonist-antagonist muscles, 32nd Annual International Conference of the IEEE EMBS, pp. 5448–5451, 2010.

[35] E.J. Nightingale, J. Raymond, J.W. Middleton, J. Crosbie, G.M. Davis, Benefits of FES gait in a spinal cord injured population, Spinal Cord, vol. 45, no. 10, pp. 646-657, 2007.

[36] A. Oymagil, J. Hitt, T. Sugar, J. Fleeger, Control of a regenerative braking powered ankle foot orthosis, Proc. 10th International Conference on Rehabilitation Robotics, Noordwijk, The Netherlands, 2007.

[37] J.L. Pons, Wearable robots. Biomechatronic Exoskeletons, John Wiley & Sons, Ltd., Chichester, UK, 2008.

[38] D. Popovic, R. Tomovic, L. Schwirtlich, Hybrid assistive system-the motor neuroprosthesis. IEEE Trans. Biomed. Eng., vol. 36, pp. 729-737, 1989.

[39] J. Pratt, B. Krupp, C. Morse, S. Collins, The Roboknee: an exoskeleton for enhancing strength and endurance during walking, Proc. IEEE International Conference on Robotics & Automation, New Orleans, USA, 2004.

[40] H.A. Quintero, R.J. Farris, M. Goldfarb, Control and implementation of a powered lower limb orthosis to aid walking in paraplegic individuals, Proc. IEEE Int. Conf. on Rehab. Robotics (ICORR'11), pp. 953-958, Zurich, Switzerland, 2011.

[41] L. Ren, R.K. Jones, D. Howard, Predictive modelling of human walking over a complete gait cycle, Journal of Biomechanics, vol. 40, pp. 1567-1574, 2007.

[42] J. Romkes , R. Brunner, Comparison of a dynamic and a hinged ankle–foot orthosis by gait analysis in patients with hemiplegic cerebral palsy, Gait & Posture, vol. 15, no. 1, pp. 18-24, 2002.

[43] C. Rustin, R. Jimenez-Fabian, O. Verlinden, Generation of physiological and balanced human gait by dynamic simulation and optimization, Proceedings of the ECCOMAS Thematic Conference on Multibody Dynamics, Brussels, Belgium, 4-7 July, 2011.

[44] G. Sawicki, D. Ferris, A pneumatically powered knee-ankle-foot orthosis (KAFO) with myoelectric activation and inhibition, Journal of NeuroEngineering & Rehabilitation, vol. 6, no. 23, pp. 1-16, 2009.

[45] O. Schill, R. Wiegand, B. Schmitz, R. Matthies, U. Eck, C. Pylatiuk, M. Reischl, R. Rupp, OrthoJacket: An active FES-hybrid orthosis for the paralysed upper extremity, Biomedizinische Technik, vol. 56, no.1, pp. 35-44, 2011.

[46] N. Sharma, R. Stein, Optimal trajectory planning for a constrained functional electrical stimulation-based human walking

[47] Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS, art.no. 6090134, pp. 603-607, 2011.

[48] P.C. Silva, M.T. Silva, J.M. Martins, Evaluation of the contact forces developed in the lower limb/orthosis interface for comfort design, Multibody System Dynamics, vol. 24, no. 3, pp. 367-388, 2010.

[49] B. Vanderborght, R. Van Ham, D. Lefeber, T.G. Sugar, K.W. Hollander, Comparison of mechanical design and energy consumption of adaptable, passive-compliant actuators, The International Journal of Robotics Research, vol. 28, no. 1, pp. 90-103, 2009.

[50] J.F. Veneman, R. Ekkelenkamp, R. Kruidhof, F.C.T. van der Helm, H. van der Kooij, Design of a series elastic- and Bowdencable-based actuation system for use as torque-actuator in exoskeleton-type training, Proc. IEEE Int. Conf. on Rehabilitation Robotics, pp. 496-499, Chicago, IL, USA, 2005.

[51] Y. Xiang, H.-J. Chung, J.H. Kim, R. Bhatt, S. Rahmatalla, J. Yang, T. Marler, J.S. Arora, K. Abdel-Malek, Predictive dynamics: an optimization-based novel approach for human motion simulation, Structural Multidisciplinary Optimization, vol. 41, pp. 465-479, 2010.

[52] T. Yakimovich, E.D. Lemaire, J. Kofman, Engineering desing review of stance-control knee-ankle-foot orthoses, Journal of Rehab. Research & Development, vol. 46, no. 2, pp 257-267, 2009.

[53] T. Yakimovich, E. Lemaire, J. Kofman, Preliminary kinematic evaluation of a new stance-control knee–ankle–foot orthosis, Clinical Biomechanics, vol. 21, no. 10, pp. 1081-1089, 2006.

[54] Góngora García LH, Rosales García CM, González Fuentes I, Pujals Victoria N. Articulación de la rodilla y su mecánica articular. [artículo en línea]. MEDISAN 2003; 7(2).

[55] Vaughan C.L. - Dynamics of Human Gait. 1999 by Christopher L Vaughan.

[56] Whittle M.W. - Gait Analysis. Fourth edition 2007

10. Análisis de costes

A continuación se procede al análisis de los costes generados durante el transcurso del presente Trabajo Fin de Máster. Estos costes son aquellos derivados de horas de ingeniería, horas de laboratorio y costes tangibles, intangibles y de amortización de equipos, software utilizado y espacio de trabajo en el que se ha llevado a cabo.

Para su realización se ha tenido en cuenta una duración del proyecto de cinco meses, diferenciando entre costes unitarios y coste total del mismo. En el primero de ellos se incluyen todos aquellos gastos referentes a los surgidos durante la elaboración del proyecto, gastos generales, software, equipamientos informáticos y gastos de personal. Sin embargo, los gastos globales se refieren al cómputo de todos los costes reunidos en un único precio que reflejará el coste total del proyecto.

10.1. Precios unitarios

En este capítulo se exponen todos los gastos que han sido necesarios para la realización del trabajo, teniendo en cuenta diversas características generales como la designación, el precio por unidad y los importes totales, además de otros factores propios como el coeficiente de amortización, puesto que ciertos elementos que se han utilizado no serán de uso exclusivo y tendrán uso para proyectos posteriores.

10.1.1. Presupuesto de gastos generales

Este apartado incluye todos los gastos de alquiler, elementos y material de estudio y oficina, el consumo de energía y herramientas de comunicación, y por último, toda la documentación que se ha tenido que tramitar durante su elaboración.

Presupuesto de Gastos Generales				
Ref	Unidades	Designación de las obras	Precio por unidad (€)	Importes totales (€)
GG1	1	Alquiler de oficinas	300	300
GG2	1	Consumo de energía	80	80
GG3	1	Conexión a internet y teléfono	40	40
GG4		Material fungible	246	246
GG5		Documentación administrativa	919,5	919,5
GG6		Mobiliario	200	200
TGG	**Presupuesto total de gastos generales (€)**			**1785,5**

10.1.2. Presupuesto de programas y equipamiento informático

En este apartado se incluyen todos los costes relacionados con equipos y programas informáticos que se han utilizado durante la elaboración del proyecto. Se incorpora además un coeficiente de amortización puesto que los equipos y programas no tienen como destino el uso exclusivo para este proyecto. Para su cálculo se ha utilizado la siguiente fórmula:

$$Coef.\,de\ amortización = Tiempo\ de\ proyecto / Tiempo\ estimado\ de\ amortización$$

Se estima que los equipos y el software tienen un periodo de amortización de al menos tres años, por tanto:

Presupuesto de Programas y equipos informáticos						
Presupuesto de software						
Ref	Unidades	Designación de las obras	Precio por unidad (€)	Importes totales (€)	Coeficiente de amortización	Coste final (€)
HS1	1	Windows 8	300	300	0,14	66,67
HS2	1	Microsoft Office 2013	499	499	0,14	110,89
HS3	1	Blender	libre	libre	-	0,0
HS4	1	Microsoft Visual Studio 2012	647	647	0,14	90,71
HS5	1	CMake	libre	libre	-	0,0
HS6	1	Miro Video Converter	libre	libre	-	0,0
HS7	1	Adobe Photoshop	199	199	0,14	27,86
HS8	1	Adobe Premiere	100	100	0,14	14,0
HS9	1	QtCreator	libre	libre	-	0,0
HS10	1	National Instruments	740	740	0,14	103,6
TGS	**Presupuesto total de software**					**413,73**

Presupuesto de Programas y equipos informáticos						
Presupuesto de equipos						
Ref	Unidades	Designación de las obras	Precio por unidad (€)	Importes totales (€)	Coeficiente de amortización	Coste final (€)
E1	1	Estación de trabajo	3865	3865	0,14	541,1
E2	1	Impresora HP Deskjet 5150	120	120	0,14	16,8
TGE	**Presupuesto total de equipos**					**557,9**

TPEI	**Presupuesto total de equipos y software (€)**	**971,63**

10.2. Presupuesto de personal

Este apartado comprende todo lo referente a la realización del proyecto. Para ello se ha tenido en cuenta el número de horas necesarias para su realización y el precio por hora que ha de cobrar cada trabajador. Este presupuesto se ha fraccionado en tres partes: ingeniería, horas de laboratorio y ensayos, organización y coordinación del proyecto, además de los controles necesarios durante su elaboración.

Presupuesto de Personal				
Presupuesto de ingeniería				
Ref	Unidades	Designación de las obras	Precio por unidad (€)	Importes totales (€)
I1	240	Documentación	35	8400
I2	600	Programación	49,76	29856
I3	40	Elección de sensores	49,76	1990,4
TGI	**Presupuesto total de ingeniería**			**40246,4**

Presupuesto de Personal				
Presupuesto de laboratorio y ensayos				
Ref	Unida-des	Designación de las obras	Precio por unidad (€)	Importes totales (€)
LE1	30	Montaje y preparación	18	540
LE2	10	Ensayos	18	180
TGOC	**Presupuesto total de organización y coordinación**			**720**

Presupuesto de Personal				
Presupuesto de organización y coordinación				
Ref	Unida-des	Designación de las obras	Precio por unidad (€)	Importes totales (€)
OCP1	10	Organización	12	120
OCP2	10	Coordinación	12	120
TGOC	**Presupuesto total de organización y coordinación**			**240**

Presupuesto de Personal				
Presupuesto de control				
Ref	Unidades	Designación de las obras	Precio por unidad (€)	Importes totales (€)
C1	30	Control de ejecución	30	900
C2	40	Control de calidad	30	1200
TGC	**Presupuesto total de control**			**2100**

TPMO	**Presupuesto total de personal (€)**	**43306,4**

10.3. Presupuesto global

En este capítulo se realiza el cómputo de todos los presupuestos parciales anteriormente expuestos incluyendo el IVA para obtener el presupuesto total presentado para el presente proyecto.

Presupuesto		
Ref	Designación	Importes totales (€)
TGG	Presupuesto gastos generales	1785,5
TPEI	Presupuesto equipos y software	971,63
TPMO	Presupuesto de personal	43306,4
Total presupuestos parciales		**46062,53**
Gastos generales (10%)		4606,25
IVA (21%)		9673,13
Presupuesto total del proyecto		**60341,9**

11. Anexos

11.1. Estudio RAMS

La metodología RAMs abarca todas aquellas técnicas y procedimientos empleados en el desarrollo de productos, encaminadas hacia la obtención de una mayor fiabilidad, disponibilidad, mantenibilidad y seguridad (RAMS: Reliability, Availability, maintainability and Safety).

La aplicación de la metodología RAMS a un sistema determinado no se reduce únicamente a la documentación generada durante su fase de diseño o su fase de test. Para una aplicación eficaz de la misma, se deben seguir una serie de acciones y conductas que tengan lugar además de en las fases mencionadas, durante toda la fase de vida del sistema. Los documentos generados durante la fase de diseño e implementación de la metodología RAMs aplicada en el presente TFM son: un Análisis Modal de Fallos Efectos y Criticidad (AMFEC) aplicado a las ortesis y una ficha técnica de un componente comercial significativo. Se mostrarán las repercusiones que un fallo pueda tener en la seguridad del sistema, y además, el funcionamiento entendible como correcto.

Análisis AMFEC

El Análisis Modal de Fallos Efectos y Criticidad (AMFEC) es un estudio detallado que recoge los posibles modos de fallo que puede tener cada pieza o componente individual de un determinado subsistema, así como sus consecuencias sobre el sistema global. El objetivo de este análisis es verificar que ningún modo de fallo pueda causar por sí mismo un impacto severo sobre la seguridad, o sobre la fiabilidad y disponibilidad del equipo. Además de evaluar las consecuencias de los diferentes modos de fallo, con este análisis se pueden priorizar las acciones correctoras en base al riesgo y probabilidad de que se produzca cada fallo.

Para ello, el primer paso es establecer los diferentes niveles de severidad, frecuencia, gravedad y la correspondiente evaluación de cada uno de ellos. Los campos adoptados para el análisis AMFEC en el presente TFM han sido extraídos de la norma UNE 50126-1:2005.

Tabla 5: Severidad del peligro

Severidad	Consecuencias para el servicio
Avería	Fallo que afecta al servicio
Incidencia	Fallo que no afecta al servicio

Tabla 6: Frecuencia sucesión peligro

Categoría	Descripción
Frecuente	Es probable que ocurra con frecuencia. El peligro se experimentará continuamente.
Probable	Se dará varias veces. Puede esperarse que el peligro ocurra con frecuencia.
Ocasional	Es probable que se dé varias veces. Puede esperarse que el peligro ocurra varias veces.
Remoto	Es probable que se dé alguna vez en el ciclo de vida del sistema. Puede razonablemente esperarse que el peligro ocurra.
Improbable	Es improbable, aunque posible que ocurra. Puede suponerse que el peligro ocurrirá excepcionalmente.
Increíble	Es extremadamente improbable que ocurra. Puede suponerse que el peligro pueda no ocurrir.

Tabla 7: Gravedad del peligro

Nivel de Gravedad	Consecuencia para las Personas o el Medio Ambiente	Consecuencia para el Servicio
Catastrófico	Víctimas mortales y / o múltiples heridas graves y /o daños importantes al medio ambiente.	
Crítico	Una sola víctima mortal y / o herida grave y/o daños señalados al medio ambiente	Pérdida de un sistema principal
Mínimo	Heridas menores y / o peligro señalada al medio ambiente	Daño grave a sistema o sistemas
Insignificante	Posible herida menor	Daño menor al sistema

Una vez establecidos y definidos los niveles de severidad, gravedad y frecuencia, se estiman los criterios de no conformidad (o riesgos) según la matriz de aceptación de riesgos.

Tabla 8: Matriz frecuencia-consecuencia

* Frecuencia con que ocurre un suceso de peligro	Niveles de Riesgo			
Frecuente	No Deseable	Intolerable	Intolerable	Intolerable
Probable	Tolerable	No Deseable	Intolerable	Intolerable
Ocasional	Tolerable	No Deseable	No Deseable	Intolerable
Remoto	Insignificante	Tolerable	No Deseable	No Deseable
Improbable	Insignificante	Insignificante	Tolerable	Tolerable
Increíble	Insignificante	Insignificante	Insignificante	Insignificante
	Insignificante	**Mínimo**	**Crítico**	**Catastrófico**
	Niveles de Gravedad de las Consecuencias de un Peligro			

Tanto los criterios de riesgo como la matriz de aceptación adoptados, se tomaron también de la norma UNE-EN 50.126-1:2005. Además, no se admitirá ningún fallo clasificado como "Intolerable" o "No deseable".

Tabla 9: Categorías cualitativas de riesgos

Categoría de Riesgo	Acciones que se han de tomar ante cada categoría
Intolerable	Debe eliminarse
No Deseable	Sólo debe aceptarse cuando la reducción del riesgo sea impracticable y con el acuerdo del cliente
Tolerable	Aceptable con un control adecuado y con el acuerdo del cliente
Insignificante	Aceptable con/sin el acuerdo del cliente

	Modo fallo	Causa	Efecto	Severidad	Detección	Impacto Sist/Seguidad	Frecuencia	Acc.Correctivas
Parte mecánica	▪ Lubricación ▪ Transmisión ▪ Eslabones ▪ Fijaciones	▪ Escasez ▪ Rotura ▪ Fatiga ▪ Mal apriete	No funcionamiento ortesis	Avería	▪ Control visual ▪ Control auditivo ▪ Control manual	Crítico/tolerable	Improbable	▪ Respetar condiciones diseño ▪ Atenerse a normas diseño ▪ Limpieza y lubricación
Control	▪ Entradas ▪ Salidas ▪ Controlador ▪ Regulador ▪ Electrónico	▪ Fallo programación ▪ Fallo sistema ▪ Dispositivo defectuoso ▪ Sobrecargas tensión/corriente	Sin control, no hay funcionamiento automático y controlado. No seguro	Avería	▪ Detección por Software	Crítico	Media	▪ Control sistema software ▪ No introducir entradas sin previa revisión
Sensores	▪ Defecto ▪ Mala conexión ▪ Fallo adquisición datos ▪ Fallo conversión	▪ Elem.comer. ▪ Cable suelto ▪ Programación ▪ Hardware	No existirá realimentación en el control (imposible comparación)	Incidente	▪ Fallo sistema ▪ Fallo comunicación software	Tolerable/tolerable	Media	Comprobar estado sensores periódicamente

	Modo fallo	Causa	Efecto	Severi-dad	Detección	Impacto Sist/Se-guidad	Frecuen-cia	Acc.Correctivas
Moto-res	▪ Defecto motor ▪ Quemado ▪ Rotura	▪ Elem.Comerc. ▪ Sobrecargas ▪ Fallo Cone-xión	No hay movi-miento rela-tivo entre fé-mur y tibia	Avería	▪ Fallo sis-tema ▪ Rigidez disposi-tivo	Crítico/Tolera-ble	Improba-ble	▪ Seguir normas fabri-cante ▪ Verificar Conexio-nes

Según los resultados del análisis desarrollado se puede concretar que el diseño del sistema "Orthesis" no presenta niveles de riesgo "Intolerables" o "No deseables" para los elementos analizados según los modos de fallo considerados. Es decir, en principio se puede deducir del AMFEC que los requisitos RAMS del sistema en materia de seguridad no se verán comprometidos por un fallo en cualquier componente del sistema.

11.1.1. Ficha técnica componente: motor Maxon EPOS2-24/5

Feature	EPOS2 24/2			EPOS2 Module 36/2	EPOS2 24/5	EPOS2 50/5	EPOS2 70/10
	DC (390438)	EC (380264)	DC/EC (390003)	(360665)	(367676)	(347717)	(375711)
							
Motors							
DC motors up to	48 W	—	48 W	72 W	120 W	250 W	700 W
EC motors up to	—	48 W	48 W	72 W	120 W	250 W	700 W
Sensors							
Digital Incremental Encoder (2 or 3 channel with Line Driver)	✓	✓	✓	✓	✓	✓	✓
Digital Hall Sensors (EC motors)	—	✓	✓	✓	✓	✓	✓
SSI Absolute Encoder (single/multiturn configurable, gray or binary coded)	—	—	—	—	—	✓	✓
Analog Incremental Encoder (sin/cos, differential, 2 channel without index)	—	—	—	—	—	✓	✓
Digital Incremental Encoder 2 (2 or 3 channel with Line Driver)	—	—	—	(✓)A	—	✓	✓

(✓)A only available with 2 channel encoder

Communication Interfaces	
CAN	✓
CANopen	Slave
CANopen application layer	DS-301
CANopen frameworks	DSP-305
CANopen profiles motion control	DSP-402
USB 2.0 / USB 3.0 (full speed)	✓
RS232	✓
Gateway function RS232-to-CAN	✓
Gateway function USB-to-CAN	✓

Feature	EPOS2 24/2			EPOS2 Module 36/2	EPOS2 24/5	EPOS2 50/5	EPOS2 70/10
	DC (390438)	EC (380264)	DC/EC (390003)	(360665)	(367676)	(347717)	(375711)
Electrical Data							
Operating voltage $+V_{cc}$	9...24 VDC			11...36 VDC	11...24 VDC	11...50 VDC	11...70 VDC
Logic supply voltage $+V_c$ (optional)	—			11...36 VDC	11...24 VDC	11...50 VDC	11...70 VDC
Max. output voltage				0.90 x V_{cc}			
Max. output current	4 A			4 A	10 A	10 A	25 A
Continuous output current	2 A			2 A	5 A	5 A	10 A
Efficiency	90%			93%	92%	94%	94%
Switching frequency output stage	100 kHz			50 kHz	50 kHz	50 kHz	50 kHz
Sampling rate PI current controller				10 kHz			
Sampling rate PI speed controller				1 kHz			
Sampling rate PID position controller				1 kHz			
Maximal speed (DC)				25 000 rpm			
Maximal speed (EC/block commutation)				100 000 rpm			
Maximal speed (EC/sinusoidal commutation)				25 000 rpm			
Built-in motor choke per phase	47 µH / 2 A			10 µH / 2 A	15 µH / 5 A	22 µH / 5 A	25 µH / 10 A

Inputs							
Hall sensor signals	—	H1,H2,H3					
Encoder signals	A,A\,B,B\,I,I\						
Max. encoder input frequency	5 MHz						
Max. encoder counts per turn	2 500 000 Imp.						
Digital inputs (thereof "high speed" up to 5 MHz)	6	6	6	6 (2)	6	11 (4)	10 (3)
Analog inputs	2	2	2	2	2	2	2
Resolution	12-bit			11-bit	12-bit	12-bit	12-bit
Range	0…+5 V			0…+5 V	0…+5 V	-10…+10 V	0…+5 V
CAN ID	1…15 (up to 127 by software setting)			1…127			

Outputs							
Digital outputs (thereof "high speed" up to 5 MHz)	2	2	2	3 (1)	4	5 (1)	5 (1)
Analog outputs	—	—	—	—	—	1	—
Encoder voltage output	+5VDC@100 mA						
Hall sensor voltage output	—	+5VDC@30 mA					
Auxiliary voltage output	+5 VDC@10 mA			—	$+V_{cc}$@1300 mA	+5 VDC@150 mA	+5 VDC@150 mA
Reference voltage output	—	—	—	—	—	—	+5 VDC (R_i = 1kΩ)

Mechanical Data							
Weight (approximate)	27 g	30 g	28 g	10 g	170 g	240 g	330 g
Dimensions (LxWxH)	55 x 40 x 15.6 mm	55 x 40 x 19.6 mm	55 x 40 x 18.2 mm	54.5 x 28.2 x 9 mm	105 x 83 x 24 mm	120 x 93.5 x 27 mm	150 x 93 x 27 mm
Mounting	M2.5 screws	M2.5 screws	M2.5 screws	Card edge connector	M3 screws	M3 screws	M3 screws

Environmental Conditions	
Operation temperature range	-10…+45°C
Storage temperature range	-40…+85°C
Humidity range (condensation not permitted)	20…80%

Functionalities							
Profile Position Mode (point-to-point)	✓						
Path generator with sinusoidal/trapezoidal profiles	✓						
Position Mode (POM)	✓						
Interpolated Position Mode (PVT)	✓						
Profile Velocity Mode (PVM)	✓						
Velocity Mode (VEM)	✓						
Current (Torque) Mode (CUM)	✓						
Homing Mode	✓						
Master Encoder Mode (Electronic gearhead)	✓						
Step/Direction Mode	✓						
Analog set value commands (CUM,VEM,POM)	✓						
Position Marker	✓						
Quickstop	✓						
Enable	✓						
Position Compare	✓						
Control of holding brakes	(✓)B	(✓)B	(✓)B	(✓)B	✓	✓	✓
Advanced automatic control settings	✓						
Position control with Feed Forward	✓						
Speed control with Feed Forward	✓						
Dual Loop Position and Speed Control	—	—	—	✓	—	✓	✓

(✓)B separate power supply for holding brake is required

11.2. Árbol de fallos

Otro de los análisis requeridos en el actual estudio RAMS es el árbol de fallos de la ortesis objeto de estudio del presente TFM. Las posibles combinaciones de fallos simples que pueden dar lugar al fallo global de un determinado subsistema pueden visualizarse de una forma mucho más clara y directa gracias a la elaboración de un árbol de fallos.

El análisis del árbol de fallos constituye un análisis inductivo, es decir, partiendo desde el fallo más global, busca las diferentes causas primarias que pueden provocarlo hasta llegar al origen de dicho problema. El árbol de fallos permite estudiar la conjunción de distintos fallos simultáneos en diferentes componentes, y el impacto que esa conjunción tiene en el sistema global (a diferencia del AMFEC, que buscaba la dependencia de elementos simples en el fallo total del sistema).

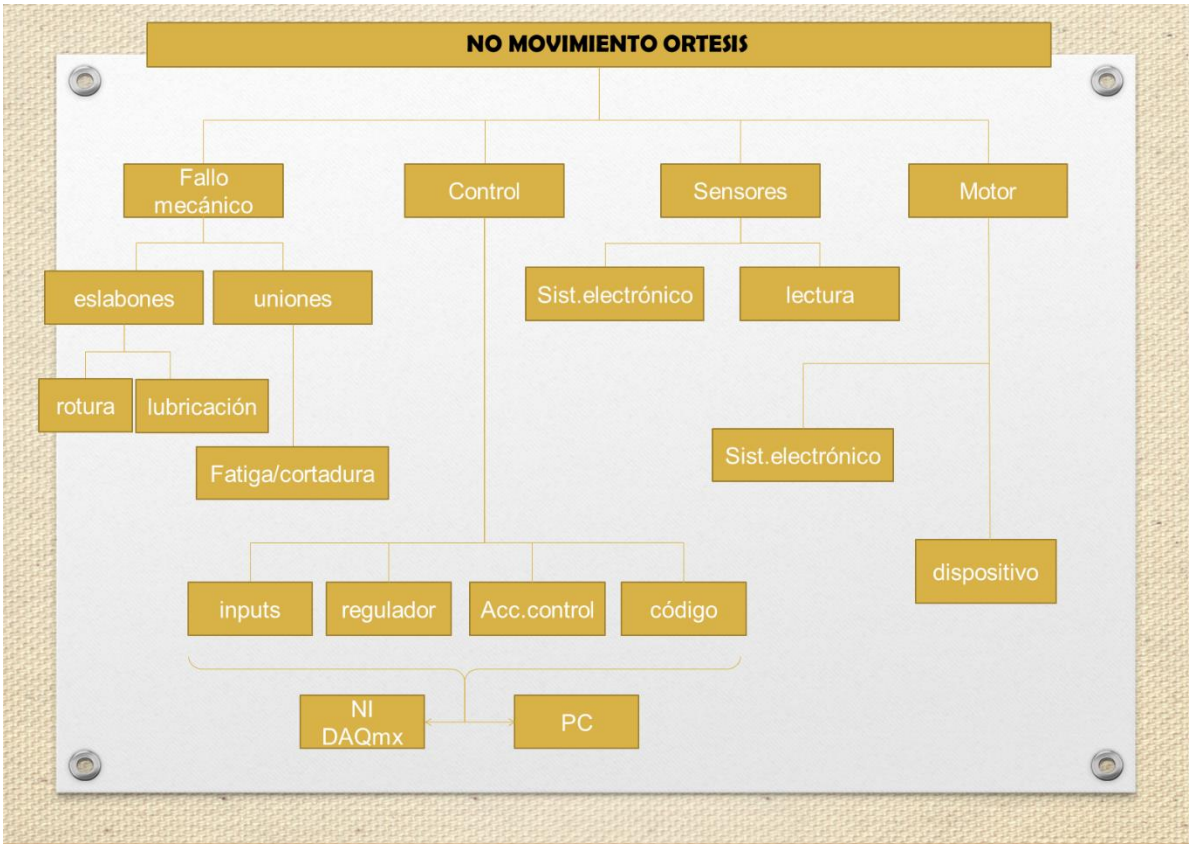

Figura 63: Árbol de fallos de la ortesis

11.3. Código c++

En este apartado se adjunta el código de movimiento con las diferentes clases que componen el control de la ortesis.

Clase EPOS2: motores

EPOS2.h

```
#ifndef EPOS2_H
#define EPOS2_H

#include <string>
#include <vector>
#include "Definitions.h"



#ifdef EPOS2_EXPORTS
	#define EPOS2Export __declspec(dllexport)
#else
	#define EPOS2Export __declspec(dllimport)
#endif
```

```
#pragma warning(disable:4251 4351)

#define TRAPEZOIDAL_PROFILE 0
#define SINUSOIDAL_PROFILE 1

static const double QC_PER_DEG = 2600.0/3;

class EPOS2Export maxonMotor {
protected:
	static const double pi;
	static HANDLE keyHandle;
	static WORD nMotors;

	WORD motor;
	DWORD error;
	BOOL reverse;
	int nFrames, currentFrame;
	long homePos, desiredPos;
	std::vector<std::vector<double> > data;

	void errChk(BOOL success);

public:
	maxonMotor(int length = 1000);
	//~maxonMotor();

	void setup(const bool rev);
	//void peek(const double t, double *out);
	void gotoPosition(long pos);
	void write(std::string file);
	void reset();

};


#endif // EPOS2_H
```

EPOS2.cpp

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <math.h>

#include "EPOS2.h"


// Initialize static constants
const double maxonMotor::pi = 4*atan(1.0);
HANDLE maxonMotor::keyHandle = 0;
WORD maxonMotor::nMotors = 0;

// maxonMotor constructor
maxonMotor::maxonMotor(const int length) : currentFrame(0), desiredPos(0)
{
	nFrames = length;
```

```
	if (!keyHandle)
	{
		char dev[] = "EPOS2";
		char ptc[] = "MAXON SERIAL V2";
		char ifc[] = "USB";
		char prt[] = "USB0";
		keyHandle = VCS_OpenDevice(dev, ptc, ifc, prt, &error);
	}
	if (!keyHandle)
		errChk(false);

	// Assign incremental motor number
	motor = nMotors + 1;


	// Reset motor state
	errChk(VCS_ClearFault(keyHandle, motor, &error));
	errChk(VCS_SetDisableState(keyHandle, motor, &error));

	// Increase motor counter if successful
	nMotors++;
}

 maxonMotor destructor
maxonMotor::~maxonMotor()
{
	if (keyHandle) {
		VCS_SetDisableState(keyHandle, motor, &error);
		std::cout << "Motor " << motor << " disabled" << std::endl;
		nMotors--;
		if (nMotors == 0) {
			VCS_CloseAllDevices(&error);
			std::cout << "EPOS2 cleared" << std::endl;
			keyHandle = 0;
		}
	}
}

//EPOS2 error parser
void maxonMotor::errChk(BOOL success)
{
	if (!success) {
		char errBuff[100];
		if (VCS_GetErrorInfo(error, errBuff, 100)) {
			std::cout << "EPOS2 error " << error << " (" << errBuff << ")"
<< std::endl;
		}
		throw "EPOS2";
	}
}

// Motor configuration
void maxonMotor::setup(bool rev)
{
	//DWORD written;
	/*DWORD maxVel = 6720;
	DWORD maxAcc = 50000;*/
```

```
	short int profile = TRAPEZOIDAL_PROFILE;
	__int8 homingMethod;

	reverse = rev;

	// Allocate data buffers (time, desired position, actual position)
	data.resize(3);

	for (int i = 0; i < 3; i++)
		data[i].resize(nFrames);

	// Set general motor parameters
	errChk(VCS_SetMotorType(keyHandle, motor, MT_EC_BLOCK_COMMUTATED_MOTOR, &er-
ror));
	errChk(VCS_SetObject(keyHandle, motor, 0x6410, 0x04, &maxVel, sizeof maxVel,
&written, &error));
	errChk(VCS_SetObject(keyHandle, motor, 0x6086, 0x00, &profile, 2, &written,
&error));
	errChk(VCS_SetMaxProfileVelocity(keyHandle, motor, maxVel, &error));
	errChk(VCS_SetMaxAcceleration(keyHandle, motor, maxAcc, &error));
	errChk(VCS_SetMaxFollowingError(keyHandle, motor, long(90*QC_PER_DEG+0.5),
&error));
	errChk(VCS_SetPositionProfile(keyHandle, motor, maxVel, maxAcc, maxAcc, &er-
ror));

	// Set the reverse rotation bit if necessary
	if (reverse)
		homingMethod = HM_CURRENT_THRESHOLD_POSITIVE_SPEED_AND_INDEX;
	else
		homingMethod = HM_CURRENT_THRESHOLD_NEGATIVE_SPEED_AND_INDEX;

	std::cout << "Motor " << motor << " homing... ";

	// Find home positions at mechanical end stop
	errChk(VCS_SetOperationMode(keyHandle, motor, OMD_HOMING_MODE, &error));
	errChk(VCS_SetHomingParameter(keyHandle, motor, maxAcc/10, maxVel, maxVel, 0,
5000, 0, &error));
	errChk(VCS_ActivateHomingMode(keyHandle, motor, &error));
	errChk(VCS_SetEnableState(keyHandle, motor, &error));
	errChk(VCS_FindHome(keyHandle, motor, homingMethod, &error));
	errChk(VCS_WaitForHomingAttained(keyHandle, motor, 5000, &error));

	// Set home position as reference
	errChk(VCS_GetPositionIs(keyHandle, motor, &homePos, &error));

	// Activate profile position mode
	errChk(VCS_SetOperationMode(keyHandle, motor, OMD_PROFILE_POSITION_MODE, &er-
ror));
	errChk(VCS_ActivateProfilePositionMode(keyHandle, motor, &error));

	std::cout << "ready" << std::endl;
}



// Motor data storage
```

```
void maxonMotor::peek(double t, double *out)
{
	long qcs;

	// Get desired and actual position
	errChk(VCS_GetPositionIs(keyHandle, motor, &qcs, &error));
	data[0][currentFrame] = t;
	if (reverse) {
		data[1][currentFrame] = -static_cast<double>(desiredPos)/QC_PER_DEG;
		data[2][currentFrame] = -static_cast<double>(qcs)/QC_PER_DEG;
	} else {
		data[1][currentFrame] = static_cast<double>(desiredPos)/QC_PER_DEG;
		data[2][currentFrame] = static_cast<double>(qcs)/QC_PER_DEG;
	}
	out[0] = data[2][currentFrame];
	out[1] = data[1][currentFrame];
	currentFrame++;
}


// Motor go-to-position command
void maxonMotor::gotoPosition(const long pos)
{
	if (reverse)
		desiredPos = -pos;
	else
		desiredPos = pos;

	errChk(VCS_MoveToPosition(keyHandle, motor, desiredPos + homePos, true, true,
&error));
}

// Dump sensor data to file
void maxonMotor::write(std::string file)
{
	std::ofstream outFile(file.c_str());
	if (outFile.is_open())
	{
		outFile << std::setprecision(8) << std::fixed;
		for (int i = 0; i < currentFrame; i++)
		{
			for (int c = 0; c < 3; c++)
			{
				outFile << std::setw(16) << data[c][i];
			}
			outFile << std::endl;
		}
		outFile.close();
	}
}

// Reset frame counter
void maxonMotor::reset()
{
	currentFrame = 0;
	desiredPos = 0;

};
```

Header Files

CalcularNorma.h

```
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <complex>
#include <vector>
#include <math.h>
#include <iostream>
#include <fstream>

class CalcularNorma
{
public:
	CalcularNorma::CalcularNorma ();
public:
	double CalcularNorma::Norm (double u[3]);

};
```

Control.h

```
#ifndef CONTROL_H
#define CONTROL_H

#include <math.h>
#include <vector>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

#include "hrtimer.h"
#include "Schmitt.h"
#include "EPOS2.h"


#define pi 3.14159265359
#define conv_to_rad pi/180.0


using namespace std;
using namespace std::chrono;
using std::ofstream;


// Functor that controls a motor depending on encoder and FSR inputs
class motorControl
{
private:
	static int nc;

	enum mode_t{STANCE, SWING};
```

```
        int mcid;
        mode_t mode;
        hrtimer timer;
        double initOppAnk, oldOppAnk;
        maxonMotor &motor_;

        double kr_, ks_, kw_, ia_, fa_, to_, th_, cutoff_;
        status_t oldOwnSt;

        double control_input [2];  //los inputs al control son los ángulos de tobillos
        double control_output [2]; //los outputs son las velocidades. Se filtran para
obtener menos retrasos (derivada de la posición, output original)


public:

        std::ofstream data; //almacenamiento de datos para exportar a Matlab y grafi-
car

        motorControl (maxonMotor &motor,                // Motor
                double kr = 1.117,       // Knee rotation range (degrees). Now radians
                double ks = 1.2035,        // Peak translation (0 = symmetric)
                double kw = -0.3262,              // Peak width (0 = normal)
                double ia = -0.4515, // Initial angle w.r.t. neutral position (de-
grees). Now radians
                double fa = 0.1261, // Final angle w.r.t neutral position (degrees).
Now radians
                double to = 0.20,          // Timeout for stance detection (seconds)
                double th = 05.0,    // Threshold ankle angle for stance (degrees)
                float cutoff = 200.0) :                 //frecuencia anterior de 0,5Hz

                motor_ (motor), oldOwnSt (HIGH), oldOppAnk (0), mode (STANCE)
        {
                kr_ = kr; ks_ = ks; kw_ = kw; ia_ = ia; fa_ = fa; to_ = to; th_ = th;
mcid = ++nc; cutoff_ = cutoff;

                //administración exportación de archivos

                if (mcid == 1) //identificador de pierna activa
                        data.open ("datos1.dat", ios::out);

                if (mcid == 2)
                        data.open ("datos2.dat", ios::out);

                memset (control_input, 0, 2 * sizeof(double)); //inicializar a 0 las
variables de control
                memset (control_output, 0, 2 * sizeof(double));
        }

        ~motorControl () { nc--; data.close (); }


        double operator()(const double oppAnk, const status_t ownSt, const status_t
oppSt)
        {

                static double outAngle = 0;
```

```
		//***First filter order
		double fltPar = 0.0;			//representa la a en el filtro
		float sampleRate = 100.0;		// 100 Hz

		if (cutoff_ > 0) //si la frecuencia de corte es mayor que 0
			fltPar = 1.0 / (sampleRate / (2.0*pi*cutoff_) + 1.0); //Cuanto
más cercana esté a de 1, más se mayora la evaluación de entradas y menos la velocidad
en el instante anterior


		control_input [1] = control_input [0];
		control_input [0] = oppAnk;

		control_output [1] = control_output [0];
		control_output [0] = fltPar*(sampleRate*(control_input [0] - con-
trol_input [1])) + (1.0 - fltPar)*control_output [1];

		//std::cout << control_output [0] << std::endl;

		//exportar datos a archivos para plotear en matlab

		data << std::setw (10) << std::setprecision (8) << std::fixed << oppAnk
<< "\t" << sampleRate*(control_input [0] - control_input [1]) << "\t" << control_out-
put [0] << endl;

		//*****
		// Start cycle when own foot leaves the ground
		if (oldOwnSt == HIGH && ownSt == LOW)
		{
			timer.reset ();
			initOppAnk = oppAnk;
			mode = SWING;
		}

		switch (mode)
		{
			case SWING:

				// Check whether the state is compatible with walking

				if (ownSt == HIGH)		//if foot is on the
floor
				{
					mode = STANCE;
					motor_.gotoPosition(0);
					outAngle = 0;   //angulo de la rodilla contraria
será 0
					break;
				}

				// Go to calculated position
				//if (oldOppAnk < oppAnk) //if the opposite angle is big-
ger than the previous one. That means the movement-forward
				//	motor_.gotoPosition(motorAngle(oppAnk));


				//la velocidad será positiva si el ángulo actual es mayor
que el anterior, va hacia delante
```

```
                        //Control de velocidad: ir a posición según ángulo tobillo
opuesto

                        if (control_output [0] > 0)
                        {
                                motor_.gotoPosition (motorAngle (oppAnk));
                                outAngle = motorAngle (oppAnk); //a la rodilla con-
traria se le pasa como entrada el ángulo del tobillo que pisa

                        }

                        break;

                case STANCE:
                        // Go to straight position
                        motor_.gotoPosition(0);
                        outAngle = 0;
                        break;
        }

        oldOwnSt = ownSt;
        oldOppAnk = oppAnk;


        return outAngle;
    }

    void setParameter(const double x, const int par)
    {
        switch (par)
        {
            case 0: kr_ = x; break;
            case 1: ks_ = x; break;
            case 2: kw_ = x; break;
            case 3: ia_ = x; break;
            case 4: fa_ = x; break;
            case 7: to_ = x; break;
            case 8: th_ = x; break;
            default:
                std::cout << "Unknown parameter" << std::endl;
        }
    }


    double motorAngle(const double a) //ojo que hay que pasar todo a radianes, "a"
original está en grados
    {
        double knee = 0.0;
        double w = 4.0*atan(1.0)/(fa_-ia_);

        // Phase shift for tuning curve shape
        double phase = ks_*sin(w*(a-ia_)) + kw_*sin(2*w*(a-ia_));

        // Sine function with phase shift
        if (a > ia_ && a < fa_)
            knee = kr_*(sin(w*(a-ia_) - phase));
```

```
                return (knee); //el calculo se realiza en radianes
        }


};

#endif // CONTROL_H
```

homeSwitcher.h

```
#ifndef _HOMESWITCHER_H_
#define _HOMESWITCHER_H_

#include <osgGA/TrackballManipulator>
#include <osgViewer/ViewerEventHandlers>



class homeSwitcher : public osgGA::GUIEventHandler
{
        osgGA::TrackballManipulator *m;

public:
        homeSwitcher (osgGA::TrackballManipulator *manipulator);
        bool handle (const osgGA::GUIEventAdapter &ea, osgGA::GUIActionAdapter &);
};

#endif
```

MatrixTrans.h

```
#ifndef _MatrixTrans_H_
#define _MatrixTrans_H_

#include <osg/Node>
#include <osg/Group>
#include <osg/MatrixTransform>
#include <osgDB/ReaderWriter>
#include <osgGA/GUIEventHandler>
#include <osgText/Text>

#include <map>

enum Bodykind { geom, mesh, chainLink };

class Body
{
public:
        Bodykind kind;              /**< Tipo de sólido: 0 sin malla, 1 con malla, 2
sólido AABB */
        osg::ref_ptr<osg::MatrixTransform> Mt; /**< Matriz de transformación del
cuerpo */
        osg::Node *Nodo;     /**< Nodo gráfico */
        Body() {Mt=new osg::MatrixTransform;}; /**< Constructor */
```

```
	virtual ~Body(){};
};

class MatrixTrans
{

protected:
	std::vector<Body *> Ground; /**< Vector de malla fija */
	std::vector<Body *> Bodies; /**< Vector de mats.trans. de malla móvil */

	typedef std::map<std::string, osg::ref_ptr<osg::Node> > geodeMap;

	geodeMap geometriaComun;

public:
	MatrixTrans (); //Constructor

	enum options { nondisplayvectors, displayvectors };
	int numMatTransBodies (){ return Bodies.size (); } /*< Devuelve el numero de
matrices de transformación de los bodies */

	//MatrixTrans (options opt, std::string file); /*< Constructor con opciones */

	void MatTransBodies (double *Mt); /*< Actualiza las matrices de transformación
de los bodies */

};


#endif /* _MatrixTrans_H_ */
```

Schmitt.h

```
#ifndef SCHMITT_H
#define SCHMITT_H

enum status_t {LOW, HIGH};

// Schmitt trigger emulation functor
class Schmitt
{
private:
	status_t status_;
	double ht_, lt_;

public:
	Schmitt(const double ht, const double lt) : status_(LOW)
	{
		ht_ = ht; // Set HIGH transition value
		lt_ = lt; // Set LOW transition value
	}

	status_t operator()(const double input)
	{
		if (input > ht_ && status_ == LOW) status_ = HIGH;
		if (input < lt_ && status_ == HIGH) status_ = LOW;
```

```
		return status_;
	}

	void setLow(const double input)
	{
		lt_ = input;
	}

	void setHigh(const double input)
	{
		ht_ = input;
	}
};


#endif // SCHMITT_H
```

Source Files

CalcularNorma.cpp

```
#include <CalcularNorma.h>

using namespace std;

//Constructor
CalcularNorma::CalcularNorma (){}

double CalcularNorma::Norm (double u[3])		//función para calcular la norma 2 de
un vector
{
	double resultado=0;
	for (int i = 0; i < 3; i++)
	{
		resultado += pow(fabs(u[i]),2);

	}
	resultado = sqrt (resultado);
	return resultado;
}
```

homeSwitcher.cpp

```
#include "homeSwitcher.h"

homeSwitcher::homeSwitcher (osgGA::TrackballManipulator *manipulator)
{
	m = manipulator;
}

bool homeSwitcher::handle (const osgGA::GUIEventAdapter &ea, osgGA::GUIActionAdapter
&)
{
	switch (ea.getEventType ())
```

```
	{
		case(osgGA::GUIEventAdapter::KEYUP) :
		{


		switch (tolower (ea.getKey ()))
			{
			case(osgGA::GUIEventAdapter::KEY_1) : // Sagittal plane view
			{

				m->setHomePosition (osg::Vec3 (0.0, -4.0, -0.5),  // eye
				osg::Vec3 (0.0, 0.0, -0.5),  // center
				osg::Vec3 (0.0, 0.0, 1.0)); // up
				m->home (0);
				return true;
			}
			case(osgGA::GUIEventAdapter::KEY_2) : // Frontal plane view
			{
				m->setHomePosition (osg::Vec3 (4.5, 0.0, -0.5),  // eye
				osg::Vec3 (0.0, 0.0, -0.5),  // center
				osg::Vec3 (0.0, 0.0, 1.0)); // up
				m->home (0);
				return true;
			}
			case(osgGA::GUIEventAdapter::KEY_3) : // Transverse plane view
			{
				m->setHomePosition (osg::Vec3 (0.0, 0.0, 2),  // eye
				osg::Vec3 (0.0, 0.0, 0.0),  // center
				osg::Vec3 (0.0, 1.0, 0.0)); // up
				m->home (0);
				return true;
			}
			case(osgGA::GUIEventAdapter::KEY_0) : // Perspective view
			{
				m->setHomePosition (osg::Vec3 (3.4, -3.0, -0.5),  // eye
				osg::Vec3 (0.0, 0.0, -0.5),  // center
				osg::Vec3 (0.0, 0.0, 1.0)); // up
				m->home (0);
				return true;
			}
		}
	}
	default:
		return false;
	}
}
```

MatrixTrans.cpp

```
#include "MatrixTrans.h"
#include <osgDB/ReadFile>


MatrixTrans::MatrixTrans ()
{

}

void MatrixTrans::MatTransBodies (double *Mt)
{
	for (int i = 0; i<numMatTransBodies (); i++)
	{
		osg::Matrixd M (Mt + 16 * i);
		Bodies[i]->Mt->setMatrix (M);
	}

}
```

Bucle principal

```
////////////////////////////////////////////////////////////////////////////////
//                                                                            //
// Programa para simulación de capturas de movimiento. Link con OpenSceneGraph //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////


#include <osg/Switch>
#include <osg/MatrixTransform>
#include <osg/PolygonMode>
#include <osg/Material>
#include <osgDB/ReadFile>
#include <osgViewer/Viewer>
#include <osgGA/OrbitManipulator>
#include <osgViewer/api/Win32/GraphicsWindowWin32>
#include <osg/BlendFunc>
#include <osg/Texture2D>
#include <osg/StateSet>

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <iomanip>
#include <stdio.h>
#include <complex>
#include <conio.h>
#include <math.h>
#include <chrono>

#include "homeSwitcher.h"
#include "MatrixTrans.h"
#include "CalcularNorma.h"
```

```
#include "Control.h"


#define ncols 174          //número de columnas de la matriz q que continene tanto
los puntos como los vectores de cada modelo
#define nfils 157          //número de filas de la matriz q
#define npoints 22         //número de puntos contenidos en el fichero ip del que se
obtienen la ubicación de las referencias en q
#define nvector 36         //ubicación de los vectores iv contenidos en la matriz q
#define PI  3.14159265359
#define constante_motor 36000     //     3600*sampleRate/10


#define rLeg 0
#define lLeg 1

#define umbral_fuerza 25
#define umbral_delta 0.040


using namespace std;
using namespace std::chrono;
using namespace osg;

int motorControl::nc = 0;

int main (int argc, char** argv)
{
       //Variables a utilizar

    double er[3] = { 0 };
    double norma_wr;
    double wr[3] = { 0 };       //vector eje de giro resultante de la matriz antisi-
métrica
    double numerador11[3] , numerador12[3] , denominador[3] = { 0 }; //variables tem-
porales para calcular el eje de giro óptimo

       //Se utiliza la clase Schmitt para que solo salte de estado en ciertas ocasio-
nes prefijadas por los valores

       Schmitt rTrigger (20.0, 5.0); //valores HIGH y LOW
       Schmitt lTrigger (20.0, 5.0);

       //Creación de objeto de la clase Calcular Norma para poder calcular el vector
w de cada movimiento

       CalcularNorma norma;

       //Objeto de la clase control para realizar el movimiento de una pierna en fun-
ción del ángulo de tobillo de la contraria

       maxonMotor rMotor (constante_motor); //rmtr(nFrames/10); donde nFrames=CapLen-
gth*Sr/10=360000
       maxonMotor lMotor (constante_motor); //clase EPOS2

       // Initialize motors (left is reversed)
```

```
        rMotor.setup (false);
        lMotor.setup (true);

        motorControl rMotorControl (rMotor); //clase control
        motorControl lMotorControl (lMotor);

        std::ofstream angulo_tibia; //almacenamiento de datos para exportar a Matlab y
graficar
        angulo_tibia.open ("datos_tibia.dat", ios::out);

        std::ofstream angulos_tobillo_rodilla; //almacenamiento de datos para exportar
a Matlab y graficar
        angulos_tobillo_rodilla.open ("datos_angulos_plot.dat", ios::out);


        //Lectura de los archivos correspondientes a femur, tibia+peroné y pie PIERNA
DCHA

        osg::ref_ptr<osg::Node> model0 = osgDB::readNodeFile ("Pelvis.ive");
        osg::ref_ptr<osg::Node> model1 = osgDB::readNodeFile ("RFemurSinOrtesis.ive");
        osg::ref_ptr<osg::Node> model2 = osgDB::readNodeFile ("RTibiaSinOrtesis.ive");
    osg::ref_ptr<osg::Node> model2r = osgDB::readNodeFile ( "RTibiaSinOrtesis.ive" );
//para visibilidad de pierna izq y modificada
        osg::ref_ptr<osg::Node> model3 = osgDB::readNodeFile ("RFootSinOrtesis.ive");
    osg::ref_ptr<osg::Node> model3r = osgDB::readNodeFile ( "RFootSinOrtesis.ive" );
//visibilidad pierna izq y modificada
    osg::ref_ptr<osg::Node> model4 = osgDB::readNodeFile ( "RToe.ive" );
    osg::ref_ptr<osg::Node> model4r = osgDB::readNodeFile ( "RToe.ive" );   //visibi-
lidad pierna izq y modificada


   /*************    Pierna original derecha en transparente    ***************/

    /* TIBIA */
    // Create StateSet and Material
    osg::StateSet* state2 = model2->getOrCreateStateSet ( );
    osg::ref_ptr<osg::Material> mat2 = new osg::Material;

    mat2->setAlpha ( osg::Material::FRONT_AND_BACK , 0.3);
    state2->setAttributeAndModes ( mat2.get ( ) , osg::StateAttribute::ON |
                                   osg::StateAttribute::OVERRIDE );

    // Turn on blending
    osg::BlendFunc* bf = new osg::BlendFunc ( osg::BlendFunc::SRC_ALPHA ,
                                              osg::BlendFunc::ONE_MINUS_SRC_ALPHA );
    state2->setAttributeAndModes ( bf );

    /* FOOT */

    // Create StateSet and Material
    osg::StateSet* state1 = model3->getOrCreateStateSet ( );
    osg::ref_ptr<osg::Material> mat3 = new osg::Material;

    mat3->setAlpha ( osg::Material::FRONT_AND_BACK , 0.3 );
    state1->setAttributeAndModes ( mat3.get ( ) , osg::StateAttribute::ON |
                                   osg::StateAttribute::OVERRIDE );

    // Turn on blending
```

```
    osg::BlendFunc* bf1 = new osg::BlendFunc ( osg::BlendFunc::SRC_ALPHA ,
                                               osg::BlendFunc::ONE_MINUS_SRC_ALPHA );
    state1->setAttributeAndModes ( bf1 );

    /* TOE */

    // Create StateSet and Material
    osg::StateSet* state0 = model4->getOrCreateStateSet ( );
    osg::ref_ptr<osg::Material> mat0 = new osg::Material;

    mat0->setAlpha ( osg::Material::FRONT_AND_BACK , 0.3 );
    state0->setAttributeAndModes ( mat0.get ( ) , osg::StateAttribute::ON |
                                   osg::StateAttribute::OVERRIDE );

    // Turn on blending
    osg::BlendFunc* bf0 = new osg::BlendFunc ( osg::BlendFunc::SRC_ALPHA ,
                                               osg::BlendFunc::ONE_MINUS_SRC_ALPHA );
    state0->setAttributeAndModes ( bf0 );



////////////////////////////////////////////////////////////////////////////////



    /*************************    Pierna derecha    ****************************/
    //PELVIS
    //Matriz especial:cadera centro de coordenadas para el resto de los huesos
    osg::ref_ptr<osg::MatrixTransform> mat_trans0 = new osg::MatrixTransform;
    mat_trans0->setMatrix ( osg::Matrix::translate ( 0.0f , 0.0f , 0.0f ) );
    mat_trans0->addChild ( model0.get ( ) );


       //FEMUR
       //Creamos matriz de transformación
       osg::ref_ptr<osg::MatrixTransform> mat_trans1 = new osg::MatrixTransform;
       mat_trans1->setMatrix (osg::Matrix::translate (0.0f, -0.0795f, -0.24f));
       //se modifica la posición de la matriz y se añade el fémur como hijo
       mat_trans1->addChild (model1.get ());


       //TIBIA
       osg::ref_ptr<osg::MatrixTransform> mat_trans2 = new osg::MatrixTransform;
       mat_trans2->setMatrix (osg::Matrix::translate (0.0f, -0.0795f, -0.6275f));
       mat_trans2->addChild (model2.get ());

    //Tibia secundaria
    osg::ref_ptr<osg::MatrixTransform> mat_trans2sec = new osg::MatrixTransform;
    mat_trans2sec->setMatrix ( osg::Matrix::translate ( 0.0f , -0.0795f , -0.6275f )
);
    mat_trans2sec->addChild ( model2r.get ( ) );


       //FOOT
       osg::ref_ptr<osg::MatrixTransform> mat_trans3 = new osg::MatrixTransform;
       mat_trans3->setMatrix (osg::Matrix::translate (0.0f, -0.0795f, -0.989f));
       mat_trans3->addChild (model3.get ());
```

```
    //Foot secundario
    osg::ref_ptr<osg::MatrixTransform> mat_trans3sec = new osg::MatrixTransform;
    mat_trans3sec->setMatrix ( osg::Matrix::translate ( 0.0f , -0.0795f , -0.989f )
);
    mat_trans3sec->addChild ( model3r.get ( ) );


    //TOE
    osg::ref_ptr<osg::MatrixTransform> mat_trans4 = new osg::MatrixTransform;
    mat_trans4->setMatrix ( osg::Matrix::translate ( 0.1102f , -0.0795f , -1.0376f )
);
    mat_trans4->addChild ( model4.get ( ) );



    /*************************    Pierna izq   ****************************/

    //PELVIS
    //Matriz especial:cadera centro de coordenadas para el resto de los huesos
    osg::ref_ptr<osg::MatrixTransform> mat_trans0_i = new osg::MatrixTransform;
    mat_trans0_i->setMatrix ( osg::Matrix::translate ( 0.0f , 0.0f , 0.0f ) );
    mat_trans0_i->addChild ( model0.get ( ) );

    //FEMUR
    //Creamos matriz de transformación
    osg::ref_ptr<osg::MatrixTransform> mat_trans1_i = new osg::MatrixTransform;
    mat_trans1_i->setMatrix ( osg::Matrix::translate ( 0.0f , -0.0795f , -0.24f ) );
       //se modifica la posición de la matriz y se añade el fémur como hijo
    mat_trans1_i->addChild ( model1.get ( ) );


    //TIBIA
    osg::ref_ptr<osg::MatrixTransform> mat_trans2_i = new osg::MatrixTransform;
    mat_trans2_i->setMatrix ( osg::Matrix::translate ( 0.0f , -0.0795f , -0.6275f )
);
    mat_trans2_i->addChild ( model2r.get ( ) );


    //FOOT
    osg::ref_ptr<osg::MatrixTransform> mat_trans3_i = new osg::MatrixTransform;
    mat_trans3_i->setMatrix ( osg::Matrix::translate ( 0.0f , -0.0795f , -0.989f ) );
    mat_trans3_i->addChild ( model3r.get ( ) );


    //TOE
    osg::ref_ptr<osg::MatrixTransform> mat_trans4_i = new osg::MatrixTransform;
    mat_trans4_i->setMatrix ( osg::Matrix::translate ( 0.1102f , -0.0795f , -1.0376f
) );
    mat_trans4_i->addChild ( model4r.get ( ) );


    //Se crea un grupo tipo Switch para poder ocultar la cadera

    osg::ref_ptr<osg::Switch> root = new osg::Switch;

    root->addChild ( mat_trans0.get ( ) , true );
    root->addChild ( mat_trans1.get ( ) , true );
    root->addChild ( mat_trans2.get ( ) , true );
```

```
    root->addChild ( mat_trans2sec.get ( ) , true );
    root->addChild ( mat_trans3.get ( ) , true );
    root->addChild ( mat_trans3sec.get ( ) , true );
    root->addChild ( mat_trans4.get ( ) , false );


    root->addChild ( mat_trans0_i.get ( ) , false );
    root->addChild ( mat_trans1_i.get ( ) , true );
    root->addChild ( mat_trans2_i.get ( ) , true );
    root->addChild ( mat_trans3_i.get ( ) , true );
    root->addChild ( mat_trans4_i.get ( ) , false );



    //Posición y orientación de la cámara por defecto

    osgGA::TrackballManipulator *manipulador ( new osgGA::TrackballManipulator );

       //Perspective plane view
    //manipulador->setHomePosition ( osg::Vec3 ( 5.0 , -2.0 , 2.0 ) ,          // eye
    //                               osg::Vec3 ( -4.0 , 2.0 , -0.5 ) ,         //
center
    //                               osg::Vec3 ( 0.0 , 0.0 , 1.0 ) );         // up

       //Sagital plane view
       manipulador->setHomePosition (osg::Vec3 (0.0, -4.0, 0.5),  // eye
                                                          osg::Vec3 (0.0, 0.0, 0.5),
// center
                                                          osg::Vec3 (0.0, 0.0, 1.0));
// up

       //Frontal plane view
   /* manipulador->setHomePosition ( osg::Vec3 ( 3.4 , -3.0 , -0.5 ) ,         // eye
                                  osg::Vec3 ( 0.0 , 0.0 , -0.5) ,         // center
                                  osg::Vec3 ( 0.0 , 0.0 , 1 ) );         // up*/


    //Creación de la escena y visualización

    osgViewer::Viewer viewer;
    viewer.setSceneData ( root.get ( ) );
    viewer.setCameraManipulator ( manipulador );
    viewer.addEventHandler ( new homeSwitcher ( manipulador ) );
    viewer.addEventHandler ( new osgViewer::WindowSizeHandler );
    viewer.addEventHandler ( new osgViewer::StatsHandler );
    viewer.getCamera ( )->setClearColor ( osg::Vec4 ( 0 , 0 , 0 , 1 ) );
    //viewer.setUpViewInWindow ( 192 , 144 , 960 , 720 ); //ordendador de sobremesa.
Pantalla centrada
       viewer.setUpViewInWindow (130, 50, 960, 640); //Centrado de la animación en la
pantalla del portátil


    //Iluminación

    float pw = 1.3;
    const int nl = 2;

    osg::StateSet *ss = root.get ( )->getOrCreateStateSet ( );
```

```
osg::Vec4::value_type x[nl] = { 5.0 , -5.0 };
osg::Vec4::value_type y[nl] = { -4.0 , 3.0 };
osg::Vec4::value_type z[nl] = { 1.2 , 1.2 };

for ( int i = 0; i < nl; i++ )
{
    osg::Light *light = new osg::Light ( i );
    osg::LightSource *lightSource = new osg::LightSource ( );

    light->setAmbient ( osg::Vec4 ( 0.1 , 0.1 , 0.1 , 1.0 ) );
    light->setDiffuse ( osg::Vec4 ( pw*1.000 , pw*0.9804 , pw*0.9412 , 1.0 ) );
    light->setSpecular ( osg::Vec4 ( 0.2 , 0.2 , 0.2 , 1.0 ) );

    light->setPosition ( osg::Vec4 ( x[i] , y[i] , z[i] , 1.0 ) );

    lightSource->setLight ( light );
    lightSource->setStateSetModes ( *ss , osg::StateAttribute::ON );

    root.get ( )->addChild ( lightSource );
}



/******************     LECTURA MATRICES     ******************/


//Lectura ficheros ip: datos posición inicio de cada modelo

std::ifstream fip ( "ip.dat" );
std::vector<std::vector<int>> ip ( 2 );

if ( fip.is_open ( ) )
{
    for ( int j = 0; j < npoints; j++ )
    {
        int elem;
        fip >> elem;
        ip[0].push_back ( elem );
        fip >> elem;
        ip[1].push_back ( elem );
    }
    fip.close ( );
}

//Lectura ficheros iv: datos vectores inicio de cada modelo

std::ifstream fiv ( "iv.dat" ); //Lectura de archivo
std::vector<std::vector<int>> iv ( 2 );

if ( fiv.is_open ( ) )
{
    for ( int i = 0; i < nvector; i++ )
    {
        int elem1;
        fiv >> elem1;
        iv[0].push_back ( elem1 );
        fiv >> elem1;
        iv[1].push_back ( elem1 );
```

```
        }
        fiv.close ( );
    }


    //Lectura ficheros matrices

    std::ifstream fq ( "qProj.dat" ); //Lectura de archivo

    std::vector<std::vector<double> > q ( nfils );


    for ( int i = 0; i < nfils; i++ )
    {
        q[i].resize ( ncols );
    }


    if ( fq.is_open ( ) )
    {
        std::cout << "Abriendo archivo de captura de movimiento...\n" << endl;
        //fread_s (q, ncols*sizeof(double), sizeof(double), ncols, f1);
        for ( int i = 0; i < nfils; i++ )
        {
            for ( int j = 0; j < ncols; j++ )
            {

                fq >> q[i][j];

            }
        }

        fq.close ( );
    }


    std::ifstream fqder ( "qpProj.dat" ); //Lectura de archivo

    std::vector<std::vector<double> > qder ( nfils );


    for ( int i = 0; i < nfils; i++ )
    {
        qder[i].resize ( ncols );
    }


    if ( fqder.is_open ( ) )
    {
        std::cout << "Abriendo archivo de captura de movimiento en derivadas...\n" <<
endl;

        for ( int i = 0; i < nfils; i++ )
        {
            for ( int j = 0; j < ncols; j++ )
            {

                fqder >> qder[i][j];
```

```
            }
        }

        fqder.close ( );
    }

       //lectura archivo fuerza pisada

       std::ifstream f_fpl1 ("FPL1.dat"); //Lectura de archivo
       std::vector<std::vector<double> > FPL1 ( nfils );


    for ( int i = 0; i < nfils; i++ )
    {
        FPL1[i].resize ( 8 ); //posee ocho columnas
    }


    if ( f_fpl1.is_open ( ) )
    {
        std::cout << "Abriendo archivo de fuerza en la pisada. Pie en el aire o pie
apoyado...\n" << endl;

        for ( int i = 0; i < nfils; i++ )
        {
                    for (int j = 0; j < 8; j++)
                    {


                           f_fpl1 >> FPL1 [i][j]; //solo se necesita la tercera co-
lumna
                    }

         }

        f_fpl1.close ( );
    }

       std::ifstream f_fpl2 ("FPL2.dat"); //Lectura de archivo
       std::vector<std::vector<double> > FPL2 (nfils);


       for (int i = 0; i < nfils; i++)
       {
              FPL2 [i].resize (8); //posee ocho columnas
       }


       if (f_fpl2.is_open ())
       {
              std::cout << "Abriendo archivo de fuerza en la pisada. Pie en el aire o
pie apoyado...\n" << endl;

              for (int i = 0; i < nfils; i++)
              {
                     for (int j = 0; j < 8; j++)
                     {
```

```
                        f_fpl2 >> FPL2 [i][j];
                }

        }

        f_fpl2.close ();
    }

    /**************      BUCLE MOVIMIENTO     ***********/


    //vectores correspondientes a matrices de transformación de dimensiones 16x1
piernas derecha e izquierda

    double m0[16] = { 0 };        //pelvis
    double m1[16] = { 0 };              //matriz por columnas correspondiente al fé-
mur
    double m2[16] = { 0 };              //tibia
    double m3[16] = { 0 };              //tobillo
    double m4[16] = { 0 };              //pie

  //matrices derivadas
    double m1d[16] = { 0 };             //matriz por columnas correspondiente al fé-
mur
    double m2d[16] = { 0 };             //tibia
    double m3d[16] = { 0 };             //tobillo
    double m4d[16] = { 0 };             //pie


    double m1_i[16] = { 0 };                    //matriz por columnas correspondiente
al fémur izq
    double m2_i[16] = { 0 };                    //tibia izq
    double m3_i[16] = { 0 };                    //tobillo izq
    double m4_i[16] = { 0 };                    //pie izq

    //femur
    double W11[9] = { 0 };
       //matriz por columnas correspondiente a la tibia
    double W12[9] = { 0 };

    double Wantisim[9] = { 0 };


    double erL11[3] = { 0 }; //femur
    double erL12[3] = { 0 }; //tibia

    double erLOptimo11[3] = { 0 };
    double erLOptimo12[3] = { 0 };

    double erGNuevo11[3] = { 0 };
    double erGNuevo12[3] = { 0 };

    double u11_p[3] = { 0 };
    double u12_p[3] = { 0 };

    double temporal1 = 0;
    double temporal2 = 0;
```

```
        double norma_u11p , norma_u12p = 0;

        double u11_pn[3] = { 0 }; //vectores proyectados sobre eje de la rodilla
        double u12_pn[3] = { 0 };

        double theta_rodilla = 0;
        double theta_tobillo_dcho = 0;
        double theta_tobillo_izq = 0;

        double temp_tob_iz [3] = { 0 }; //utilizadas para almacenar temporalmente el
producto vectorial de los vectores u
        double temp_tob [3] = { 0 };
        double temp[3] = { 0 };

        double norma_temp = 0;
        double norma_temp_tobillo = 0; //almacenar la norma del producto vectorial
para calcular el angulo del tobillo
        double norma_temp_tobillo_iz = 0;

        double identity[9] = { 0 };
        double antisimetrica_rodilla[9] = { 0 };
        double mat_trans_relativa_rodilla[9] = { 0 };

        double Atibia[9] = { 0 };

        double Atibia_d[16] = { 0 };

        double Atobillo[16] = { 0 };

        double square_antisim[9] = { 0 };

        double theta_rodilla_iz = 0;
        double theta_rodilla_dcha = 0;

        double A0 [9] = { 0 }; //esta matriz la podemos leer del archivo matrizA0.dat
para no tener que escribirla a mano

        double theta_tibia = 0;
        double theta_tibia_iz = 0;
        double theta_pie_iz = 0;
        double theta_pie = 0;

        float Xcop1, Xcop2 = 0; //posición centro de presión de los pies (archivos
FPL)
        float delta1, delta2, delta1_iz, delta2_iz = 0; //distancia al tobillo del
centro de aplicación de fuerza en las placas. Condiciona pie sobre puntera o plano

        status_t rFSR, lFSR;

        //BUCLE LECTURA MATRIZ DE MOVIMIENTO QPROJ

        for (int filas = 0; filas < nfils; filas++)
        {

         viewer.realize ( );   //actualización de animación
```

```
   /*************************      PELVIS      ************************/


        int pointpel = 12;               //punto distal
        size_t pospel = std::distance ( ip[0].begin ( ) , std::find ( ip[0].begin ( )
, ip[0].end ( ) , pointpel ) );
        double rPointpel[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPointpel[i] = q[filas][ip[1][pospel] + ( i - 1 )];


        int vector_upel = 11;
        int vector_vpel = vector_upel + 1;
        size_t vec_upel = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_upel ) );
        size_t vec_vpel = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_vpel ) );
        double uVectorpel[3] = { 0 };
        double vVectorpel[3] = { 0 };


        for ( int j = 0; j < 3; j++ )
        {
            uVectorpel[j] = q[filas][iv[1][vec_upel] + ( j - 1 )];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
            vVectorpel[j] = q[filas][iv[1][vec_vpel] + ( j - 1 )];
        }



        int point_proxpel = 11;           //punto proximal
        size_t pos_proxpel = std::distance ( ip[0].begin ( ) , std::find (
ip[0].begin ( ) , ip[0].end ( ) , point_proxpel ) );
        double rPoint_proxpel[3] = { 0 };



        for ( int i = 0; i < 3; i++ )
            rPoint_proxpel[i] = q[filas][ip[1][pos_proxpel] + ( i - 1 )];

       //vector W aquí resulta del producto vectorial por no ser un hueso esbelto

        double wVectorpel[3] = { 0 };

        wVectorpel[0] = uVectorpel[1] * vVectorpel[2] - uVectorpel[2] * vVec-
torpel[1];
        wVectorpel[1] = uVectorpel[2] * vVectorpel[0] - uVectorpel[0] * vVec-
torpel[2];
        wVectorpel[2] = uVectorpel[0] * vVectorpel[1] - uVectorpel[1] * vVec-
torpel[0];


        //la matriz de 16x1 tendrá los elementos, "comenzando por 0", [3], [7] y [11]
como ceros y el elemento [15] como un 1 EN TODOS LOS SÓLIDOS
```

```
        m0[3] = m0[7] = m0[11] = 0;
        m0[15] = 1;

        //Entre [0] y [3] es el vector u; entre [4] y [7] es el vector v; entre [8] y
[11] es el vector w; entre [12] y [15] es el vector del punto proximal

        for ( int i = 0; i < 3; i++ )
        {
            m0[i] = uVectorpel[i]; //factores de escala según archivos matscale y
ppoint
            m0[i + 4] = vVectorpel[i];
            m0[i + 8] = wVectorpel[i];
            m0[i + 12] = rPoint_proxpel[i];

        }


   /**********************      Pierna Derecha     ************************/


   /************************      FEMUR      *********************/


              int pointf = 111;                 //punto FEMUR de la tabla de ip para
obtener correspondencia en q
              size_t posf = std::distance (ip[0].begin (), std::find (ip[0].begin (),
ip[0].end (), pointf));
               double rPointf[3] = { 0 };


              for (int i = 0; i < 3; i++)
                     rPointf[i] = q[filas][ip[1][posf] + (i - 1)];


              int vector_uf = 111;       //punto FEMUR vector u de la tabla de iv
para obtener correspondencia en q. El femur es el 111, se busca su correspondiente
              int vector_vf = vector_uf + 1; //vector v correspondiente al FEMUR
              size_t vec_uf = std::distance (iv[0].begin (), std::find (iv[0].begin
(), iv[0].end (), vector_uf));
              size_t vec_vf = std::distance (iv[0].begin (), std::find (iv[0].begin
(), iv[0].end (), vector_vf));
              double uVectorf[3] = { 0 };
              double vVectorf[3] = { 0 };


              for (int j = 0; j < 3; j++)
              {
                     uVectorf[j] = q[filas][iv[1][vec_uf] + (j - 1)];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
                     vVectorf[j] = q[filas][iv[1][vec_vf] + (j - 1)];
              }


              // se calcula el punto proximal del fémur para poder calcular la norma
entre el punto distal y el proximal
```

```
		int point_proxf = 12;                          //punto FEMUR de la tabla
de ip para obtener correspondencia en q
		size_t pos_proxf = std::distance (ip[0].begin (), std::find
(ip[0].begin (), ip[0].end (), point_proxf));
		double rPoint_proxf[3] = { 0 };



		for (int i = 0; i < 3; i++)
			rPoint_proxf[i] = q[filas][ip[1][pos_proxf] + (i - 1)];


		//Vectores para calcular la norma con la que se forma el vector W

		double U[3] = { 0 };
		double Ut[3] = { 0 };
		double Utb[3] = { 0 };
		double Up[3] = { 0 };

		double norma_distal_f=0;
		double norma_distal_t = 0;
		double norma_distal_tb = 0;
		double norma_distal_p = 0;

		//Vector intermediario para cálculo de la norma

		for (int j = 0; j<3; j++)
			U[j] = rPointf[j] - rPoint_proxf[j];


		norma_distal_f = norma.Norm (U);

		//vector W

		double wVectorf[3] = { 0 };

		for (int i = 0; i < 3; i++)
			wVectorf[i] = U[i] / (-norma_distal_f);


		//la matriz de 16x1 tendrá los elementos, "comenzando por 0", [3], [7]
y [11] como ceros y el elemento [15] como un 1 EN TODOS LOS SÓLIDOS

		m1[3] = m1[7] = m1[11] = 0;
		m1[15] = 1;

		//Entre [0] y [3] es el vector u; entre [4] y [7] es el vector v; entre
[8] y [11] es el vector w; entre [12] y [15] es el vector del punto proximal

		for (int i = 0; i < 3; i++)
		{
			m1[i] = uVectorf[i]; //factores de escala según archivos
matscale y ppoint
		m1[i + 4] = vVectorf[i];
		m1[i + 8] = wVectorf[i];
		m1[i + 12] = rPoint_proxf[i];

		}
```

```
        /************************      FEMUR DERIVADO     **********************/

        //vectores u y v derivados

        double rPointfd[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPointfd[i] = qder[filas][ip[1][posf] + ( i - 1 )];

        double uVectorfd[3] = { 0 };
        double vVectorfd[3] = { 0 };

        for ( int j = 0; j < 3; j++ )
        {
            uVectorfd[j] = qder[filas][iv[1][vec_uf] + ( j - 1 )];
      //para que empiece justo en la posición que indica la tabla (añadir j-1)
            vVectorfd[j] = qder[filas][iv[1][vec_vf] + ( j - 1 )];
        }

        // se calcula el punto proximal del fémur para poder calcular la norma entre
el punto distal y el proximal

        double rPoint_proxfd[3] = { 0 };

        for ( int i = 0; i < 3; i++ )
            rPoint_proxfd[i] = qder[filas][ip[1][pos_proxf] + ( i - 1 )];

        //Vectores para calcular la norma con la que se forma el vector W

        double Ud[3] = { 0 };
        double Utd[3] = { 0 };


        //Vector intermediario para cálculo de la norma

        for ( int j = 0; j<3; j++ )
            Ud[j] = rPointfd[j] - rPoint_proxfd[j];


      //vector W

        double wVectorfd[3] = { 0 };

        for ( int i = 0; i < 3; i++ )
            wVectorfd[i] = Ud[i] / ( -norma_distal_f );


        //la matriz de 16x1 tendrá los elementos, "comenzando por 0", [3], [7] y [11]
como ceros y el elemento [15] como un 1 EN TODOS LOS SÓLIDOS

        m1d[3] = m1d[7] = m1d[11] = 0;
        m1d[15] = 1;

        //Entre [0] y [3] es el vector u; entre [4] y [7] es el vector v; entre [8] y
[11] es el vector w; entre [12] y [15] es el vector del punto proximal
```

```
        for ( int i = 0; i < 3; i++ )
        {
            m1d[i] = uVectorfd[i]; //factores de escala según archivos matscale y
ppoint
            m1d[i + 4] = vVectorfd[i];
            m1d[i + 8] = wVectorfd[i];
            m1d[i + 12] = rPoint_proxfd[i];

        }


      /*************************      TIBIA   **********************/


            int pointt = 121;                 //punto TIBIA de la tabla de ip para
obtener correspondencia en q (punto distal)
            size_t post = std::distance (ip[0].begin (), std::find (ip[0].begin (),
ip[0].end (), pointt));
            double rPointt[3] = { 0 };


            for (int i = 0; i < 3; i++)
                  rPointt[i] = q[filas][ip[1][post] + (i - 1)];


            int vector_ut = 121;           //punto TIBIA vector u de la tabla de iv
para obtener correspondencia en q. La TIBIA es el 121, se busca su correspondiente
            int vector_vt = vector_ut + 1;  //vector v correspondiente al TIBIA
            size_t vec_ut = std::distance (iv[0].begin (), std::find (iv[0].begin
(), iv[0].end (), vector_ut));
            size_t vec_vt = std::distance (iv[0].begin (), std::find (iv[0].begin
(), iv[0].end (), vector_vt));
            double uVectort[3] = { 0 };
            double vVectort[3] = { 0 };


            for (int j = 0; j < 3; j++)
            {
                  uVectort[j] = q[filas][iv[1][vec_ut] + (j - 1)];
      //para que empiece justo en la posición que indica la tabla (añadir j-1)
                  vVectort[j] = q[filas][iv[1][vec_vt] + (j - 1)];
            }

            //se calcula el punto proximal de la tibia para poder calcular la norma
entre el punto distal y el proximal

            int point_proxt = 111;      //coincide con el punto distal del fémur
            size_t pos_proxt = std::distance (ip[0].begin (), std::find
(ip[0].begin (), ip[0].end (), point_proxt));
            double rPoint_proxt[3] = { 0 };


            for (int i = 0; i < 3; i++)
                  rPoint_proxt[i] = q[filas][ip[1][pos_proxt] + (i - 1)];
```

```
            for (int j = 0; j<3; j++)
                  Ut[j] = rPointt[j] - rPoint_proxt[j];

            norma_distal_t = norma.Norm (Ut);

        double wVectort[3] = { 0 };

            for (int i = 0; i < 3; i++)
                  wVectort[i] = Ut[i] / (-norma_distal_t);


            m2[3] = m2[7] = m2[11] = 0;
            m2[15] = 1;

            for (int i = 0; i < 3; i++)
            {
            m2[i] = uVectort[i];
            m2[i + 4] = vVectort[i];
            m2[i + 8] = wVectort[i];
            m2[i + 12] = rPoint_proxt[i];

            }


            //Cálculo del ángulo de la tibia Proyección eje z del vector u sobre el
eje de la rodilla

            theta_tibia = asin (uVectort[2]); // arcsen de la componente z del vec-
tor u de la tibia


        /*********************     TIBIA DERIVADA    ******************/

        //vectores u y v derivados

        double rPointtd[3] = { 0 };

        for ( int i = 0; i < 3; i++ )
            rPointtd[i] = qder[filas][ip[1][post] + ( i - 1 )];

        double uVectortd[3] = { 0 };
        double vVectortd[3] = { 0 };


        for ( int j = 0; j < 3; j++ )
        {
            uVectortd[j] = qder[filas][iv[1][vec_ut] + ( j - 1 )];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
            vVectortd[j] = qder[filas][iv[1][vec_vt] + ( j - 1 )];
        }

        //se calcula el punto proximal de la tibia para poder calcular la norma entre
el punto distal y el proximal


        double rPoint_proxtd[3] = { 0 };
```

```
        for ( int i = 0; i < 3; i++ )
            rPoint_proxtd[i] = qder[filas][ip[1][pos_proxt] + ( i - 1 )];


        for ( int j = 0; j<3; j++ )
            Utd[j] = rPointtd[j] - rPoint_proxtd[j];


        double wVectortd[3] = { 0 };

        for ( int i = 0; i < 3; i++ )
            wVectortd[i] = Utd[i] / ( -norma_distal_t );


        m2d[3] = m2d[7] = m2d[11] = 0;
        m2d[15] = 1;

        for ( int i = 0; i < 3; i++ )
        {
            m2d[i] = uVectortd[i];
            m2d[i + 4] = vVectortd[i];
            m2d[i + 8] = wVectortd[i];
            m2d[i + 12] = rPoint_proxtd[i];

        }


        /************************      TOBILLO      **********************/


              int pointtb = 131;                //punto TOBILLO de la tabla de ip
para obtener correspondencia en q
              size_t postb = std::distance (ip[0].begin (), std::find (ip[0].begin
(), ip[0].end (), pointtb));
              double rPointtb[3] = { 0 };


              for (int i = 0; i < 3; i++)
                     rPointtb[i] = q[filas][ip[1][postb] + (i - 1)];


              int vector_utb = 131;             //punto TOBILLO vector u de la tabla
de iv para obtener correspondencia en q. El TOBILLO es el 131, se busca su correspon-
diente
              int vector_vtb = vector_utb + 1; //vector v correspondiente al TOBILLO
              size_t vec_utb = std::distance (iv[0].begin (), std::find (iv[0].begin
(), iv[0].end (), vector_utb));
              size_t vec_vtb = std::distance (iv[0].begin (), std::find (iv[0].begin
(), iv[0].end (), vector_vtb));
              double uVectortb[3] = { 0 };
              double vVectortb[3] = { 0 };


              for (int j = 0; j < 3; j++)
              {
                     uVectortb[j] = q[filas][iv[1][vec_utb] + (j - 1)];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
```

```
			vVectortb[j] = q[filas][iv[1][vec_vtb] + (j - 1)];
		}


		//se calcula el punto proximal del tobillo para poder calcular la norma entre el punto distal y el proximal

		int point_proxtb = 121;
		size_t pos_proxtb = std::distance (ip[0].begin (), std::find (ip[0].begin (), ip[0].end (), point_proxtb));

		double rPoint_proxtb[3] = { 0 };


		for (int i = 0; i < 3; i++)
			rPoint_proxtb[i] = q[filas][ip[1][pos_proxtb] + (i - 1)];


		for (int j = 0; j<3; j++)
			Utb[j] = rPointtb[j] - rPoint_proxtb[j];

		norma_distal_tb = norma.Norm (Utb);

		double wVectortb[3] = { 0 };

		for (int i = 0; i < 3; i++)
			wVectortb[i] = Utb[i] / (-norma_distal_tb);


		m3[3] = m3[7] = m3[11] = 0;
		m3[15] = 1;

		for (int i = 0; i < 3; i++)
		{
		m3[i] = uVectortb[i];
		m3[i + 4] = vVectortb[i];
		m3[i + 8] = wVectortb[i];
		m3[i + 12] = rPoint_proxtb[i];

		}

		theta_pie = asin (uVectortb [2]);




		/******************   PUNTA PIE DERECHO   ***************/


		int pointp = 171;			//punto punta del PIE de la tabla de ip para obtener correspondencia en q
		size_t posp = std::distance (ip[0].begin (), std::find (ip[0].begin (), ip[0].end (), pointp));
		double rPointp[3] = { 0 };

		for (int i = 0; i < 3; i++)
			rPointp[i] = q[filas][ip[1][posp] + (i - 1)];
```

```
		int vector_up = 171;		//punto PIE vector u de la tabla de iv para obtener correspondencia en q. El PIE es el 171, se busca su correspondiente
		int vector_vp = vector_up + 1; //vector v correspondiente al PIE
		size_t vec_up = std::distance (iv[0].begin (), std::find (iv[0].begin (), iv[0].end (), vector_up));
		size_t vec_vp = std::distance (iv[0].begin (), std::find (iv[0].begin (), iv[0].end (), vector_vp));
		double uVectorp[3] = { 0 };
		double vVectorp[3] = { 0 };

		for (int j = 0; j < 3; j++)
		{
			uVectorp[j] = q[filas][iv[1][vec_up] + (j - 1)];
	//para que empiece justo en la posición que indica la tabla (añadir j-1)
			vVectorp[j] = q[filas][iv[1][vec_vp] + (j - 1)];
		}


		//se calcula el punto proximal del pie para poder calcular la norma entre el punto distal y el proximal

		int point_proxp = 131;				//el punto 131 representa la articulación del pie derecho
		size_t pos_proxp = std::distance (ip[0].begin (), std::find (ip[0].begin (), ip[0].end (), point_proxp));
		double rPoint_proxp[3] = { 0 };

		for (int i = 0; i < 3; i++)
			rPoint_proxp[i] = q[filas][ip[1][pos_proxp] + (i - 1)];


		double wVectorp[3] = { 0 };

	wVectorp[0] = uVectorp[1] * vVectorp[2] - uVectorp[2] * vVectorp[1];
	wVectorp[1] = uVectorp[2] * vVectorp[0] - uVectorp[0] * vVectorp[2];
	wVectorp[2] = uVectorp[0] * vVectorp[1] - uVectorp[1] * vVectorp[0];


		m4[3] = m4[7] = m4[11] = 0;
		m4[15] = 1;

		for (int i = 0; i < 3; i++)
		{
		m4[i] = uVectorp[i];
		m4[i + 4] = vVectorp[i];
		m4[i + 8] = wVectorp[i];
		m4[i + 12] = rPoint_proxp[i];

		}

    /******************	Pierna Izquierda	*********************/


	/******************		FEMUR		******************/


	int pointf_i = 141;			//punto FEMUR de la tabla de ip para obtener correspondencia en q
```

```
        size_t posf_i = std::distance ( ip[0].begin ( ) , std::find ( ip[0].begin ( )
, ip[0].end ( ) , pointf_i ) );
        double rPointf_i[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPointf_i[i] = q[filas][ip[1][posf_i] + ( i - 1 )];


        int vector_uf_i = 141;           //punto FEMUR vector u de la tabla de iv
para obtener correspondencia en q. El femur es el 111, se busca su correspondiente
        int vector_vf_i = vector_uf_i + 1; //vector v correspondiente al FEMUR
        size_t vec_uf_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_uf_i ) );
        size_t vec_vf_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_vf_i ) );
        double uVectorf_i[3] = { 0 };
        double vVectorf_i[3] = { 0 };


        for ( int j = 0; j < 3; j++ )
        {
            uVectorf_i[j] = q[filas][iv[1][vec_uf_i] + ( j - 1 )];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
            vVectorf_i[j] = q[filas][iv[1][vec_vf_i] + ( j - 1 )];
        }


        // se calcula el punto proximal del fémur para poder calcular la norma entre
el punto distal y el proximal

        int point_proxf_i = 13;                    //punto FEMUR de la tabla de ip
para obtener correspondencia en q
        size_t pos_proxf_i = std::distance ( ip[0].begin ( ) , std::find (
ip[0].begin ( ) , ip[0].end ( ) , point_proxf_i ) );
        double rPoint_proxf_i[3] = { 0 };



        for ( int i = 0; i < 3; i++ )
            rPoint_proxf_i[i] = q[filas][ip[1][pos_proxf_i] + ( i - 1 )];


        //Vectores para calcular la norma con la que se forma el vector W

        double U_i[3] = { 0 };
        double Ut_i[3] = { 0 };
        double Utb_i[3] = { 0 };
        double Up_i[3] = { 0 };

        double norma_distal_f_i = 0;
        double norma_distal_t_i = 0;
        double norma_distal_tb_i = 0;
        double norma_distal_p_i = 0;
```

```
//Vector intermediario para cálculo de la norma

        for ( int j = 0; j<3; j++ )
            U_i[j] = rPointf_i[j] - rPoint_proxf_i[j];


        norma_distal_f_i = norma.Norm ( U_i );

        //vector W

        double wVectorf_i[3] = { 0 };

        for ( int i = 0; i < 3; i++ )
            wVectorf_i[i] = U_i[i] / ( -norma_distal_f_i );


        //la matriz de 16x1 tendrá los elementos, "comenzando por 0", [3], [7] y [11]
como ceros y el elemento [15] como un 1 EN TODOS LOS SÓLIDOS

        m1_i[3] = m1_i[7] = m1_i[11] = 0;
        m1_i[15] = 1;

        //Entre [0] y [3] es el vector u; entre [4] y [7] es el vector v; entre [8] y
[11] es el vector w; entre [12] y [15] es el vector del punto proximal

        for ( int i = 0; i < 3; i++ )
        {
            m1_i[i] = uVectorf_i[i]; //factores de escala según archivos matscale y
ppoint
            m1_i[i + 4] = vVectorf_i[i]; //escala negativa para hacer simetría res-
pecto al eje Y
            m1_i[i + 8] = wVectorf_i[i];
            m1_i[i + 12] = rPoint_proxf_i[i];

        }


        /******************          TIBIA      *************/



        int pointt_i = 151;             //punto TIBIA de la tabla de ip para obtener
correspondencia en q (punto distal)
        size_t post_i = std::distance ( ip[0].begin ( ) , std::find ( ip[0].begin ( )
, ip[0].end ( ) , pointt_i ) );
        double rPointt_i[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPointt_i[i] = q[filas][ip[1][post_i] + ( i - 1 )];


        int vector_ut_i = 151;              //punto TIBIA vector u de la tabla de iv
para obtener correspondencia en q. La TIBIA es el 121, se busca su correspondiente
        int vector_vt_i = vector_ut_i + 1;  //vector v correspondiente al TIBIA
        size_t vec_ut_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_ut_i ) );
```

```
        size_t vec_vt_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_vt_i ) );
        double uVectort_i[3] = { 0 };
        double vVectort_i[3] = { 0 };


        for ( int j = 0; j < 3; j++ )
        {
            uVectort_i[j] = q[filas][iv[1][vec_ut_i] + ( j - 1 )];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
            vVectort_i[j] = q[filas][iv[1][vec_vt_i] + ( j - 1 )];
        }

        //se calcula el punto proximal de la tibia para poder calcular la norma entre
el punto distal y el proximal

        int point_proxt_i = 141;      //coincide con el punto distal del fémur
        size_t pos_proxt_i = std::distance ( ip[0].begin ( ) , std::find (
ip[0].begin ( ) , ip[0].end ( ) , point_proxt_i ) );
        double rPoint_proxt_i[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPoint_proxt_i[i] = q[filas][ip[1][pos_proxt_i] + ( i - 1 )];

        for ( int j = 0; j<3; j++ )
            Ut_i[j] = rPointt_i[j] - rPoint_proxt_i[j];

        norma_distal_t_i = norma.Norm ( Ut_i );

        double wVectort_i[3] = { 0 };

        for ( int i = 0; i < 3; i++ )
            wVectort_i[i] = Ut_i[i] / ( -norma_distal_t_i );


        m2_i[3] = m2_i[7] = m2_i[11] = 0;
        m2_i[15] = 1;

        for ( int i = 0; i < 3; i++ )
        {
            m2_i[i] = uVectort_i[i] ;
            m2_i[i + 4] = vVectort_i[i];
            m2_i[i + 8] = wVectort_i[i] ;
            m2_i[i + 12] = rPoint_proxt_i[i];

        }


             theta_tibia_iz = asin (uVectort_i [2]);

             /*angulo_tibia << std::setw (10) << std::setprecision (8) << std::fixed
<< theta_tibia<<"\t"<<theta_tibia_iz << endl;*/
```

```
        /*************************   TOBILLO   ************************/


        int pointtb_i = 161;                    //punto TOBILLO de la tabla de ip
para obtener correspondencia en q
        size_t postb_i = std::distance ( ip[0].begin ( ) , std::find ( ip[0].begin (
) , ip[0].end ( ) , pointtb_i ) );
        double rPointtb_i[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPointtb_i[i] = q[filas][ip[1][postb_i] + ( i - 1 )];


        int vector_utb_i = 161;          //punto TOBILLO vector u de la tabla de iv
para obtener correspondencia en q. El TOBILLO es el 131, se busca su correspondiente
        int vector_vtb_i = vector_utb_i + 1; //vector v correspondiente al TOBILLO
        size_t vec_utb_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin
( ) , iv[0].end ( ) , vector_utb_i ) );
        size_t vec_vtb_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin
( ) , iv[0].end ( ) , vector_vtb_i ) );
        double uVectortb_i[3] = { 0 };
        double vVectortb_i[3] = { 0 };


        for ( int j = 0; j < 3; j++ )
        {
            uVectortb_i[j] = q[filas][iv[1][vec_utb_i] + ( j - 1 )];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
            vVectortb_i[j] = q[filas][iv[1][vec_vtb_i] + ( j - 1 )];
        }


        //se calcula el punto proximal del tobillo para poder calcular la norma entre
el punto distal y el proximal

        int point_proxtb_i = 151;
        size_t pos_proxtb_i = std::distance ( ip[0].begin ( ) , std::find (
ip[0].begin ( ) , ip[0].end ( ) , point_proxtb_i ) );
        double rPoint_proxtb_i[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPoint_proxtb_i[i] = q[filas][ip[1][pos_proxtb_i] + ( i - 1 )];


        for ( int j = 0; j<3; j++ )
            Utb_i[j] = rPointtb_i[j] - rPoint_proxtb_i[j];

        norma_distal_tb_i = norma.Norm ( Utb_i );

        double wVectortb_i[3] = { 0 };

        for ( int i = 0; i < 3; i++ )
            wVectortb_i[i] = Utb_i[i] / ( -norma_distal_tb_i );


        m3_i[3] = m3_i[7] = m3_i[11] = 0;
```

```
        m3_i[15] = 1;

        for ( int i = 0; i < 3; i++ )
        {
            m3_i[i] = uVectortb_i[i];
            m3_i[i + 4] = vVectortb_i[i]; //escala negativa para simetría respecto
eje Y
            m3_i[i + 8] = wVectortb_i[i];
            m3_i[i + 12] = rPoint_proxtb_i[i];

        }

             theta_pie_iz = asin (uVectortb_i [2]);
             angulo_tibia << std::setw (10) << std::setprecision (8) << std::fixed
<< theta_tibia << "\t" << theta_tibia_iz << "\t" << theta_pie << "\t" <<
theta_pie_iz<< endl;

        /**************   PUNTA PIE IZQ   ******************/


        int pointp_i = 181;             //punto punta del PIE de la tabla de ip para
obtener correspondencia en q
        size_t posp_i = std::distance ( ip[0].begin ( ) , std::find ( ip[0].begin ( )
, ip[0].end ( ) , pointp_i ) );
        double rPointp_i[3] = { 0 };


        for ( int i = 0; i < 3; i++ )
            rPointp_i[i] = q[filas][ip[1][posp_i] + ( i - 1 )];


        int vector_up_i = 181;          //punto PIE vector u de la tabla de iv para
obtener correspondencia en q. El PIE es el 171, se busca su correspondiente
        int vector_vp_i = vector_up_i + 1; //vector v correspondiente al PIE
        size_t vec_up_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_up_i ) );
        size_t vec_vp_i = std::distance ( iv[0].begin ( ) , std::find ( iv[0].begin (
) , iv[0].end ( ) , vector_vp_i ) );
        double uVectorp_i[3] = { 0 };
        double vVectorp_i[3] = { 0 };


        for ( int j = 0; j < 3; j++ )
        {
            uVectorp_i[j] = q[filas][iv[1][vec_up_i] + ( j - 1 )];
       //para que empiece justo en la posición que indica la tabla (añadir j-1)
            vVectorp_i[j] = q[filas][iv[1][vec_vp_i] + ( j - 1 )];
        }


        //se calcula el punto proximal del pie para poder calcular la norma entre el
punto distal y el proximal

        int point_proxp_i = 161;                        //el punto 131 representa
la articulación del pie derecho
        size_t pos_proxp_i = std::distance ( ip[0].begin ( ) , std::find (
ip[0].begin ( ) , ip[0].end ( ) , point_proxp_i ) );
        double rPoint_proxp_i[3] = { 0 };
```

```
        for ( int i = 0; i < 3; i++ )
            rPoint_proxp_i[i] = q[filas][ip[1][pos_proxp_i] + ( i - 1 )];


        double wVectorp_i[3] = { 0 };

        wVectorp_i[0] = uVectorp_i[1] * vVectorp_i[2] - uVectorp_i[2] * vVec-
torp_i[1];
        wVectorp_i[1] = uVectorp_i[2] * vVectorp_i[0] - uVectorp_i[0] * vVec-
torp_i[2];
        wVectorp_i[2] = uVectorp_i[0] * vVectorp_i[1] - uVectorp_i[1] * vVec-
torp_i[0];


        m4_i[3] = m4_i[7] = m4_i[11] = 0;
        m4_i[15] = 1;

        for ( int i = 0; i < 3; i++ )
        {
            m4_i[i] = uVectorp_i[i];
            m4_i[i + 4] = vVectorp_i[i];    //escala negativa para simetría respecto
a eje Y (espejo)
            m4_i[i + 8] = wVectorp_i[i];
            m4_i[i + 12] = rPoint_proxp_i[i];

        }


/****************    Cálculo ejes de rotación de la rodilla   *****************/

             //Cálculo ángulo descrito por los tobillos

             //TOBILLO IZQUIERDO

             //Producto vectorial = sen (theta tobillo izquierdo)
             temp_tob_iz [0] = uVectort_i [1] * uVectortb_i [2] - uVectort_i [2] *
uVectortb_i [1];
             temp_tob_iz [1] = uVectort_i [2] * uVectortb_i [0] - uVectort_i [0] *
uVectortb_i [2];
             temp_tob_iz [2] = uVectort_i [0] * uVectortb_i [1] - uVectort_i [1] *
uVectortb_i [0];

             //Cálculo de la norma del producto vectorial
             norma_temp_tobillo_iz = norma.Norm (temp_tob_iz);

             //Cáculo del ángulo del tobillo izquierdo
             theta_tobillo_izq = atan2 (norma_temp_tobillo_iz, (uVectortb_i [0] *
uVectort_i [0] + uVectortb_i [1] * uVectort_i [1] + uVectortb_i [2] * uVectort_i
[2]));


             //TOBILLO DERECHO

             //Producto vectorial que representa el seno del ángulo descrito por el
tobillo
             temp_tob [0] = uVectort [1] * uVectortb [2] - uVectort [2] * uVectortb
[1];
```

```
            temp_tob [1] = uVectort [2] * uVectortb [0] - uVectort [0] * uVectortb
[2];
            temp_tob [2] = uVectort [0] * uVectortb [1] - uVectort [1] * uVectortb
[0];

            //Cálculo de la norma del producto vectorial
            norma_temp_tobillo = norma.Norm (temp_tob);

            //Cáculo del ángulo del tobillo izquierdo
            theta_tobillo_dcho = atan2 (norma_temp_tobillo, (uVectortb [0] * uVec-
tort [0] + uVectortb [1] * uVectort [1] + uVectortb [2] * uVectort [2]));

            //Con estos ángulos se puede realizar el control para indicar cuál debe
ser el ángulo de la rodilla contraria

            //Evaluación de swing o stance y corrección de posición motor pierna
contraria

            //if (rTrigger (FPL1 [filas][2]) == HIGH) //pie derecho en el suelo
            //     std::cout << "pie derecho   ON" << std::endl;
            //else
            //     std::cout << "pie derecho   OFF" << std::endl;

            //if (lTrigger (FPL2 [filas][2]) == HIGH) //pie izq en el suelo
            //     std::cout << "pie izquierdo ON" << std::endl;
            //else
            //     std::cout << "pie izquierdo OFF" << std::endl;

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

            //Ademas de las placas de fuerza, se mide la posición del centro de
presión para poder utilizar el control antes de que considere qeu está pisando o no
            //Así el movimiento será menos brusco, si el centro de presión está a
una cierta distancia, implica que el movimiento se inicia para caminar

            /*delta1 = rPoint_proxp [0] + 0.345 + (FPL1 [filas][4] / FPL1 [fi-
las][2]);
            delta2 = rPoint_proxp_i[0] - 0.345 + (FPL2[filas][4] / FPL2[filas][2]);


            std::cout << std::endl << delta1 << std::endl;*/


      /*    if (delta1 >= umbral_delta && FPL1 [filas][2] >= umbral_fuerza)
                  rFSR= HIGH;
            else
                  rFSR = LOW;

            if (delta2 >= umbral_delta && FPL2 [filas][2] >= umbral_fuerza)
                  lFSR = HIGH;
            else
                  lFSR= LOW;*/


      //delta1 será la distancia en el pie derecho del tobillo al suelo y delta2 la
distancia la zona proximal de la puntera del pie

            delta1 = rPoint_proxtb [2]; //coordendada z del punto
```

```
		//delta2 = rPoint_proxp [2];

		delta1_iz = rPoint_proxtb_i [2];
		//delta2_iz = rPointp_i [2];

		if (delta1 >= 0.115 )
			rFSR = LOW;
		else
			rFSR = HIGH;

		if (delta1_iz >= 0.11)
			lFSR = LOW;
		else
			lFSR = HIGH;

	/*	if (FPL1 [filas][2] >= umbral_fuerza)
			std::cout << rFSR << lFSR << std::endl; */

		//theta_rodilla_iz = lMotorControl (-theta_tibia, lFSR, rFSR); //idem pierna contraria con una frecuencia de corte de 100Hz
		//si funciona la pierna izquierda, theta_rodilla_izquierda= ángulo que debe moverse el motor virtual izquierdo

		theta_rodilla_dcha = rMotorControl (-theta_tibia_iz, rFSR, lFSR);


//a la pierna derecha se le envia como parámetros el ángulo del tobillo izquierdo y la fuerza de pisada de su propio pie y del contrario
		//si funciona la pierna derecha, theta_rodilla_derecha= ángulo que debe moverse el motor virtual derecho

		//primera prueba: restar theta_tibia-theta_tobillo como input al control. Para comprobar si asi se ajusta mejor el movimiento


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

		//Dependiendo únicamente de las placas de fuerza
		//theta_rodilla_iz = lMotorControl (-theta_tibia, lTrigger(FPL2[filas][2]), rTrigger(FPL1[filas][2])); //idem pierna contraria con una frecuencia de corte de 100Hz
		////si funciona la pierna izquierda, theta_rodilla_izquierda= ángulo que debe moverse el motor virtual izquierdo

		//theta_rodilla_dcha = rMotorControl (-theta_tibia_iz, rTrigger(FPL1[filas][2]), lTrigger(FPL2[filas][2])); //a la pierna derecha se le envia como parámetros el ángulo del tobillo izquierdo y la fuerza de pisada de su propio pie y del contrario
		////si funciona la pierna derecha, theta_rodilla_derecha= ángulo que debe moverse el motor virtual derecho


		//Sleep (1000);

	//esta matriz convertida en vector de 16x1 es la resultante de multiplicar Ad11*A11' en Matlab
```

```
        W11[0] = m1d[0] * m1[0] + m1d[4] * m1[4] + m1d[8] * m1[8];
        W11[1] = m1d[1] * m1[0] + m1d[5] * m1[4] + m1d[9] * m1[8];
        W11[2] = m1d[2] * m1[0] + m1d[6] * m1[4] + m1d[10] * m1[8];
        W11[3] = m1d[0] * m1[1] + m1d[4] * m1[5] + m1d[8] * m1[9];
        W11[4] = m1d[1] * m1[1] + m1d[5] * m1[5] + m1d[9] * m1[9];
        W11[5] = m1d[2] * m1[1] + m1d[6] * m1[5] + m1d[10] * m1[9];
        W11[6] = m1d[0] * m1[2] + m1d[4] * m1[6] + m1d[8] * m1[10];
        W11[7] = m1d[1] * m1[2] + m1d[5] * m1[6] + m1d[9] * m1[10];
        W11[8] = m1d[2] * m1[2] + m1d[6] * m1[6] + m1d[10] * m1[10];


        W12[0] = m2d[0] * m2[0] + m2d[4] * m2[4] + m2d[8] * m2[8];
        W12[1] = m2d[1] * m2[0] + m2d[5] * m2[4] + m2d[9] * m2[8];
        W12[2] = m2d[2] * m2[0] + m2d[6] * m2[4] + m2d[10] * m2[8];
        W12[3] = m2d[0] * m2[1] + m2d[4] * m2[5] + m2d[8] * m2[9];
        W12[4] = m2d[1] * m2[1] + m2d[5] * m2[5] + m2d[9] * m2[9];
        W12[5] = m2d[2] * m2[1] + m2d[6] * m2[5] + m2d[10] * m2[9];
        W12[6] = m2d[0] * m2[2] + m2d[4] * m2[6] + m2d[8] * m2[10];
        W12[7] = m2d[1] * m2[2] + m2d[5] * m2[6] + m2d[9] * m2[10];
        W12[8] = m2d[2] * m2[2] + m2d[6] * m2[6] + m2d[10] * m2[10];


        for ( int i = 0; i < 9; i++ )
            Wantisim[i] = W12[i] - W11[i];

       //vector eje de giro resultante de la matriz antisimétrica

        wr[0] = Wantisim[5];        //es (Wantisim(3,2); Wantisim(1,3);Wan-
tisim(2,1))-> son los negativos
        wr[1] = Wantisim[6];        //Como es un vector de 16x1 las posiciones son
las de entre corchetes
        wr[2] = Wantisim[1];


        //Cálculo del vector er en coordenadas globales, sistema completo

        norma_wr = norma.Norm (wr);

        for ( int i = 0; i < 3; i++ )
        {

            er[i] = ( -wr[i] / norma_wr );  //se trata del vector wr entre su norma,
con cambio de signo

            if ( ( wr[0] * m1[4] + wr[1] * m1[5] + wr[2] * m1[6] ) < 0 )
                er[i] = -er[i];

        }

        //Cálculo de ejes locales

        for ( int i = 0; i < 3; i++ )
        {
            erL11[i] = m1[i] * er[0] + m1[i + 4] * er[1] + m1[i + 8] * er[2];  //eje
local femur
            erL12[i] = m2[i] * er[0] + m2[i + 4] * er[1] + m2[i + 8] * er[2];   //eje
local tibia
        }
```

```
      //estos datos se puede coger de los archivos erLOptimo11.dat y erLoptimo12.dat
exportados de Matlab

            std::ifstream erLoptimoFemur ("erLOptimo11.dat");

            if (erLoptimoFemur.is_open ())
            {
                  for (int i = 0; i < 9; i++)
                  {
                        erLoptimoFemur >> erLOptimo11 [i];
                  }

                  erLoptimoFemur.close ();
            }

            std::ifstream erLoptimoTibia ("erLOptimo12.dat");

            if (erLoptimoTibia.is_open ())
            {
                  for (int i = 0; i < 9; i++)
                  {
                        erLoptimoTibia >> erLOptimo12 [i];
                  }

                  erLoptimoTibia.close ();
            }



        //Cálculo de los ejes globales nuevos a partir de los óptimos

        for ( int i = 0; i < 3; i++ )
        {
            erGNuevo11[i] = m1[i] * erLOptimo11[0] + m1[i + 4] * erLOptimo11[1] +
m1[i + 8] * erLOptimo11[2];
            erGNuevo12[i] = m2[i] * erLOptimo12[0] + m2[i + 4] * erLOptimo12[1] +
m2[i + 8] * erLOptimo12[2];
        }

        //se calculan las proyecciones de los vectores u de la tibia y del fémur para
calcular el ángulo que describe la rodilla durante el movimiento
        //Estos vectores serán normalizados para que sean unitarios. Se utilizan es-
tos porque el ángulo que describen entre ellos durante el movimento debe conservarse
        //entre el sólido que representa el fémur y el que representa la tibia


        temporal1 = m1[0] * erGNuevo11[0] + m1[1] * erGNuevo11[1] + m1[2] * erG-
Nuevo11[2];
        temporal2 = m2[0] * erGNuevo11[0] + m2[1] * erGNuevo11[1] + m2[2] * erG-
Nuevo11[2];

        for ( int j = 0; j < 3; j++ )
        {
            u11_p[j] = m1[j] - temporal1*erGNuevo11[j];
            u12_p[j] = m2[j] - temporal2*erGNuevo11[j];
        }
```

```
        norma_u11p = norma.Norm ( u11_p );
        norma_u12p = norma.Norm ( u12_p );

      //Normalizamos los vectores u y sobrescribimos
        for ( int i = 0; i < 3; i++ )
        {
            u11_pn[i] = u11_p[i] / norma_u11p;
            u12_pn[i] = u12_p[i] / norma_u12p;
        }


        //A partir de estos vectores se puede calcular el ángulo descrito por la ro-
dilla

        //Producto vectorial
        temp[0] = u11_pn[1] * u12_pn[2] - u11_pn[2] * u12_pn[1];
        temp[1] = u11_pn[2] * u12_pn[0] - u11_pn[0] * u12_pn[2];
        temp[2] = u11_pn[0] * u12_pn[1] - u11_pn[1] * u12_pn[0];

        //Norma del producto vectorial
        norma_temp = norma.Norm ( temp );


        theta_rodilla = (atan2 ( norma_temp , ( u11_pn[0] * u12_pn[0] + u11_pn[1] *
u12_pn[1] + u11_pn[2] * u12_pn[2] ) ))

       angulos_tobillo_rodilla << std::setw (10) << std::setprecision (8) << std::fi-
xed << theta_tibia_iz << "\t" << theta_rodilla <<"\t"<<theta_rodi-
lla_dcha<<"\t"<<theta_pie_iz<< endl;


        //calcular matriz antisimétrica con el vector erLOptimo11

        identity[0] = identity[4] = identity[8] =  1 ;
        identity[1] = identity[2] = identity[3] = identity[5] = identity[6] = iden-
tity[7] =  0 ;

        antisimetrica_rodilla[0] = antisimetrica_rodilla[4] = antisimetrica_rodi-
lla[8] =  0 ;
        antisimetrica_rodilla[1] = -erLOptimo11[2];
        antisimetrica_rodilla[2] = erLOptimo11[1];
        antisimetrica_rodilla[3] = erLOptimo11[2];
        antisimetrica_rodilla[5] = -erLOptimo11[0];
        antisimetrica_rodilla[6] = -erLOptimo11[1];
        antisimetrica_rodilla[7] = erLOptimo11[0];

        //Rodrigues Formula

        for ( int i = 0; i < 3; i++ )
            square_antisim[i] = antisimetrica_rodilla[i] * antisimetrica_rodilla[0] +
antisimetrica_rodilla[i + 3] * antisimetrica_rodilla[1] + antisimetrica_rodilla[i +
6] * antisimetrica_rodilla[2];
        for ( int i = 3; i < 6; i++ )
            square_antisim[i] = antisimetrica_rodilla[i - 3] * antisimetrica_rodi-
lla[3] + antisimetrica_rodilla[i] * antisimetrica_rodilla[4] + antisimetrica_rodi-
lla[i + 3] * antisimetrica_rodilla[5];
        for ( int i = 6; i < 9; i++ )
```

```
square_antisim[i] = antisimetrica_rodilla[i - 6] * antisimetrica_rodilla[6] + antisi-
metrica_rodilla[i - 3] * antisimetrica_rodilla[7] + antisimetrica_rodilla[i] * anti-
simetrica_rodilla[8];

	for ( int i = 0; i < 9; i++ ) //calculado para pierna derecha
	{
		mat_trans_relativa_rodilla[i] = identity[i] + antisimetrica_ro-
dilla[i] * sin ( theta_rodilla_dcha ) + 2 * (square_antisim[i])*pow ( sin ( theta_ro-
dilla_dcha / 2) , 2 );
	}


	//esta matriz ha sido calculada en Matlab como un sistema de encuaciones (10
ecuaciones y 9 incógnitas)


		std::ifstream matrizA0 ("matrizA0.dat");

		if (matrizA0.is_open ())
		{
			for (int i = 0; i < 9; i++)
			{
				if (i== 0 || i == 4 || i==8)
					matrizA0 >> A0 [i];
				else
				{
					if (i==1||i==5)
						matrizA0 >> A0 [i + 2];
					if (i == 2)
						matrizA0 >> A0 [i + 4];
					if (i == 6)
						matrizA0 >> A0 [i - 4];
					if (i == 3 || i == 7)
						matrizA0 >> A0 [i - 2];
				}

			}

			matrizA0.close ();
		}


	//Calculo de la matriz de la tibia vista desde el fémur, proyectada sobre el
eje de giro de la rodilla A2= A11*Atheta*A0

		for (int i = 0; i < 3; i++)
			Atibia [i] = m1 [i] * mat_trans_relativa_rodilla [0] + m1 [i +
4] * mat_trans_relativa_rodilla [1] + m1 [i + 8] * mat_trans_relativa_rodilla [2];
		for (int j = 3; j < 6; j++)
			Atibia [j] = m1 [j - 3] * mat_trans_relativa_rodilla [3] + m1 [j
+ 1] * mat_trans_relativa_rodilla [4] + m1 [j + 5] * mat_trans_relativa_rodilla [5];
		for (int k = 6; k < 9; k++)
			Atibia [k] = m1 [k - 6] * mat_trans_relativa_rodilla [6] + m1 [k
- 2] * mat_trans_relativa_rodilla [7] + m1 [k + 2] * mat_trans_relativa_rodilla [8];
```

```
        ////transformación a 16x1

        Atibia_d[0] = ( Atibia[0] * A0[0] + Atibia[3] * A0[1] + Atibia[6] * A0[2] );
        Atibia_d[1] = ( Atibia[1] * A0[0] + Atibia[4] * A0[1] + Atibia[7] * A0[2] );
        Atibia_d[2] = ( Atibia[2] * A0[0] + Atibia[5] * A0[1] + Atibia[8] * A0[2] );

        Atibia_d[3] = Atibia_d[7] = Atibia_d[11] = 0;
        Atibia_d[15] = 1;

        Atibia_d[4] = ( Atibia[0] * A0[3] + Atibia[3] * A0[4] + Atibia[6] * A0[5] );
        Atibia_d[5] = ( Atibia[1] * A0[3] + Atibia[4] * A0[4] + Atibia[7] * A0[5] );
        Atibia_d[6] = ( Atibia[2] * A0[3] + Atibia[5] * A0[4] + Atibia[8] * A0[5] );

        Atibia_d[8] = ( Atibia[0] * A0[6] + Atibia[3] * A0[7] + Atibia[6] * A0[8] );
        Atibia_d[9] = ( Atibia[1] * A0[6] + Atibia[4] * A0[7] + Atibia[7] * A0[8] );
        Atibia_d[10] = ( Atibia[2] * A0[6] + Atibia[5] * A0[7] + Atibia[8] * A0[8] );

        Atibia_d[12] = rPoint_proxt[0];
        Atibia_d[13] = rPoint_proxt[1];
        Atibia_d[14] = rPoint_proxt[2];

        //Cálculo posición nuevo tobillo

        Atobillo[3] = Atobillo[7] = Atobillo[11] = 0;
        Atobillo[15] = 1;

        for ( int i = 0; i < 3; i++ )
        {
            Atobillo[i] = uVectortb[i];
            Atobillo[i + 4] = vVectortb[i];
            Atobillo[i + 8] = wVectortb[i];
            Atobillo[i + 12] = rPoint_proxt[i] + Atibia_d[i + 8] * ( -norma_distal_t
);

        }

        //Aplicación de escala

        //Pelvis
        for ( int i = 0; i < 3; i++ )
        {
            m0[i] = m0[i] * 1.0845; //factores de escala según archivos matscale y
ppoint
            m0[i + 4] = m0[i+4] * 1.1683;
            m0[i + 8] = m0[i+8] * 1.0008;
            m0[i + 12] = m0[i+12];
        }

        //Femur dcho
        for ( int i = 0; i < 3; i++ )
        {
            m1[i] = m1[i] * 1.0660; //factores de escala según archivos matscale y
ppoint
            m1[i + 4] =m1[i+4] * 1.0660;
            m1[i + 8] = m1[i+8] * 1.0660;
            m1[i + 12] = m1[i+12];
        }
```

```
        //Femur izq
        for ( int i = 0; i < 3; i++ )
        {
            m1_i [i] = m1_i [i] * 1.0664; //factores de escala según archivos
matscale y ppoint
            m1_i[i + 4] = m1_i[i+4] * ( -1.0664 ); //escala negativa para hacer sime-
tría respecto al eje Y
            m1_i[i + 8] = m1_i[i+8] * 1.0664;
            m1_i[i + 12] = m1_i[i+12];
        }

        //Tibia dcha

        for ( int i = 0; i < 3; i++ )
        {
            m2[i] = m2[i] * 1.1635;
            m2[i + 4] = m2[i+4] * 1.1635;
            m2[i + 8] = m2[i+8] * 1.1635;
            m2[i + 12] = m2[i+12];
        }

        //Tibia izq
        for ( int i = 0; i < 3; i++ )
        {
            m2_i[i] = m2_i[i] * 1.1926;
            m2_i[i + 4] = m2_i[i+4] * ( -1.1926 );
            m2_i[i + 8] = m2_i[i+8] * 1.1926;
            m2_i[i + 12] = m2_i[i+12];
        }

        //Tobillo dcho
        for ( int i = 0; i < 3; i++ )
        {
            m3[i] = m3[i] * 1.1845;
            m3[i + 4] = m3[i+4] * 1.3240;
            m3[i + 8] = m3[i+8] * 1.4634;
            m3[i + 12] = m3[i+12];
        }

        //Tobillo izq
        for ( int i = 0; i < 3; i++ )
        {
            m3_i[i] = m3_i[i] * 1.3238;
            m3_i[i + 4] = m3_i[i+4] * ( -1.2949 ); //escala negativa para simetría
respecto eje Y
            m3_i[i + 8] = m3_i[i+8] * 1.2659;
            m3_i[i + 12] = m3_i[i+12];
        }

        //Punta pie dcho
        for ( int i = 0; i < 3; i++ )
        {
            m4[i] = m4[i] * 1.1845;
            m4[i + 4] = m4[i+4] * 1.3240;
            m4[i + 8] = m4[i+4] * 1.4634;
            m4[i + 12] = m4[i+12];
        }
```

```
        //Punta pie izq
        for ( int i = 0; i < 3; i++ )
        {
            m4_i[i] = m4_i[i] * 1.3238;
            m4_i[i + 4] = m4_i[i+4] * ( -1.2949 );    //escala negativa para simetría
respecto a eje Y (espejo)
            m4_i[i + 8] = m4_i[i+8] * 1.2659;
            m4_i[i + 12] = m4_i[i+12];
        }

        //Tibia secundaria. Coordenadas en base al fémur

        for ( int i = 0; i < 3; i++ )
        {
            Atibia_d[i] = Atibia_d[i] * 1.1635;
            Atibia_d[i + 4] = Atibia_d[i + 4] * 1.1635;
            Atibia_d[i + 8] = Atibia_d[i + 8] * 1.1635;
            Atibia_d[i + 12] = Atibia_d[i + 12];
        }


        //tobillo secundario para nueva tibia
        for ( int i = 0; i < 3; i++ )
        {
            Atobillo[i] = Atobillo[i] * 1.1845;
            Atobillo[i + 4] = Atobillo[i + 4] * 1.3240;
            Atobillo[i + 8] = Atobillo[i + 8] * 1.4634;
            Atobillo[i + 12] = Atobillo[i + 12];

        }

        //Actualización de matrices

        osg::Matrixd mpel ( m0 );
        mat_trans0->setMatrix ( mpel );


        osg::Matrixd mf ( m1 );
        mat_trans1->setMatrix ( mf );


        osg::Matrixd mt ( m2 );
        mat_trans2->setMatrix ( mt );


        osg::Matrixd mt2 ( Atibia_d );
        mat_trans2sec->setMatrix ( mt2 );


        osg::Matrixd mtb ( m3 );
        mat_trans3->setMatrix ( mtb );

        osg::Matrixd mtb2 ( Atobillo );
        mat_trans3sec->setMatrix ( mtb2 );


        osg::Matrixd mp ( m4 );
        mat_trans4->setMatrix ( mp );
```

```
        osg::Matrixd mf_i ( m1_i );
        mat_trans1_i->setMatrix ( mf_i );


        osg::Matrixd mt_i ( m2_i );
        mat_trans2_i->setMatrix ( mt_i );


        osg::Matrixd mtb_i ( m3_i );
        mat_trans3_i->setMatrix ( mtb_i );


        osg::Matrixd mp_i ( m4_i );
        mat_trans4_i->setMatrix ( mp_i );


        //Visualización

        viewer.frame ( );
        Sleep ( 30 );

        osgViewer::RecordCameraPathHandler ( );

 }  //bucle for para matriz qProj

 angulo_tibia.close ();
 angulos_tobillo_rodilla.close ();

}   //main
```

11.4. Código experimentación de sensores

DAQmx

DAQmx.h

```
#ifndef DAQMX_H
#define DAQMX_H

#include <string>
#include <vector>
#include <math.h>
#include "NIDAQmx.h"



#ifdef __MINGW32__
	#define strcpy_s strcpy
#endif

#ifdef DAQmx_EXPORTS
	#define DAQmxExport __declspec(dllexport)
#else
	#define DAQmxExport __declspec(dllimport)
#endif
```

```
#pragma warning(disable:4251 4351)


// Type: pointer to sensor scale function
typedef float64 (*scaleFun_t) (const float64);



// Abstract base class for DAQmx tasks
class DAQmxExport DAQmx
{
protected:
	static const double pi;

	double fltPar;
	char taskName[2048];
	TaskHandle taskHandle;
	std::vector<double> buffer;
	std::vector<std::vector<double> > data;
	int32 nCh, nFrames, sampleRate, currentFrame, read;

	DAQmx(const char *name, int length = 1000, int frameRate = 1000, float cutoff
= 0);
	~DAQmx();

	void errChk(int32 error);

public:
	void start();
	int32 peek(double *out);
	void write(std::string file);
	void reset();
	void clear();

private:
	virtual void getCurrentData() = 0;
};

// Analog input class
class DAQmxExport Analog: public DAQmx
{
public:
	Analog(const char *name, int length = 1000, int frameRate = 1000, float cutoff
= 0) :
	  DAQmx(name, length, frameRate, cutoff) {};

	void addChannel(const char *chn, scaleFun_t sf = nullptr, int32 config =
DAQmx_Val_RSE);
	void triggerOut(const char *terminal);

private:
	std::vector<scaleFun_t> scaleFun;

	void getCurrentData(); //deja de ser abstracta al declarar la funcion virtual
proveniente de la clase base (que si es abstracta)
};
```

```
// Rotary encoder class
class DAQmxExport Encoder: public DAQmx
{
public:
	Encoder(const char *name, int length = 1000, int frameRate = 1000, float cut-
off = 0) :
	  DAQmx(name, length, frameRate, cutoff), zero(0) {};

	void addChannel(const char *ctr, const int ppr, const bool rev = false,
		const char *A = "", const char *Z = "", const char *B = "");

private:
	double zero;
	bool reverse;

	void getCurrentData();
};


#endif // DAQMX_H
```

DAQmx.cpp

```
#include <string.h>
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <fstream>
#include "DAQmx.h"

// Initialize the static constant pi
const double DAQmx::pi = 4*atan(1.0);

// DAQmx constructor
DAQmx::DAQmx(const char *name, const int length, const int frameRate, const float
cutoff) :
nCh(0), fltPar(1.0), currentFrame(0), taskHandle(0)
{
	nFrames = length;
	sampleRate = frameRate;

	strcpy_s(taskName, name);

	errChk(DAQmxCreateTask(taskName, &taskHandle));

	if (cutoff > 0)
		fltPar = 1.0 / (sampleRate/(2.0*pi*cutoff) + 1.0);
}

// DAQmx destructor
DAQmx::~DAQmx()
{
	clear();
}

// DAQmx error code parser
void DAQmx::errChk(int32 error)
```

```
{
	char errBuff[2048];
	if (DAQmxFailed(error))
    {
		DAQmxGetExtendedErrorInfo(errBuff, 2048);
		std::cout << "DAQmx Error: " << errBuff << std::endl;
		throw "DAQmx";
	}
}

// DAQmx start capture trigger
void DAQmx::start()
{
	errChk(DAQmxStartTask(taskHandle));
}

// DAQmx data storage and output
int32 DAQmx::peek(double *out)
{
	// Get available samples into temporary buffer
	this->getCurrentData();

	// Check if acquired samples fit into remaining storage space
	read = std::min(read, nFrames - currentFrame);

	// Write buffer data into storage array
	for (int ch = 0; ch < nCh; ch++)
    {
		for (int f = 0; f < read; f++)
		 {
			// Get data from DAQ buffer and scale to output units
			data[ch][currentFrame + f] = buffer[nCh*f + ch];

			// Apply first-order low-pass filter
			if (fltPar < 1 && currentFrame + f > 0)
            {
				data[ch][currentFrame + f] *= fltPar;
				data[ch][currentFrame + f] += (1-fltPar)*data[ch][cur-
rentFrame + f - 1];
			}
		}
		if (read)
			out[ch] = static_cast<double>(data[ch][currentFrame + read -
1]);
	}
	currentFrame += read;

	return currentFrame;
}

// Dump sensor(s) data to file
void DAQmx::write(const std::string file)
{
	std::ofstream outFile(file.c_str());
	if (outFile.is_open())
    {
		outFile << std::setprecision(8) << std::fixed;
		for (int i = 0; i < currentFrame; i++)
```

```
		{
			for (int ch = 0; ch < nCh; ch++)
		    {
				outFile << std::setw(16) << data[ch][i];
			}
			outFile << std::endl;
		}
		outFile.close();
	}
}

// Analog channel configuration
void Analog::addChannel(const char *chn, scaleFun_t sf, int32 config)
{
	nCh++;
	data.resize(nCh);
	data[nCh-1].resize(nFrames);
	buffer.resize(nCh*sampleRate);
	scaleFun.resize(nCh, 0);
	scaleFun[nCh-1] = sf;

	errChk(DAQmxCreateAIVoltageChan(taskHandle, chn, "", config, 0.0, 5.0,
DAQmx_Val_Volts, NULL));
	errChk(DAQmxCfgSampClkTiming(taskHandle, "OnboardClock", sampleRate,
DAQmx_Val_Rising, DAQmx_Val_ContSamps, sampleRate));
}

// Send analog start trigger through terminal
void Analog::triggerOut(const char *terminal)
{
	errChk(DAQmxExportSignal(taskHandle, DAQmx_Val_StartTrigger, terminal));
}

// Analog buffer read
void Analog::getCurrentData()
{
	errChk(DAQmxReadAnalogF64(taskHandle, DAQmx_Val_Auto, 5.0, DAQmx_Val_GroupBy-
ScanNumber, &buffer[0], 2*sampleRate, &read, NULL));

	// Transform voltages into output units
	for (int ch = 0; ch < nCh; ch++)
    {
		if (scaleFun[ch] != 0)
		{
			for (int f = 0; f < read; f++)
				buffer[nCh*f + ch] = scaleFun[ch](buffer[nCh*f + ch]);
		}
	}
};

// Encoder channel configuration
void Encoder::addChannel(const char *ctr, int ppr, bool rev, const char *A, const
char *Z, const char *B)
{
	if (nCh > 0)
    {
		std::cout << taskName << ": Encoder tasks can only have one channel" <<
std::endl;
```

```
			throw "DAQmx";
		}
		nCh = 1;
		zero = 0.0;
		reverse = rev;

		data.resize(1);
		data[0].resize(nFrames);
		buffer.resize(sampleRate);

		errChk(DAQmxCreateCIAngEncoderChan(taskHandle, ctr, "", DAQmx_Val_X4, 0, 0.0,
DAQmx_Val_AHighBHigh, DAQmx_Val_Degrees, ppr, 0.0, ""));
		errChk(DAQmxCfgSampClkTiming(taskHandle, "/Orthosis/ai/SampleClock", sam-
pleRate, DAQmx_Val_Rising, DAQmx_Val_ContSamps, sampleRate));

		if (*A != 0) errChk(DAQmxSetCIEncoderAInputTerm(taskHandle, "", A));
		if (*Z != 0) errChk(DAQmxSetCIEncoderZInputTerm(taskHandle, "", Z));
		if (*B != 0) errChk(DAQmxSetCIEncoderBInputTerm(taskHandle, "", B));
}

// Encoder buffer read
void Encoder::getCurrentData()
{
		errChk(DAQmxReadCounterF64(taskHandle, DAQmx_Val_Auto, 5.0, &buffer[0], sam-
pleRate, &read, NULL));

		// Invert if reverse is true
		for (int i = 0; i < read; i++)
	{
			if (reverse)
				buffer[i] = -buffer[i];
		}
};

// Stop task and reset frame counter
void DAQmx::reset()
{
		if (taskHandle)
	{
			errChk(DAQmxStopTask(taskHandle));
			currentFrame = 0;
		}
};
// Clear task
void DAQmx::clear()
{
		reset();
		if (taskHandle)
	{
			errChk(DAQmxGetTaskName(taskHandle, taskName, 2048));
			std::cout << "DAQmx task \"" << taskName << "\" cleared" << std::endl;
			errChk(DAQmxClearTask(taskHandle));
			taskHandle = 0;
		}
};
```

Header Files

CalculateMatrix.h

```
#ifndef CALCULATEMATRIX_H
#define CALCULATEMATRIX_H

#include <math.h>
#include <iostream>

#define pi_rot 4*atan(1)

using namespace std;

class CalculateMatrix
{

public:

    float CalculateRotation ( float qw , float qx , float qy , float qz )

    {
       // float m[3][3]; //matriz de 3x3 en la que se convertirá el quaternion

        float m[3][3] ;
        float angulo_pitch;
        float angulo_roll;

        m[0][0] = qw*qw + qx*qx - qy*qy - qz*qz;
        m[0][1] = 2 * ( qw*qz + qx*qy );
        m[0][2] = 2 * ( qx*qz - qw*qy );
        m[1][0] = 2 * ( qx*qy - qw*qz );
        m[1][1] = qw*qw + qy*qy - qx*qx - qz*qz;
        m[1][2] = 2 * ( qw*qx + qy*qz );
        m[2][0] = 2 * ( qw*qy + qx*qz );
        m[2][1] = 2 * ( qy*qz - qw*qx );
        m[2][2] = qw*qw + qz*qz - qx*qx - qy*qy;


        angulo_pitch = ( asinf ( -2.0f * ( qx*qz - qy*qw ) ) )*180.0 / pi_rot; //ma-
triz de rotación, sen del ángulo roll será la componente z del vector Y

        angulo_roll = atan2f ( 2.0f * ( qy*qz + qx*qw ) , -qx*qx - qy*qy + qz*qz +
qw*qw );

        return angulo_pitch;
    }

};


#endif //CALCULATEMATRIX_H
```

Calibration.h

```
#ifndef CALIBRATION_H
#define CALIBRATION_H

// Accelerometer calibration: Ka*a(LSB) + a0 = a(g)
const float Ka[3][3] = {{ 3.809070e-03,-1.620002e-05,-4.287743e-05},
                        { 6.330071e-05, 3.713880e-03, 3.380687e-05},
                        { 6.018032e-05,-5.007405e-05, 3.832537e-03}};

const float a0[3] = {-1.806397e-02, 3.796210e-03, 4.385656e-02};

// Magnetometer calibration: Km*m(LSB) + m0 = m(Gauss)
const float Km[3][3] = {{ 8.711491e-04, 9.425670e-07, 4.027776e-06},
                        { 9.425670e-07, 8.736228e-04, 2.061346e-05},
                        { 4.027776e-06, 2.061346e-05, 9.072996e-04}};

const float m0[3] = { 9.228341e-02, 1.150613e-02, 3.169481e-02};

// Gyroscope calibration: Kg*g(LSB) + g0 = g(rad/s)
const float Kg[3][3] = {{ 1.205197e-03, 5.273700e-07,-5.388768e-06},
                        { 4.927599e-06, 1.206750e-03,-6.411275e-06},
                        {-2.484503e-05, 1.048186e-05, 1.213966e-03}};

const float g0[3] = { 5.932811e-02,-2.360141e-02, 1.242429e-02};

#endif //CALIBRATION_H
```

Hrtimer.h

```
#ifndef HRTIMER_H
#define HRTIMER_H

#include "Windows.h"

// High resolution timer class
class hrtimer
{
private:
	LARGE_INTEGER t0;   // Reference time in clock ticks
	LARGE_INTEGER freq; // Timer frequency in ticks per second

public:
	hrtimer()
	{
		QueryPerformanceFrequency(&freq);
		reset();
	}

	void reset()
	{
		QueryPerformanceCounter(&t0);
	}

	double t()
	{
		LARGE_INTEGER t;

		QueryPerformanceCounter(&t);
```

```
                return static_cast<double>(t.QuadPart-t0.QuadPart)/freq.QuadPart;

        }
};


#endif // HRTIMER_H
```

SensorScale.h

```
#ifndef SENSORSCALE_H
#define SENSORSCALE_H

#include "DAQmx.h"

enum rMode_t {pullUp, pullDown};

// FSR 402 resistive sensor (R: Resistor value in Ohm, mode: Resistor configuration)
inline float64 FSR402(const float64 Vin, const float64 R = 1e4, const rMode_t mode =
pullUp)
{
        float64 V = Vin;

        if (V > 4.800) V = 4.800;
        if (V < 0.075) V = 0.075;

        if (mode == pullUp)
                return pow(R/6360.5 * V/(5.0-V), -1.4126);
        else
                return pow(R/6360.5 * (5.0/V-1.0), -1.4126);
}


#endif // SENSORSCALE_H
```

Serial.h

```
#ifndef __SERIAL_H__
#define __SERIAL_H__

#define FC_DTRDSR       0x01
#define FC_RTSCTS       0x02
#define FC_XONXOFF      0x04
#define ASCII_BEL       0x07
#define ASCII_BS        0x08
#define ASCII_LF        0x0A
#define ASCII_CR        0x0D
#define ASCII_XON       0x11
#define ASCII_XOFF      0x13

class CSerial
{

public:
        CSerial();
        ~CSerial();

        BOOL Open( int nPort = 2, int nBaud = 9600 );
```

```
	BOOL Close( void );

	int ReadData( void *, int );
	int SendData( const char *, int );
	int ReadDataWaiting( void );

	BOOL IsOpened( void ){ return( m_bOpened ); }

protected:
	BOOL WriteCommByte( unsigned char );

	HANDLE m_hIDComDev;
	OVERLAPPED m_OverlappedRead, m_OverlappedWrite;
	BOOL m_bOpened;

};

#endif
```

Sockets.h

```
#include<winsock2.h>
#include<stdio.h>

#pragma comment(lib, "ws2_32.lib")

class UDPSocket
{
	SOCKET s;
	WSADATA wsa;
	struct sockaddr_in server;
	int slen, recv_len;
	int BUFLEN, err;
	char *buf;

public:

	UDPSocket(const int PORT=8888, const int buflen=512)
	{
		BUFLEN = buflen;
		slen = sizeof(server);
		buf = new char[BUFLEN];

		//Initialise winsock
		printf("Initialising Winsock...");
		if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
			printf("Failed. Error Code: %d", WSAGetLastError());
			exit(EXIT_FAILURE);
		}
		printf(" done\n");

		//Create a socket
		if ((s = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET) {
			printf("Could not create socket: %d", WSAGetLastError());
			exit(EXIT_FAILURE);
		}
		printf("Socket created\n");

		//Set non-blocking mode
```

```
		u_long iMode = 1;
		ioctlsocket(s, FIONBIO, &iMode);

		//Prepare the sockaddr_in structure
		server.sin_family = AF_INET;
		server.sin_addr.s_addr = INADDR_ANY;
		server.sin_port = htons(PORT);

		//Bind
		if (bind(s, (struct sockaddr *)&server, sizeof(server)) == SOCKET_ER-
ROR) {
			printf("Bind failed with error code: %d", WSAGetLastError());
			exit(EXIT_FAILURE);
		}
		printf("Bind done\n");
	}

	bool receive(char *message, struct sockaddr_in *address)
	{
		memset(message, '\0', BUFLEN);
		if ((recv_len = recvfrom(s, message, BUFLEN, 0, (struct sockaddr *)ad-
dress, &slen)) == SOCKET_ERROR) {
			if ((err = WSAGetLastError()) != 10035 && WSAGetLastError() !=
10054) {
				printf("recvfrom() failed with error code: %d\n", err);
				exit(EXIT_FAILURE);
			}
			return false;
		} else {
			if (!strncmp(message, "S", 1)) {
				printf("Received \"%s\" from %s\n", message, inet_ntoa(ad-
dress->sin_addr));
			}
			return true;
		}
	}

	void send(const char *out, int len, struct sockaddr_in address)
	{
		if (sendto(s, out, len, 0, (struct sockaddr*) &address, slen) ==
SOCKET_ERROR) {
			printf("sendto() failed with error code: %d\n", WSAGetLastEr-
ror());
			exit(EXIT_FAILURE);
		}
	}

	~UDPSocket()
	{
		closesocket(s);
		WSACleanup();
	}

};
```

Source Files

Serial.cpp

```
#include <Windows.h>
#include "Serial.h"

CSerial::CSerial()
{
	memset(&m_OverlappedRead, 0, sizeof(OVERLAPPED));
	memset(&m_OverlappedWrite, 0, sizeof(OVERLAPPED));
	m_hIDComDev = NULL;
	m_bOpened = FALSE;
}

CSerial::~CSerial()
{
	Close();
}

BOOL CSerial::Open(int nPort, int nBaud)
{
	if (m_bOpened)
		return(TRUE);

	char szPort[15] = { 0 };
	char szComParams[50] = { 0 };
	DCB dcb;

	wsprintf(szPort, "\\\\.\\COM%d", nPort);
	m_hIDComDev = CreateFile(szPort, GENERIC_READ | GENERIC_WRITE, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, NULL);
	if (m_hIDComDev == NULL)
		return(FALSE);

	memset(&m_OverlappedRead, 0, sizeof(OVERLAPPED));
	memset(&m_OverlappedWrite, 0, sizeof(OVERLAPPED));

	COMMTIMEOUTS CommTimeOuts;
	CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;
	CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
	CommTimeOuts.ReadTotalTimeoutConstant = 0;
	CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
	CommTimeOuts.WriteTotalTimeoutConstant = 5000;
	SetCommTimeouts(m_hIDComDev, &CommTimeOuts);

	wsprintf(szComParams, "COM%d:%d,n,8,1", nPort, nBaud);

	m_OverlappedRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
	m_OverlappedWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

	dcb.DCBlength = sizeof(DCB);
	GetCommState(m_hIDComDev, &dcb);
	dcb.BaudRate = nBaud;
	dcb.ByteSize = 8;
	unsigned char ucSet;
	ucSet = (unsigned char)((FC_RTSCTS & FC_DTRDSR) != 0);
	ucSet = (unsigned char)((FC_RTSCTS & FC_RTSCTS) != 0);
```

```
	ucSet = (unsigned char)((FC_RTSCTS & FC_XONXOFF) != 0);
	if (!SetCommState(m_hIDComDev, &dcb) || !SetupComm(m_hIDComDev, 10000, 10000)
|| m_OverlappedRead.hEvent == NULL || m_OverlappedWrite.hEvent == NULL)
	{
		DWORD dwError = GetLastError();
		if (m_OverlappedRead.hEvent != NULL) CloseHandle(m_OverlappedRead.hE-
vent);
		if (m_OverlappedWrite.hEvent != NULL) CloseHandle(m_OverlappedWrite.hE-
vent);
		CloseHandle(m_hIDComDev);
		return(FALSE);
	}

	m_bOpened = TRUE;

	return(m_bOpened);
}

BOOL CSerial::Close(void)
{
	if (!m_bOpened || m_hIDComDev == NULL)
		return(TRUE);
	if (m_OverlappedRead.hEvent != NULL)
		CloseHandle(m_OverlappedRead.hEvent);
	if (m_OverlappedWrite.hEvent != NULL)
		CloseHandle(m_OverlappedWrite.hEvent);
	CloseHandle(m_hIDComDev);
	m_bOpened = FALSE;
	m_hIDComDev = NULL;

	return(TRUE);
}

BOOL CSerial::WriteCommByte(unsigned char ucByte)
{
	BOOL bWriteStat;
	DWORD dwBytesWritten;

	bWriteStat = WriteFile(m_hIDComDev, (LPSTR)&ucByte, 1, &dwBytesWritten,
&m_OverlappedWrite);
	if (!bWriteStat && (GetLastError() == ERROR_IO_PENDING))
	{
		if (WaitForSingleObject(m_OverlappedWrite.hEvent, 1000))
			dwBytesWritten = 0;
		else
		{
			GetOverlappedResult(m_hIDComDev, &m_OverlappedWrite,
&dwBytesWritten, FALSE);
			m_OverlappedWrite.Offset += dwBytesWritten;
		}
	}

	return(TRUE);
}

int CSerial::SendData(const char *buffer, int size)
{
	if (!m_bOpened || m_hIDComDev == NULL) return(0);
```

```
	DWORD dwBytesWritten = 0;
	int i;
	for (i = 0; i < size; i++)
	{
		WriteCommByte(buffer[i]);
		dwBytesWritten++;
	}

	return((int)dwBytesWritten);
}

int CSerial::ReadDataWaiting(void)
{
	if (!m_bOpened || m_hIDComDev == NULL) return(0);

	DWORD dwErrorFlags;
	COMSTAT ComStat;

	ClearCommError(m_hIDComDev, &dwErrorFlags, &ComStat);

	return((int)ComStat.cbInQue);
}

int CSerial::ReadData(void *buffer, int limit)
{
	if (!m_bOpened || m_hIDComDev == NULL) return(0);

	BOOL bReadStatus;
	DWORD dwBytesRead, dwErrorFlags;
	COMSTAT ComStat;

	ClearCommError(m_hIDComDev, &dwErrorFlags, &ComStat);
	if (!ComStat.cbInQue) return(0);

	dwBytesRead = (DWORD)ComStat.cbInQue;
	if (limit < (int)dwBytesRead) dwBytesRead = (DWORD)limit;

	bReadStatus = ReadFile(m_hIDComDev, buffer, dwBytesRead, &dwBytesRead,
&m_OverlappedRead);
	if (!bReadStatus)
	{
		if (GetLastError() == ERROR_IO_PENDING)
		{
			WaitForSingleObject(m_OverlappedRead.hEvent, 2000);
			return((int)dwBytesRead);
		}
		return(0);
	}

	return((int)dwBytesRead);
}
```

Bucle principal

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                  Código Lectura de Encoder e IMU                       //
//                                                                        //
////////////////////////////////////////////////////////////////////////////


#include <Windows.h>
#include <conio.h>
#include <iostream>
#include <ios>
#include <iomanip>
#include <fstream>
#include <vector>
#include <chrono>
#include <thread>
#include <iomanip>


#include "Serial.h"
#include "hrtimer.h"
#include "CalculateMatrix.h"
#include "DAQmx.h"
#include "NIDAQmx.h"



#define NOUT 60000
//#define PORT 11
#define PORT 3
//#define NFLOATS 9 //para cálculos  en binario
#define NFLOATS 4 //para cálculos en cuaterniones
#define BUFSIZE (4*NFLOATS+2)



//Definición de objetos de las clases asignadas

CSerial s;
//CalculateMatrix matrix_rot;


/******Definición de objetos de las clases asignadas y construcción ******/


Analog clk ( "Clock" , NOUT , 100 );
Encoder encoder ( "Encoder" , NOUT , 100 , 10 ); // Inicialización del constructor,
6000frames, 100Hz de frecuencia y 10 Hz para el filtro


using namespace std;
using namespace std::chrono;
using std::ofstream;
```

```
#pragma pack(1)
static union
{
    char buffer[BUFSIZE];
    struct
    {
        char header;
        float q[NFLOATS];
        char checksum;
    };
};

std::vector<std::vector<float>> data ( NFLOATS );


void circshift ( size_t size )
{
    for ( int i = 0; i < size - 1; i++ )
        buffer[i] = buffer[i + 1];

    s.ReadData ( &buffer[size - 1] , 1 );
}

void saveFile ( const std::string file )
{
    std::ofstream outFile ( file.c_str ( ) );
    if ( outFile.is_open ( ) )
    {
        outFile << std::setprecision ( 7 ) << std::fixed;
        for ( int i = 0; i < data[0].size ( ); i++ )
        {
            for ( int ch = 0; ch < NFLOATS; ch++ )
            {
                outFile << std::setw ( 15 ) << data[ch][i];
            }
            outFile << std::endl;
        }
        outFile.close ( );
    }
}


int main ( int argc , char* argv[] )

{

   /*******************Output values encoders & IMU **********************/


    double state_encoder;   //variable a la que apunta la función peek para obtener
el valor actual del encoder
    double state_IMU;       //variable a la que apunta la función peek para obtener
el valor actual de la IMU
    double temp[NOUT] = { 0 };       //variable temporal para almacenar en un array
los valores de salida del encoder
    double encj = 0.0;

    // float angulo_pitch = 0.0;
```

```
    /************** Creación canal analógico y encoder. Inicialización ************/

       encoder.addChannel ("Orthosis/ctr0", 4000, true, "/Orthosis/PFI0", "/Ortho-
sis/PFI1", "/Orthosis/PFI2"); //(encoder q)
       std::cout << "Adding encoder channel" << std::endl;


       clk.addChannel ( "Orthosis/ai0" ); //solo se utiliza un canal analógico (ort-
hosis)
       std::cout << "Adding analog channel" << std::endl;


    /*************** Inicialización de la adquisición de datos ***************/

    encoder.start ( ); // Trigger encoders first, then analog input to provide clock
    std::cout << "Starting encoder" << std::endl;

    int j = 0;       //variable muestreo del encoder y almacenamiento en array
    int jd = 0;


    memset ( buffer , 0 , sizeof( buffer ) );


    if ( s.Open ( PORT , 57600 ) )
    {
        double t;
        char date[50];
        double freq = 100.0;
        double fpar = 0.001;
        time_t now = time ( NULL );
        ULONG frame = 0;
        hrtimer timer;
        tm timestr;

        localtime_s ( &timestr , &now );
        strftime ( date , 50 , "%Y-%m-%d %H.%M.%S.dat" , &timestr );

        while ( s.ReadDataWaiting ( ) < 100 );

        std::cout << "Boot delay: " << timer.t ( ) << " s" << std::endl;

        s.SendData ( "#oqb#s00" , 9 ); //para cuaterniones

        while ( strcmp ( "#SYNCH00\r\n" , buffer ) )
        circshift ( 10 );

        clk.start ( );
        std::cout << "Starting analog input to provide clock" << std::endl;

        timer.reset ( );

        while ( !_kbhit ( ))// && j < NOUT) //mientras no se pulse ninguna tecla
        {
            if ( s.ReadDataWaiting ( ) >= BUFSIZE )
            {
                char XOR = 0;
```

```
                s.ReadData ( buffer , BUFSIZE );

                for ( int i = 0; i < BUFSIZE - 1; i++ )
                    XOR ^= buffer[i];

                while ( header != 0 || checksum != XOR )
                {
                    XOR ^= ~header ^ checksum;
                    circshift ( BUFSIZE );
                }

                t = timer.t ( );
                freq = fpar * ( ( double ) frame++ / t ) + ( 1 - fpar ) * freq;


                for ( int i = 0; i < NFLOATS; i++ )
                {

                    data[i].push_back ( q[i] );

                }

                /******************    Lectura datos encoder  ******************/

                jd = encoder.peek ( &encj );
                cout << jd << " " << j << " " << encj << endl;

                temp[jd] = encj;


                if ( jd - ( jd - 1 ) > 1 )
                    temp[j] = temp[jd - 1];
                else
                    temp[j] = encj;

                j++;
            }

            else
                std::this_thread::sleep_for ( std::chrono::milliseconds ( 5 ) );

        }//while (!kbhit)

        encoder.write ( "ENC");

        saveFile ( "IMU");
        return 0;
    }
    else
        return 1;}//main
```

11.5. Control de velocidad

Control.h

```
#ifndef CONTROL_H
#define CONTROL_H

#include <math.h>
#include <vector>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

#include "hrtimer.h"
#include "Schmitt.h"
#include "EPOS2.h"


#define pi 3.14159265359
#define conv_to_rad pi/180.0


using namespace std;
using namespace std::chrono;
using std::ofstream;


// Functor that controls a motor depending on encoder and FSR inputs
class motorControl
{
private:
	static int nc;

	enum mode_t{STANCE, SWING};

	int mcid;
	mode_t mode;
	hrtimer timer;
	double initOppAnk, oldOppAnk;
	maxonMotor &motor_;

	double kr_, ks_, kw_, ia_, fa_, to_, th_, cutoff_;
	status_t oldOwnSt;

	double control_input [2];  //los inputs al control son los ángulos de tobillos
	double control_output [2]; //los outputs son las velocidades. Se filtran para
obtener menos retrasos (derivada de la posición, output original)


public:

	std::ofstream data; //almacenamiento de datos para exportar a Matlab y grafi-
car

	motorControl (maxonMotor &motor,                // Motor
```

```
                double kr = 0.87,            // Knee rotation range (degrees). Now radi-
ans
                double ks = 1.2128,         // Peak translation (0 = symmetric)
                double kw = -0.3563,        // Peak width (0 = normal)

                double ia = -0.6605, // Initial angle w.r.t. neutral position (de-
grees). Now radians
                double fa = 0.1599,  // Final angle w.r.t neutral position (degrees).
Now radians
                double to = 0.20,    // Timeout for stance detection (seconds)
                double th = 05.0,    // Threshold ankle angle for stance (degrees)
                float cutoff = 200.0) :           //frecuencia anterior de 0,5Hz

                motor_ (motor), oldOwnSt (HIGH), oldOppAnk (0), mode (STANCE)
        {
                kr_ = kr; ks_ = ks; kw_ = kw; ia_ = ia; fa_ = fa; to_ = to; th_ = th;
mcid = ++nc; cutoff_ = cutoff;

                //administración exportación de archivos

                if (mcid == 1) //identificador de pierna activa
                        data.open ("datos1.dat", ios::out);

                if (mcid == 2)
                        data.open ("datos2.dat", ios::out);

                memset (control_input, 0, 2 * sizeof(double)); //inicializar a 0 las
variables de control
                memset (control_output, 0, 2 * sizeof(double));
        }

        ~motorControl () { nc--; data.close (); }


        double operator()(const double oppAnk, const status_t ownSt, const status_t
oppSt)
        {

                static double outAngle = 0;

                //***First filter order
                double fltPar = 0.0;              //representa la a en el filtro
                float sampleRate = 100.0;         // 100 Hz

                if (cutoff_ > 0) //si la frecuencia de corte es mayor que 0
                        fltPar = 1.0 / (sampleRate / (2.0*pi*cutoff_) + 1.0); //Cuanto
más cercana esté a de 1, más se mayora la evaluación de entradas y menos la velocidad
en el instante anterior


                control_input [1] = control_input [0];
                control_input [0] = oppAnk;

                control_output [1] = control_output [0];
                control_output [0] = fltPar*(sampleRate*(control_input [0] - con-
trol_input [1])) + (1.0 - fltPar)*control_output [1];

                //std::cout << control_output [0] << std::endl;
```

```
		//exportar datos a archivos para plotear en matlab

		data << std::setw (10) << std::setprecision (8) << std::fixed << oppAnk
<< "\t" << sampleRate*(control_input [0] - control_input [1]) << "\t" << control_out-
put [0] << endl;

		//*****
		// Start cycle when own foot leaves the ground
		if (oldOwnSt == HIGH && ownSt == LOW)
		{
			timer.reset ();
			initOppAnk = oppAnk;
			mode = SWING;
		}

		switch (mode)
		{
		case SWING:
// Check whether the state is compatible with walking

		if (ownSt == HIGH)		//if foot is on the floor
		{
			mode = STANCE;
			motor_.gotoPosition(0);
			outAngle = 0;   //angulo de la rodilla contraria será 0
			break;
		}


		//Control de velocidad: ir a posición según ángulo tobillo opuesto


		if (control_output [0] > 0)
		{
			motor_.gotoPosition (motorAngle (oppAnk));
			outAngle = motorAngle (oppAnk)*((100 / (2 * pi * 200) + 1)/200);
//a la rodilla contraria se le pasa como entrada el ángulo del tobillo que pisa

		}
		break;

		case STANCE:
		// Go to straight position
			motor_.gotoPosition(0);
			outAngle = 0;
			break;
		}

		oldOwnSt = ownSt;
		oldOppAnk = oppAnk;


		return outAngle;
	}
```

```
	void setParameter(const double x, const int par)
	{
		switch (par)
		{
			case 0: kr_ = x; break;
			case 1: ks_ = x; break;
			case 2: kw_ = x; break;
			case 3: ia_ = x; break;
			case 4: fa_ = x; break;
			case 7: to_ = x; break;
			case 8: th_ = x; break;
			default:
				std::cout << "Unknown parameter" << std::endl;
		}
	}


	double motorAngle(const double a) //ojo que hay que pasar todo a radianes, "a"
original está en grados
	{
		double knee = 0.0;
		double vel = 0.0;
		double w = 4.0*atan(1.0)/(fa_-ia_);

		// Phase shift for tuning curve shape
		double phase = ks_*sin(w*(a-ia_)) + kw_*sin(2*w*(a-ia_));
		double fpar = 0;
		fpar = 1 / (100 / (2 * pi * 200) + 1);

		// Sine function with phase shift
		if (a > ia_ && a < fa_)
		{
			knee = kr_*(sin (w*(a - ia_) - phase));
			vel = fpar * 200 * knee;
		}


		return (vel);
	}


};#endif // CONTROL_H
```

Bucle principal

A continuación se especifica únicamente los cambios realizados en todo el bucle:

```
delta1 = rPoint_proxtb [2]; //coordendada z del punto
delta1_iz = rPoint_proxtb_i [2];

if (delta1 >= 0.115)
{
	rFSR = LOW;
	/*timer++;*/

}
else
{
	rFSR = HIGH;
	//timer = 0;
}


if (delta1_iz >= 0.11)
{
	lFSR = LOW;
	timer++;
}
else
{
	lFSR = HIGH;
	timer = 0;
}
if (rFSR == HIGH && lFSR == LOW)
{
	start = clock ();
	timer++;
}

if (lFSR == HIGH && rFSR == LOW)
	timer = 0;

if (timer == 0)
{
	temporizacion = (std::clock () - start) / CLOCKS_PER_SEC;
	cout << "temporizacion en segundos: " << temporizacion << endl; //TARDA 3se-
gundos en dar un paso completo
}


if (temporizacion >= 1 && temporizacion <= 3)
{
	theta_rodilla_dcha = rMotorControl (-theta_tibia_iz, rFSR, lFSR);
}
```

11.6. Código Matlab

Sensores

```
function sensor_lecture

%Lectura datos simulación de movimiento recogidos en codigo c++

load IMU
load ENC


%Conversión de Quaternion leído por la IMU a ángulo de rotación
%El ángulo interesante es el arcsen(Yz)

yaw = 0;
pitch = 0;
roll = 0;
theta_enc = 0;

for i=1:1:size(IMU,1)

    [yaw(i), pitch(i), roll(i)] = quat2an-
gle([IMU(i,1),IMU(i,2),IMU(i,3),IMU(i,4)]);

end

for j=1:1:size(ENC)
    theta_enc(j) = ENC(j);
end

% figure(1)
% plot(pitch*180.0/pi,'LineWidth',2)
% title('Angulo PITCH de la IMU')
% grid on
% figure(2)
% plot(theta_enc,'r','LineWidth',2)
% grid on
% title('Angulo descrito por el encoder')

figure(1)
plot(-pitch*180.0/pi,'LineWidth',2)
hold on
grid on
plot(theta_enc+15.6,'r','LineWidth',2)
title('Angulos sensores')
legend ('Pitch Imu','Encoder');
hold off


end
```

Cálculo eje de rotación

```
function matrixA

%Fémur. Sólido 11
%Tibia. Sólido 12
%Pie. Sólido 13

%%%%%%%%%%%%%%%%%%%%%%%%%%    BUCLE MOVIMIENTO   %%%%%%%%%%%%%%%%%%%%%%%%%%%
ruta='E:/Máster/__Trabajo Fin de Master/15   Código Movi-
miento/15_06_25__Movimiento_Cova/';
load([ruta 'qProj.dat']);
load([ruta 'qpProj.dat']);
load([ruta 'FPL1.dat']);
load([ruta 'FPL2.dat']);


numerador11 = 0;
numerador12 = 0;
denominador = 0;
theta_tobillo = zeros(1,1920);
theta_tobillo_izq = zeros(1,1920);
theta_rodilla = zeros(1,1920);
prod_vect=0;


for i=1:1:1920


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%INCISO: la ubicación en el archivo qProj y qpProj de los puntos      %
%proximales y distales así como de los vectores se encuentra a partir %
%de los archivos de datos ip,iv, pointmapping. Las columnas son fijas,%
%el movimiento se realiza por el paso de las filas (captura)          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %Sin derivar

    %femur
    u1 = [qProj(i,19); qProj(i,20); qProj(i,21)];
    v1 = [qProj(i,22); qProj(i,23); qProj(i,24)];

    %tibia
    u2 = [qProj(i,31); qProj(i,32); qProj(i,33)];
    v2 = [qProj(i,34); qProj(i,35); qProj(i,36)];

    %Pie
    u3 = [qProj(i,40); qProj(i,41); qProj(i,42)];
    v3 = [qProj(i,43); qProj(i,44); qProj(i,45)];

    %tibia izquierda
    u2iz = [qProj(i,70); qProj(i,71); qProj(i,72)];
    v2iz = [qProj(i,73); qProj(i,74); qProj(i,75)];

    %Pie izquierdo
```

```
    u3iz = [qProj(i,79); qProj(i,80); qProj(i,81)];
    v3iz = [qProj(i,82); qProj(i,83); qProj(i,84)];

    %femur
    pp1 = [qProj(i,4); qProj(i,5); qProj(i,6)];
    pd1 = [qProj(i,16); qProj(i,17); qProj(i,18)];

    %tibia
    pp2 = pd1;
    pd2 = [qProj(i,28); qProj(i,29); qProj(i,30)];

    %En el archivo rl.dat podemos encontrar las normas de los huesos, es
    %decir, su longitud

    %norma fémur -0.4136
    %norma tibia -0.421200476296

    w1 = -(pd1-pp1)/norm(pd1-pp1);
    w2 = -(pd2-pp2)/norm(pd2-pp2);
    w3 = cross(u3,v3);

    A11 = [u1 v1 w1];
    matrixA11(i,:) = reshape (A11,[1 9]);

    A12 = [u2 v2 w2];
    matrixA12 (i,:) = reshape (A12, [1 9]);

    A13 = [u3 v3 w3];

    A11'*A11;
    A12'*A12;

    %Cálculo ángulo del tobillo entre pie y tibia

   % angulo_tobillo = acos(u3'*u2);%se cumple que el coseno es el producto
escalar por ser vectores unitarios
    %  theta_rodilla (i) = atan2(norm(cross(u1,u2)),u1'*u2)*180.0/pi; %an-
gulo entre femur y tibia

    theta_tobillo (i) = atan2(norm(cross(u2,u3)),u3'*u2)*180.0/pi; %angulo
en el tobillo,entre pie y tibia
    theta_tobillo_izq (i) =
atan2(norm(cross(u2iz,u3iz)),u3iz'*u2iz)*180.0/pi;

    %Derivadas

    ud1 = [qpProj(i,19); qpProj(i,20); qpProj(i,21)];
    vd1 = [qpProj(i,22); qpProj(i,23); qpProj(i,24)];

    ud2 = [qpProj(i,31); qpProj(i,32); qpProj(i,33)];
    vd2 = [qpProj(i,34); qpProj(i,35); qpProj(i,36)];

    ppd1 = [qpProj(i,4); qpProj(i,5); qpProj(i,6)];
    pdd1 = [qpProj(i,16); qpProj(i,17); qpProj(i,18)];
```

```
    ppd2 = pdd1;
    pdd2 = [qpProj(i,28); qpProj(i,29); qpProj(i,30)];

    %norma fémur -0.4136
    %norma tibia -0.421200476296

    wd1 = -(pdd1-ppd1)/norm(pd1-pp1);
    wd2 = -(pdd2-ppd2)/norm(pd2-pp2);

    Ad11 = [ud1 vd1 wd1];
    Ad12 = [ud2 vd2 wd2];

    Ad11'*Ad11;
    Ad12'*Ad12;

    W11 = Ad11*A11'; %matriz antisimétrica sólido 11
    W12 = Ad12*A12'; %matriz antisimétrica sólido 12

    Wantir = W12-W11; %matriz antisimétrica resultante

    Wr=[Wantir(3,2);Wantir(1,3);Wantir(2,1)] ;   %vector Wr

    %Cálculo del vector er en coordenadas globales, sistema completo

    er = -Wr/norm(Wr);
    if Wr'*A11(:, 2) < 0
        er = -er;
    end

    erL11 = A11'*er;
    erL12 = A12'*er;

    erG11 = A11*erL11; %operacion de verificacion de igualdad.
erG11=erG12=er
    erG12 = A12*erL12;

    %Se calcula el vector er local óptimo corrigiendo el error generado
con
    %el cambio del vector W con el movimiento

    numerador11 = numerador11 + (erL11*norm(Wr)^5);
    numerador12 = numerador12 + (erL12*norm(Wr)^5);

    denominador = denominador + (norm(Wr)^5);

end

erLOptimo11 = numerador11/denominador;
erLOptimo12 = numerador12/denominador;

erLOptimo11 = erLOptimo11/norm(erLOptimo11);
erLOptimo12 = erLOptimo12/norm(erLOptimo12);
```

```
dlmwrite('erLOptimo11.dat',erLOptimo11,'delimiter','\t','precision',15);
type erLOptimo11.dat;
dlmwrite('erLOptimo12.dat',erLOptimo12,'delimiter','\t','precision',15);
type erLOptimo12.dat;

%Al corregir el error, los valores del vector er en coordenadas
%globales son diferentes a partir de ahora


%FIGURAS RELATIVAS A ÁNGULOS CALCULADOS EN BUCLE

figure(1) %ángulos

%t=linspace(0,length(theta_rodilla));


plot(theta_rodilla,'LineWidth',2)
hold on
plot(theta_tobillo,'r','LineWidth',2)
title('Ángulos en la marcha')
legend('rodilla','tobillo')
grid on
hold off


figure (2) %Pisadas. Ambos pies en el suelo mitad gráfica (doble apoyo)

plot (FPL1(:,3),'LineWidth',2)
hold on
plot(FPL2(:,3),'m','LineWidth',2)
title('Fuerza aplicada en pisada: pie en suelo/aire')
legend('Pie 1','Pie 2')
grid on
hold off


figure (3)

plot(theta_tobillo_izq, 'LineWidth',2)
hold on
plot(theta_rodilla,'r', 'LineWidth',2)
title('Ángulos durante marcha: Tobillo izquierdo/Rodilla derecha')
legend('tobillo izquierdo','rodilla derecha')
grid on
hold off


figure (4)

plot(theta_tobillo_izq,(theta_rodilla))
hold on
hy = graph2d.constantline(60, 'Color','m','LineWidth',2);
changedependvar(hy,'y');
grid on
```

```
title('Rodilla derecha vs Tobillo izquierdo')
hold off

s = 0.15;
r = zeros(3, 1);
r = r + sin(i*2*pi/25)/500;


A0 = Funcion_matriz_cte(erLOptimo11, erLOptimo12);
dlmwrite('matrizA0.dat',A0,'delimiter','\t','precision',15);
type matrizA0.dat;

%%%%%%%%%%%%%%%%%%%%%%%%%    ANIMACIÓN    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 for j=1:1:1920

     %femur
    pp1 = [qProj(i,4); qProj(i,5); qProj(i,6)];
    pd1 = [qProj(i,16); qProj(i,17); qProj(i,18)];

    %tibia
    pp2 = pd1;
    pd2 = [qProj(i,28); qProj(i,29); qProj(i,30)];

    %Utilización de la matriz A11 y A12 del bucle anterior para no reali-
zar nuevamente el cálculo

    a11=reshape(matrixA11(j,:),[3 3]); %Coordenadas globales (femur)
    a12=reshape(matrixA12(j,:),[3 3]); %Coordenadas globales (tibia)

    erGNuevo11 = a11*erLOptimo11;
    erGNuevo12 = a12*erLOptimo12;

    %Se calculan las proyecciones de los vectores u de tibia y femur sobre
    %el eje de giro de la rodilla

    u1p = a11(:,1)-(a11(:,1)'*erGNuevo11)*erGNuevo11;
    u2p = a12(:,1)-(a12(:,1)'*erGNuevo11)*erGNuevo11;

    %Se normalizan los vectores proyectados
    u1p_n = u1p/norm(u1p);
    u2p_n = u2p/norm(u2p);

    prod_vect = cross(u1p_n, u2p_n);
    norma = norm(cross(u1p_n,u2p_n));

    theta_rodilla (j) =
atan2(norm(cross(u1p_n,u2p_n)),(u1p_n'*u2p_n))*180.0/pi; %angulo entre fe-
mur y tibia

    %Se halla la matriz Ar que representa la visión del sólido 2 respecto
    %del sólido 1. La tibia vista desde el fémur

    %Ar=A11'*A12
```

```
    %matriz antisimétrica del vector representativo de la rodilla con eje
    %local del fémur

    mat_vec_rodilla = [0 erLOptimo11(3) -erLOptimo11(2);-erLOptimo11(3) 0
erLOptimo11(1);erLOptimo11(2) -erLOptimo11(1) 0];

    Ar_rodilla = eye(3)+ mat_vec_rodilla*sind(theta_rodi-
lla(j))+2*((mat_vec_rodilla)*mat_vec_rodilla)*(sind(theta_rodi-
lla(j)/2)^2);

    %Utilización de la fórmula de Rodríguez para hallar la matriz de
    %transformacion de la tibia con respecto al femur. Para ángulos muy
pequeños, Ar será la matriz identidad
    %3x3 aproximadamente. Se utiliza el ángulo de la rodilla para compro-
bación


%     %se calcula la matriz de transf de la tibia vista desde el fémur

     A2 = a11*Ar_rodilla*A0 ;   %A2 representa la matriz de transf de la
tibia relativa al fémur
     matrixA2(j,:) = reshape (A2,[1 9]);
     a2=reshape(matrixA2(j,:),[3 3]);    %matriz tibia coordenadas res-
pecto del fémur


%     %Se pretende mantener el ángulo relativo entre el pie y la tibia
     Ar_tobillo = A12'*A13;     %A13 es la matriz de transf original del
pie (solido 13)
     A3 = A11*Ar_rodilla*Ar_tobillo;


    figure(6)

    plot3(pp1(1), pp1(2), pp1(3), 'c.', 'markersize', 30) %cadera
    hold on
    plot3(pd1(1), pd1(2), pd1(3), 'c.', 'markersize', 30) %rodilla
    plot3(pd2(1), pd2(2), pd2(3), 'c.', 'markersize', 30) %tobillo

    %nuevo tobilllo en coordenadas locales del fémur
    plot3(-a2(1,3)*norm(pd2-pd1)+pd1(1), -a2(2,3)*norm(pd2-pd1)+pd1(2), -
a2(3,3)*norm(pd2-pd1)+pd1(3), 'k.', 'markersize',30)

    femur=line([pp1(1),pd1(1)],[pp1(2), pd1(2)],[pp1(3), pd1(3)],'Co-
lor',[0.5 0.5 0.5],'linewidth',2.5); %recta que representa el fémur
    tibia=line([pd1(1),pd2(1)],[pd1(2), pd2(2)],[pd1(3), pd2(3)],'Co-
lor',[0.5 0.5 0.5],'linewidth',2.5); %recta que representa la tibia

    tibia_secun=line([pd1(1),-a2(1,3)*norm(pd2-pd1)+pd1(1)],[pd1(2), -
a2(2,3)*norm(pd2-pd1)+pd1(2)],[pd1(3), -a2(3,3)*norm(pd2-pd1)+pd1(3)],'Co-
lor','r','linewidth',2.5); %recta que representa la tibia
```

```
    %ejes cadera
    vplot(pp1, a11(:,1), s, [1.0 0.0 0.0], 2)
    vplot(pp1, a11(:,2), s, [0.0 0.5 0.0], 2)      %0.5 equivale a verde
oscuro
    vplot(pp1, a11(:,3), s, [0.0 0.0 1.0], 2)

    %ejes rodilla
    vplot(pd1, a12(:,1), s, [1.0 0.0 0.0], 2)      %R
    vplot(pd1, a12(:,2), s, [0.0 0.5 0.0], 2)      %0.5 equivale a verde
oscuro %G
    vplot(pd1, a12(:,3), s, [0.0 0.0 1.0], 2)      %B

    % vplot(pd1, a2(:,3), s, [1.0 1.0 0.0], 3)       %amarillo nueva ma-
triz transformación (tibia respecto femur)

    %ejes globales nuevos
    vplot(pd1, erGNuevo11, s, [1.0 0.0 1.0], 2)      %magenta
    vplot(pd1, erGNuevo12, s, [0.0 1.0 0.0], 2)      %verde claro

    %punto origen y escalado de los ejes
    %plot3(r(1), r(2), r(3), 'k.', 'markersize', 20) %punto gordo origen

    set(gca, 'dataaspectratio', [1 1 1])    %igual escala tres ejes con
1,5m de margen
    axis([-1 1.2 -0.5 0.5 0 1])

    grid on

    hold off

  %  view(90, 0)

pause(0.1)

 end


figure (5)

plot(theta_tobillo,(theta_rodilla))
hold on
hy = graph2d.constantline(60, 'Color','g','LineWidth',2);
changedependvar(hy,'y');
grid on
title('Rodilla derecha vs Tobillo derecho')
hold off
function vplot(r, v, scale, color, width)

x = [r(1) r(1) + scale*v(1)];
y = [r(2) r(2) + scale*v(2)];
z = [r(3) r(3) + scale*v(3)];

plot3(x, y, z, 'color', color, 'linewidth', width)
```

Función matriz constante

```
function A0 = Funcion_matriz_cte(u1, u2)

fun = @(x) ecuacion(x,u1,u2);

A0 = reshape(fsolve(fun, [1 0 0 0 1 0 0 0 1]), 3, 3);

function y = ecuacion(x,u1,u2)

y(1) = x(1)*u2(1)+x(4)*u2(2)+x(7)*u2(3)-u1(1);
y(2) = x(2)*u2(1)+x(5)*u2(2)+x(8)*u2(3)-u1(2);
y(3) = x(3)*u2(1)+x(6)*u2(2)+x(9)*u2(3)-u1(3);

y(4) = x(1)^2+x(2)^2+x(3)^2-1;
y(5) = x(4)^2+x(5)^2+x(6)^2-1;
y(6) = x(7)^2+x(8)^2+x(9)^2-1;

y(7) = x(1)*x(4)+x(2)*x(5)+x(3)*x(6);
y(8) = x(1)*x(7)+x(2)*x(8)+x(3)*x(9);
y(9) = x(4)*x(7)+x(5)*x(8)+x(6)*x(9);

y(10) = x(3);
```

Ploteado Filtro

```
function ploteadoFiltro

ruta='E:/Máster/__Trabajo Fin de Master/15   Código Movi-
miento/15_06_25__Movimiento_Cova/';
load([ruta 'datos1.dat']);
load([ruta 'datos2.dat']);
load([ruta 'FPL1.dat']);
load([ruta 'FPL2.dat']);
load([ruta 'datos_tibia.dat']);


for i=1:157

    angulo = datos1(i,1);
    derivada = datos1(i,2);
    filtro = datos1(i,3);

end

% figure (1)
% plot(datos1,'LineWidth',2)
% title('Pierna 1')
% legend('angulo','derivada','filtro')
% grid on
% figure (2)
% plot (datos2,'LineWidth',2)
% title('Pierna 2')
% legend('angulo','derivada','filtro')
```

```
% grid on
% figure(3)
% plot([datos1(:, 2) gradient(datos1(:, 1), 0.01)],'LineWidth',2)
% title('Comparativa angulo y derivada')
% grid on

figure(4)
plot(datos1(:,1),'g','LineWidth',2)
hold on
plot((FPL1(:,3)/1000),'LineWidth',2)
plot(-datos_tibia(:,1)/4,'m','LineWidth',2)
hold off
title('Fuerza placas1 vs angulo tobillo vs angulo tibia')
legend('angulo rodilla', 'FPL1','angulo Tibia dcha','angulo tibia iz')
grid on

figure(5)
plot(datos2(:,1),'r','LineWidth',2)
hold on
plot((FPL2(:,3)/1000),'LineWidth',2)
hold off
title('Fuerza placas2 vs angulo')
legend('angulo 2', 'FPL2')
grid on

figure(6)
plot(-datos_tibia(:,1),'m','LineWidth',2)
hold on
plot(-datos_tibia(:,2),'LineWidth',2)
hold off
title('Angulo tibia dcha/izq')
legend('Derecha', 'Izquierda')
grid on

figure(7)
plot(datos_tibia(:,3),'m','LineWidth',2)
hold on
plot((FPL1(:,3)/100),'LineWidth',2)
hold off
title('Angulo pie dcha/FPL1')
legend('pie derecho', 'fpl1')
grid on

figure(8)
plot(datos_tibia(:,4),'m','LineWidth',2)
hold on
plot((FPL2(:,3)/100),'LineWidth',2)
hold off
title('Angulo pie izquierdo /FPL2')
legend('pie izquierdo','fpl2')
grid on

figure(9)
plot(FPL1(:,3),'LineWidth',2)
grid on
```

Cálculo de la posición del pie

```
function Posicion_pie

ruta='E:/Máster/__Trabajo Fin de Master/15   Código Movi-
miento/15_05_04__Movimiento_control/';
load([ruta 'datos_posicionpie_plot.dat']);
load([ruta 'FPL2.dat']);

talon = datos_posicionpie_plot(:,2);
punta = datos_posicionpie_plot(:,1);
theta_femur = datos_posicionpie_plot(:,3);
theta_tibia = datos_posicionpie_plot(:,4);

figure(1)
plot(talon,'r','LineWidth',2)
hold on
plot(punta,'LineWidth',2)
plot(FPL2(:, 3)/1000, 'k')
hold off
title('Posiciones talon y punta pie derecho')
legend('talon','punta')
grid on

figure(2)
plot(theta_femur*180.0/pi,'r','LineWidth',2)
hold on
plot(theta_tibia*180.0/pi,'LineWidth',2)
hold off
title('Angulos femur y tibia derechos')
legend('femur','tibia')
grid on

figure(3) %se puede realizar el control mediante la medida de velocidad
del femur y de la tibia
plot(gradient(theta_femur, 0.01),'LineWidth',2)
hold on
plot (gradient(theta_tibia, 0.01),'r','LineWidth',2)
hold off
title('derivadas angulos femur y tibia')
legend('velocidad femur','velocidad tibia')
grid on
```

Optimización de los parámetros de control: minimización del error

```
function x = Optimizacion_angulos

x0 = [0.8; -0.15; -0.45; 0.20]; %valores iniciales
x0 = [0; 0; -1; 1]; %valores iniciales
lb = [-2.0 -2.0 -1.0 -1.0];  %valores límite inferior
ub = [2.0 2.0 1.2 2.2];      %valores límite superior

% x = fmincon(@funcion_objetivo, x0,[],[],[],[],lb,ub);
```

```
figure (1)

options=gaoptimset('generations',2000,'populationSize',800,'TolFun',1e-
8,'TolCon',1e-8,'HybridFcn',{@fmincon},'PlotFcns',@gaplotbestf);

x = ga(@funcion_objetivo,4,[],[],[],[],lb,ub,[],options);
% [x,fval] = simulannealbnd(@funcion_objetivo,x0,lb,ub);

ruta='E:/Máster/__Trabajo Fin de Master/15   Código Movi-
miento/15_06_25__Movimiento_Cova/';
load([ruta 'datos_angulos_plot.dat']);
load([ruta 'FPL2.dat']);

theta_tibia = datos_angulos_plot(:, 1);
theta_rodilla = datos_angulos_plot(:, 2);
theta_pie_iz = datos_angulos_plot(:,4);

theta_rodilla_control = Funcion_angulo(-theta_tibia, 0.87, x(1), x(2),
x(3), x(4));

figure (2)
plot(theta_rodilla,'r','LineWidth', 2)
hold on
plot(theta_tibia, 'k:')
plot(theta_rodilla_control,'LineWidth', 2)
% plot(datos_angulos_plot(:,3),'g','linewidth',2)
hold off
title('Ángulos de rodilla de referencia, tibia iz y rodilla controlada')
legend('rodilla de referencia','tibia izquierda','rodilla controlada','ro-
dilla c++')
grid on


function theta_rodilla= Funcion_angulo(a,kr,ks,kw,ia,fa)

theta_rodilla=zeros(size(a));
w= pi/(fa-ia);

aon = a(a > ia & a < fa);

phase = ks*sin(w*(aon-ia)) + kw*sin(2*w*(aon-ia));

theta_rodilla(a > ia & a < fa) = kr*(sin(w*(aon-ia) - phase));


% figure(3)
% plot(a,'r','LineWidth',2)
% hold on
% plot(theta_tibia,'g','LineWidth',2)
% hold off
% title('Ángulo de tobillo y ángulo de rodilla con factores')
% legend('tobillo','rodilla')
% grid on
```

```
function f =funcion_objetivo(x)

load('datos_angulos_plot.dat')

frames = 80:130;

for i=frames

theta_tibia(i) = datos_angulos_plot(i, 1);
theta_pie_iz(i) = datos_angulos_plot(i,4);

% theta_control(i) = Funcion_angulo(-theta_tibia(i)-theta_pie_iz(i),
1.117, x(1), x(2), x(3), x(4));
theta_control(i) = Funcion_angulo(-theta_tibia(i), 1.117, x(1), x(2),
x(3), x(4));
theta_referencia(i) = datos_angulos_plot(i,2);

end

%La minimización del error interesará entre los frames 80 y 125, donde se
%alcanza el ángulo máximo de marcha

sumatorio= sum((theta_control(frames)-theta_referencia(frames)).^2);

f=sqrt(sumatorio/length(frames)); %125-80 son el rango de frames donde in-
teresa el cálculo
```