tecnun Universidad de Navarra

Proyecto Fin de Grado

INGENIERIA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO Y DESARROLLO DE UNA APLICACIÓN PARA LA VISUALIZACIÓN DE LOS RESULTADOS DE UNA CAPTURA DE MOVIMIENTO EN UN MODELO 3D EN MATLAB

Ignacio Pardo Tomás

San Sebastián, septiembre de 2020

Pº Manuel Lardizabal, 13. 20018 Donostia-San Sebastián, Gipuzkoa
Tel. 943 219 877 · Fax 943 311 442 · www.tecnun.es

Agradecimientos

Antes de comenzar el proyecto, me gustaría dedicar unas pequeñas palabras de agradecimiento a todas aquellas personas que han estado presentes durante mi paso por las Universidad. Sin todos vosotros no habría sido posible llevar a cabo este trabajo.

En primer lugar, me gustaría agradecer a Sergio, tutor de este proyecto, por tu ayuda, paciencia y dedicación en el transcurso de éste.

En segundo lugar, a mis amigos y compañeros que, durante estos cuatro años, me habéis acompañado siempre y me habéis hecho vivir momentos inolvidables.

En tercer lugar, a mi familia, en especial a mi padre y a mi madre, que habéis estado a mi lado siempre, guiándome y aconsejándome, y me habéis brindado la oportunidad de llegar hasta aquí gracias a vuestro esfuerzo.

Por último, a Lucía, porque me has apoyado siempre, me has ayudado en los peores momentos y has hecho que los buenos sean aún mejores. Por ser lo mejor que me llevo de estos cuatro años.

Índice de contenidos

Índice de figuras

Índice de tablas

1. RESUMEN

El desarrollo completo del Proyecto se ha dividido en dos partes principales. La primera de ellas ha consistido en el diseño y la programación del software. Por otro lado, en la segunda etapa se ha completado el manual de uso del software en cuestión.

Primero, se fijaron los objetivos, alcanzables y concretos, que se debían completar durante el transcurso del proyecto. El objetivo principal era desarrollar una aplicación en Matlab que funcionase conjuntamente con el software de reconstrucción de movimiento MotRec, permitiendo visualizar en un modelo 3D el movimiento capturado mediante un sistema de captura de movimiento y permitiese al usuario interactuar con ella. Así, se pretendía sustituir el software Compamm 5.1 que hasta ahora se utilizaba para cumplir dicha función en la asignatura de Biomecánica y Biorrobótica en Tecnun.

Asimismo, se establecieron las características de diseño, funcionalidad e interactividad que debía tener la aplicación que se iba a programar para sustituir al software empleado hasta el momento para visualizar el movimiento en modelos 3D. El software debía resultar atractivo al usuario, fácil de manejar y con controles estándares para ejecutar la animación, como por ejemplo botones de *Pause/Play*, avanzar y retroceder un fotograma y una barra slider entre otros. Por otra parte, debía funcionar en sistemas operativos de Windows y Mac y ser ejecutable en la versión 2020a de Matlab.

Con las bases del proyecto bien claras, se procedió al diseño de la aplicación en el entorno de programación Matlab 2020a. Debido a que se partía de cero, esta fase ha resultado ser la más costosa de todo el proyecto en lo que a complejidad y tiempo se refiere. El método de trabajo seguido ha consistido en ir añadiendo funcionalidades a la aplicación, comenzando por las más básicas, para después pulir detalles y hacer la interfaz de usuario lo más atractiva y entendible posible. Ha habido que modificar también el código del software MotRec para hacerlo compatible tanto con los sistemas operativos de Windows y Mac como con la propia aplicación.

Una vez verificado el correcto funcionamiento de la app y haber asegurado que cumplía todos los objetivos y requisitos impuestos al inicio del proyecto, se dio por finalizado el proceso de diseño y programación de la aplicación.

Por último, se completó el manual de uso del software MotRec ya existente. Se añadió una sección en la parte final del manual en la que se explica el funcionamiento de la aplicación y cuáles son los controles y conocimientos mínimos que debe dominar el usuario acerca de ella.

2. ABSTRACT

The whole development of the Project has been divided into two main parts. The first of these has consisted of the design and programming of the software. On the other hand, in the second stage, the user manual of the software has been completed.

First, achievable and concrete objectives were set to be completed during the course of the project. The main objective was to develop an application in Matlab that would work together with the MotRec motion reconstruction software, allowing the motion captured by a motion capture system to be viewed in a 3D model and allowing the user to interact with it. Thus, it was intended to replace the Compamm 5.1 software that, until now, was used to fulfill this function in the Biomechanics and Biorrobotics subject at Tecnun.

Likewise, the design, functionality and interactivity characteristics that the application to be programmed should have were established. The software had to be attractive to the user, easy to handle and with standard controls to execute the animation, such as *Pause*/*Play* buttons, move forward and backward one frame and a slider bar, among others. On the other hand, it had to work on Windows and Mac operating systems and be executable in Matlab 2020a.

Once the project bases were clear, the design of the application in the Matlab 2020a programming environment began. Since it started from scratch, this phase has turned out to be the most expensive of the entire project in terms of complexity and time. The working method followed has consisted of adding functionalities to the application, starting with the most basic ones, and then polishing details and making the user interface as attractive and understandable as possible. The MotRec software code also had to be modified to make it compatible with both Windows and Mac operating systems and the application itself.

Once the correct operation of the app had been verified and it had been ensured that it met all the objectives and requirements imposed at the beginning of the project, the application design and programming process was concluded.

Finally, the existing MotRec software user manual was completed. A section has been added in the final part of the manual in which the operation of the application is explained and what are the minimum controls and knowledge that the user must master about it.

3. INTRODUCCIÓN

La captura de movimiento o MoCap (abreviatura del nombre en inglés *Motion Capture)* es una técnica que consiste en la grabación del movimiento de un sujeto u objeto real para después importarla a una computadora, y mediante un software determinado visualizar dichos patrones de movimiento de manera virtual en un modelo 3D [1].

Este método se emplea en diferentes áreas como pueden ser la investigación médica o el deporte, aunque es cierto que su uso está considerablemente más extendido en la industria cinematográfica y los videojuegos [1].

Tecnun dispone de un software desarrollado por el grupo de simulación de CEIT en el marco del proyecto europeo DHErgo [2]. Este software llamado MotRec, y que funciona en código de Matlab, permite calcular todos los parámetros necesarios para reconstruir un modelo 3D a partir del movimiento capturado mediante un sistema de MoCap.

Una vez hecha la reconstrucción virtual del movimiento, se necesita otro software para visualizar los resultados. En la actualidad existen multitud de este tipo de softwares. En las próximas líneas se mencionan algunos de ellos.

El primero es *OpenSim*. Se trata de un software gratuito que fue lanzado en 2007, accesible por cualquier persona y que permite desarrollar modelos de estructuras musculoesqueléticas y crear simulaciones dinámicas de movimiento [3].

Existe también una toolbox programada por Omid Heidari en el entorno de programación Matlab [4]. Este software fue inicialmente diseñado para temas de cinemática y síntesis de robots, pero también permite visualizar y comparar datos capturados mediante un sistema de MoCap.

Compamm es un software de simulación multicuerpo basado en coordenadas naturales [5] que fue desarrollado por el grupo de simulación de CEIT. Este programa es el que actualmente se emplea en la Escuela en la asignatura de Biomecánica y Biorrobótica para visualizar el movimiento capturado. Además de mostrar los resultados de una captura de movimiento, el software es capaz de desempeñar otra gran variedad de funciones.

Sin embargo, Compamm dejó de desarrollarse en 2003 y se prevé que debido a la falta de mantenimiento dejará de funcionar por problemas de compatibilidad con futuras versiones de Windows y OpenGL. Además, teniendo en cuenta que la Escuela ya dispone de la licencia de Matlab, resultaría muy interesante disponer de una aplicación que funcione en Matlab, posibilitando así que tanto la reconstrucción como la visualización puedan realizarse en este mismo programa.

4. OBJETIVOS

4.1. Objetivo principal

- El objetivo principal es desarrollar una aplicación en el entorno de programación Matlab que trabaje conjuntamente con el software de reconstrucción de movimiento MotRec, permitiendo visualizar en un modelo 3D el movimiento capturado mediante un sistema de MoCap y dé la posibilidad al usuario de interactuar con ella. De esta manera, se pretende sustituir el software Compamm 5.1 que actualmente se utiliza para visualizar los resultados en 3D en la asignatura de Biomecánica y Biorrobótica en Tecnun por la nueva aplicación en Matlab.

4.2. Objetivos secundarios

- Asegurar el correcto funcionamiento de la aplicación desarrollada en la versión 2020a de Matlab.

- Asegurar que el software desarrollado es capaz de desempeñar todas las funciones en su totalidad tanto en el sistema operativo de Windows como en el de Mac.

- El usuario debe gozar de la posibilidad de manipular la escena con facilidad ya sea rotando, haciendo o quitando zoom, o moviendo el punto de vista.

- Los controles de la aplicación que permitan la interacción con el usuario deben ser sencillos y parecidos a los estándares de este tipo de softwares. Estos controles deben incluir al menos las siguientes funcionalidades:

    - Ejecutar la animación hacia delante y hacia atrás
    - Avanzar y retroceder un fotograma
    - Pausar la animación
    - Slider para avanzar y retroceder la animación a mano
    - Ocultar y mostrar segmentos, marcadores y sistemas de coordenadas individualmente
    - Mostrar el fotograma actual y el tiempo en segundos al usuario

- El software debe tener la capacidad de leer correctamente ficheros de formato stl, independientemente de su tamaño, mostrarlos y ofrecer la posibilidad al usuario de interactuar con ellos.

- El software debe ofrecer también la posibilidad de visualizar los segmentos del modelo 3D en modo Wireframe, en el cual se dan unos determinados puntos y se unen mediante cilindros.

5. DISEÑO DE LA APLICACIÓN

La aplicación se ha diseñado de manera que se compone de dos ventanas independientes. El nombre de la aplicación es MotVis3D. Como se puede apreciar en la Figura 1, cuando se inicia la aplicación, en la parte izquierda de la pantalla por defecto aparece el panel de control de usuario y en la derecha una ventana donde se muestra el modelo 3D.

Como es lógico, y a pesar de que las ventanas son diferentes, los componentes del panel izquierdo afectan a la animación 3D que se muestra en la venta del lado derecho.

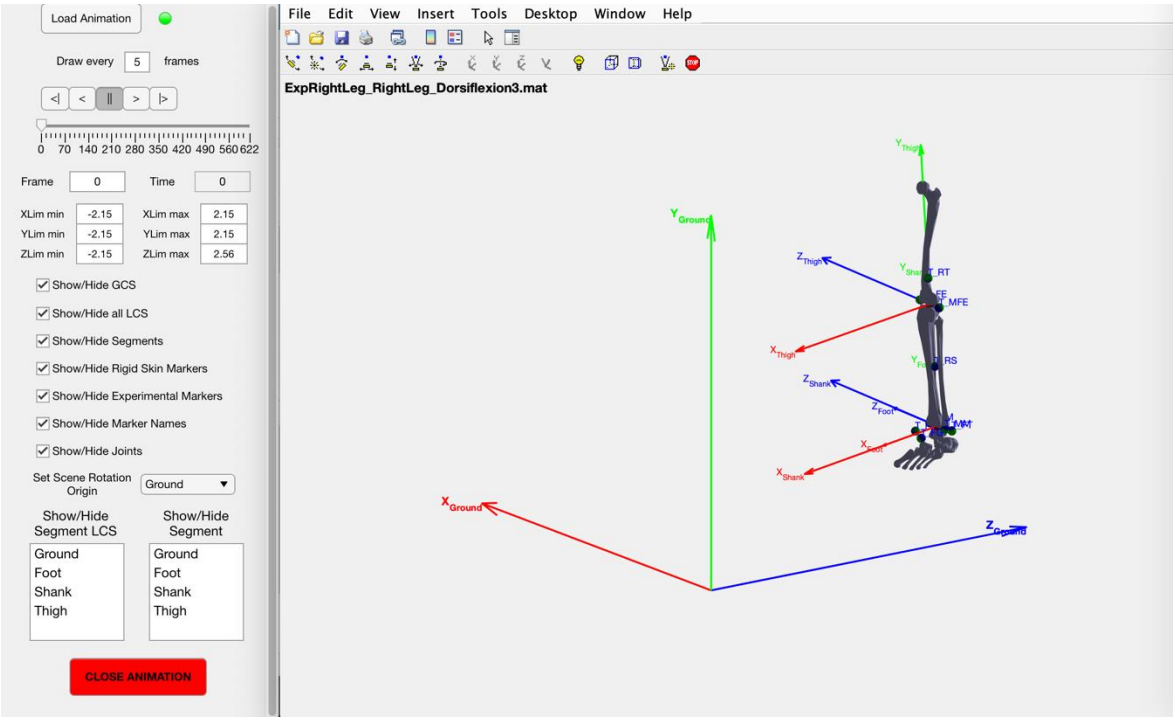


Figura 1. Vista general de la aplicación

5.1. Panel de control del usuario

Esta ventana contiene todos los botones y componentes necesarios para que el usuario interactúe con la animación, salvo los controles para modificar el punto de vista de la escena como por ejemplo rotar, hacer zoom o desplazar el objetivo del punto de vista.

5.1.1. Arrancar la simulación

Una vez se ha iniciado la aplicación, al principio se muestra al usuario únicamente el panel de control a la izquierda de la pantalla de su ordenador. Pulsando el botón *LOAD ANIMATION* se abre una nueva ventana como la que se muestra en la Figura 2, donde se puede seleccionar el fichero con extensión .mat que contiene los resultados de la captura de movimiento que se desea visualizar.

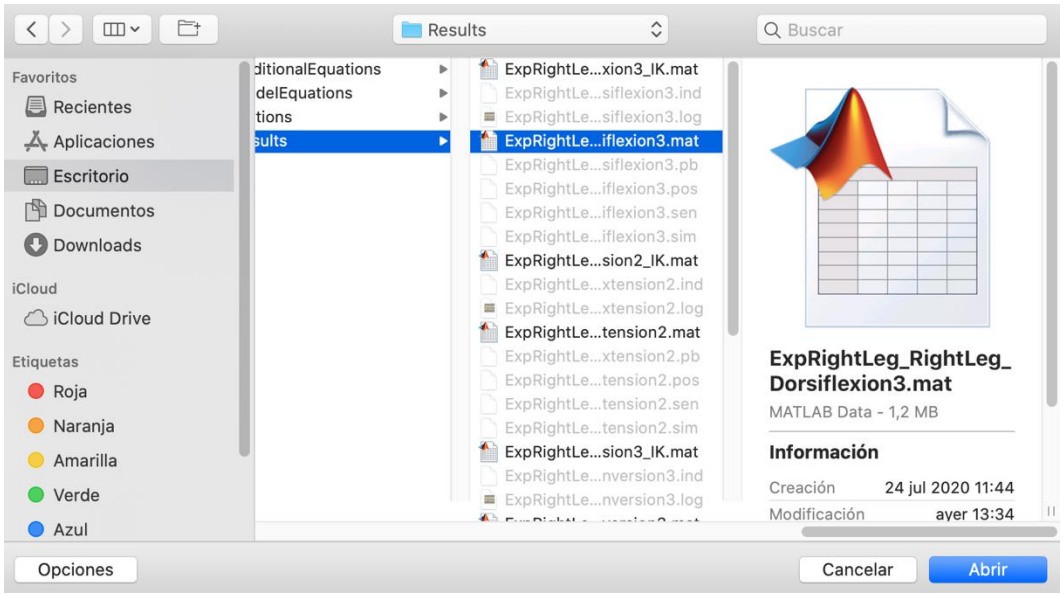


Figura 2. Ventana para seleccionar el archivo .mat con los resultados

La luz de al lado del botón para cargar la animación se mantiene roja mientras no se seleccione ningún fichero o el fichero seleccionado no tenga el formato correcto. En el momento que el fichero se carga correctamente y la animación está lista para ejecutarse, la luz cambia a verde y aparece la ventana con el modelo 3D. Estas dos situaciones se plasman en las Figuras 3 y 4 de abajo.

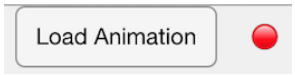


Figura 3. Botón Load Animation antes de cargar la animación

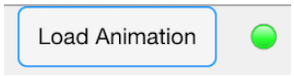


Figura 4. Botón Load Animation después de cargar la animación

5.1.2. Durante la simulación

En la Figura 5 aparecen los botones para controlar la ejecución de la animación. El usuario tiene la posibilidad de ejecutar la animación hacia adelante (con el botón “>”) o rebobinarla (con el botón “<”). También puede pausarla con el botón central (“||”). Con el movimiento pausado, es posible avanzar o retroceder un fotograma con los botones “|>” y “<|” respectivamente.

Figura 5. Botones para controlar la ejecución de la animación

También se ha añadido una barra a modo de slider, como la mostrada en la Figura 6, que permite al usuario controlar la ejecución de la animación a mano y a la velocidad que desee. Los valores que aparecen en la barra representan los fotogramas de la animación.

Figura 6. Slider

Por otra parte, dependiendo del modelo de ordenador empleado y de su tarjeta gráfica, la velocidad y fluidez de la simulación pueden variar ligeramente. Por esta misma razón, se ha introducido en la app un campo que permite controlar la velocidad de ejecución (Figura 7). Esto permite al usuario ajustarla lo mejor posible a las características del ordenador y obtener los mejores resultados.

Figura 7. Control de velocidad de ejecución

Debajo de la slider se han situado dos campos que muestran el fotograma y el tiempo actual en segundos como se puede ver en la Figura 8. El campo *frame* es editable, por lo que permite al usuario introducir el fotograma deseado directamente y visualizarlo rápidamente en la pantalla.

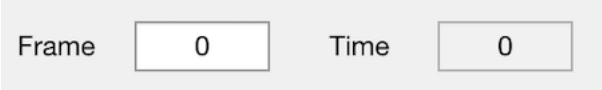


Figura 8. Frame box y Time box

Si bien es cierto que al cargar el modelo el software automáticamente calcula los límites de la ventana 3D para se vean todos los objetos incluyendo los sistemas de coordenadas y marcadores, el usuario tiene la posibilidad de cambiar los límites de los ejes. En la Figura 9 se ven los límites actuales de los ejes. Simplemente indicando un nuevo valor en cualquiera de los campos la escena se reajusta automáticamente.

Figura 9. Límites de los ejes

Si el software detecta que el límite inferior es mayor o igual que el superior o viceversa, muestra por pantalla al usuario un mensaje de error como el de la Figura 10 y mantiene el límite actual.

Figura 10. Aviso de error en los límites de los ejes

Debajo de los campos que muestran los límites de los ejes, se han situado una serie de *checkbox* que permiten mostrar u ocultar ciertos objetos gráficos en la ventana 3D.

Como se puede leer en la Figura 11 los elementos que se pueden ocultar o mostrar, siguiendo el orden de la lista de la figura, son:

- Los sistemas de coordenadas global y local de cada segmento
- Los propios segmentos
- Los marcadores modelo y los experimentales
- Los nombres de los marcadores
- Las articulaciones

Cuando el checkbox está seleccionado los componentes se muestran en la ventana de visualización 3D y cuando está desactivado se ocultan.

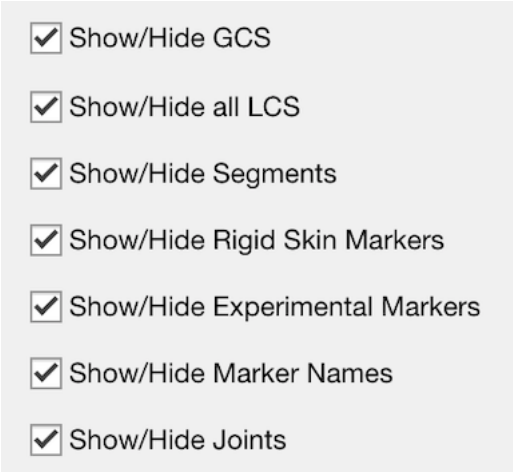


Figura 11. Checkbox para mostrar y ocultar objetos gráficos en la ventana 3D

Pero además de esto, la app también ofrece la posibilidad de mostrar u ocultar algunos objetos gráficos de manera individual. En la Figura 12 se pueden ver los controles que se han instalado en la parte inferior del panel de control. Estos componentes muestran los nombres de todos los segmentos del modelo. La lista de la izquierda permite al usuario mostrar u ocultar individualmente el sistema de coordenadas local de cada segmento. Al clickar en el nombre de un segmento la visibilidad del sistema de coordenadas local de dicho segmento cambia, es decir, si está oculto se vuelve visible y viceversa.

El componente de la derecha tiene un funcionamiento análogo, pero muestra y oculta el propio segmento.

Figura 12. Controles para mostrar y ocultar individualmente los segmentos y sus sistemas de coordenadas

Por último, cuando se rota el punto de vista de la escena, se hace por defecto respecto al origen del sistema de coordenadas global. Sin embargo, resulta de gran utilidad, cuando por ejemplo se desea ver en detalle uno de los segmentos del modelo, poder girar el punto de vista respecto al origen del sistema de coordenadas local de ese segmento en concreto.

Por esta razón, se ha añadido un componente que desempeña esta función. En la Figura 13 mostrada abajo, al seleccionar uno de los segmentos en el menú desplegable, el origen de rotación de la escena cambia automáticamente al origen del sistema de coordenadas local de ese segmento.

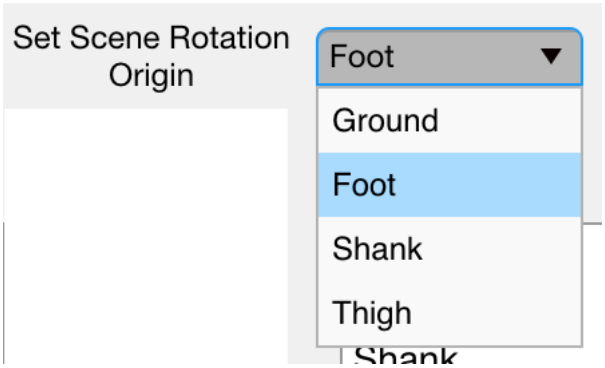


Figura 13. Control para seleccionar el origen de rotación de la escena

5.1.3. Cerrar la simulación

La aplicación puede cerrarse en cualquier momento. En la parte inferior del panel se ha colocado un botón rojo en el que se puede leer “CLOSE ANIMATION”, como se aprecia en la Figura 14 de abajo.

Figura 14. Botón para cerrar la aplicación

Al pulsar este botón, la animación se interrumpe inmediatamente, y a continuación se cierran tanto la ventada del modelo 3D como el panel de control del usuario.

5.2. Ventana del modelo 3D

Al margen del panel de control del usuario se encuentra la ventana donde se visualiza el movimiento en un modelo 3D. Esta ventana se muestra en la Figura 15.

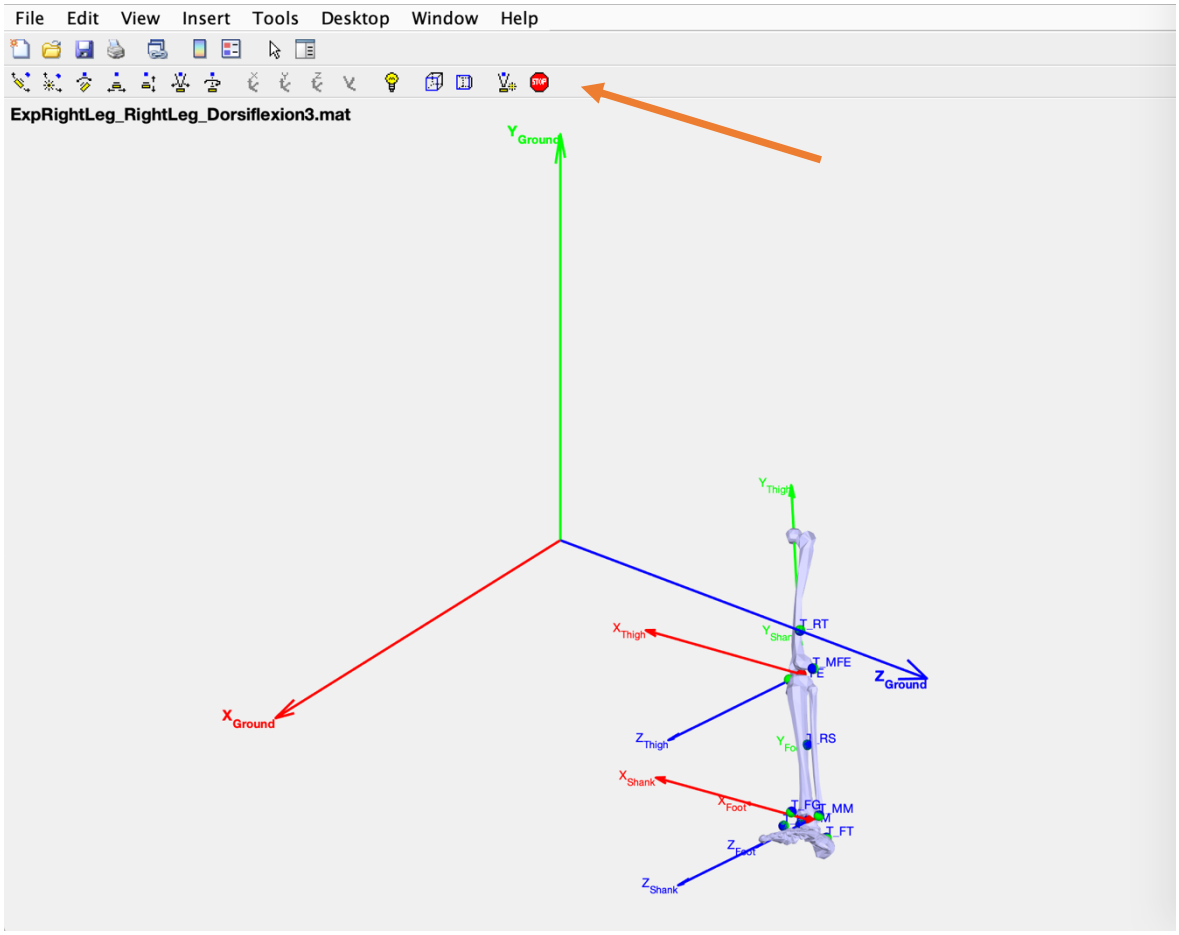


Figura 15. Ventana 3D de la aplicación

En la esquina superior izquierda de la imagen de arriba se muestra el nombre del fichero de resultados que se está reproduciendo.

En la parte superior de la ventana, señalada con una flecha naranja, se puede ver una barra de herramientas. Aquí se encuentran todos los controles necesarios para manipular la escena gráfica. Esta barra de herramientas es la *CameraToolbar* de Matlab y se trata de una funcionalidad propia del software ya programada internamente [6].

5.2.1. Representación de objetos gráficos

El modelo 3D que se muestra en la ventana de visualización se conforma de los siguientes componentes.

- Segmentos

La unión de éstos representa el cuerpo que se ha grabado en la captura de movimiento. Estos pueden ser:

  - Archivo de formato stl del hueso (Figura 16)
  - Cilindro (*Wireframe*) que simula el hueso (Figura 17)

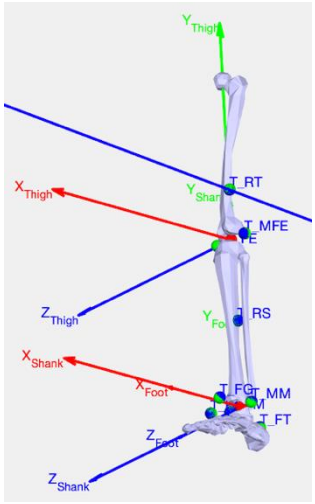


Figura 16. Archivo de formato stl del hueso

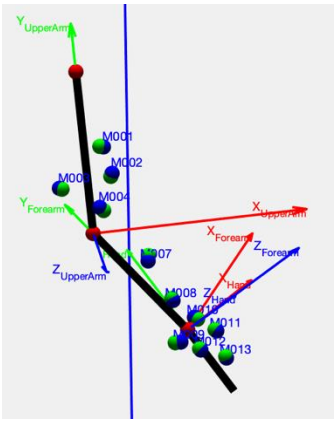


Figura 17. Segmento Wireframe

- Marcadores modelo: se representan mediante esferas azules.

- Marcadores experimentales: se representan mediante esferas verdes.

- Cilindros auxiliares

Se representan mediante cilindros rojos. Unen el centro de cada marcador modelo (azul) con su correspondiente marcador experimental (verde). Por lo tanto, indican los errores de medición del sistema de captura de movimiento empleado. Si la medición es suficientemente buena, los cilindros auxiliares no se ven.

- Articulaciones: se representan mediante esferas rojas. Se encuentran en la unión entre dos segmentos diferentes.

- Sistemas de coordenadas y sus respectivos nombres

Por una parte, el sistema de coordenadas global, el cual pertenece al segmento *Ground* y es fijo. Por otra parte, cada segmento tiene un sistema de coordenadas local, en cuyos ejes se indica el nombre de dicho segmento como subíndice y se desplaza solidariamente con él. Por convenio, el eje x tiene color rojo, el eje y verde y el eje z azul. En la Figura 15 se puede observar esta representación.

6. PROGRAMACIÓN DE LA APLICACIÓN

La aplicación se ha programado en el App Designer de Matlab (versión 2020a).

6.1. Funcionamiento y lógica de la aplicación

La estructura de programación de la aplicación consta de tres partes. La primera se ejecuta al iniciar la aplicación. La segunda se ejecuta cuando se carga un modelo 3D y consiste en inicializar todos los parámetros y variables necesarios. La tercera es el bucle de animación, y comienza cuando el usuario lo decida, una vez está cargado el modelo que se desea visualizar.

Por otra parte, la aplicación tiene dos tipos de funciones: funciones normales y *callbacks*. Las primeras son funciones a las que se les llamas a medida que se ejecuta el código. Las segundas son llamadas cuando el usuario interactúa con alguno de los botones o controles de la aplicación.

En lo que a las variables se refiere, muchas de ellas se han declarado como globales, ya que al existir multitud de funciones y *callbacks* se facilita así el acceso a todas ellas.

A lo largo del código se han insertado comentarios, los cuales aparecen en color verde, que ayudan a aclarar el funcionamiento de cada línea de código.

6.1.1. Inicialización de la aplicación

Esta sección del código se ejecuta automáticamente cuando el usuario abre la aplicación, antes de cargar cualquier modelo. En la Figura 18 mostrada abajo, se puede observar esta parte del código, que consiste en una función llamada *startupFcn*.

Como se puede apreciar en la imagen, se crean dos variables de tipo global: *closenointeraction* y *slide*. Estas dos variables posibilitan el correcto funcionamiento de la aplicación. La primera de ellas permite cerrar la aplicación antes de que el usuario cargue algún modelo, en caso de que se haya abierto por error. La segunda, permite controlar la interacción con la barra slider para que esta no dé errores cuando se posiciona en el último o primer fotograma.

Por otra parte, se ajusta la posición de la ventana del panel de control, de tal manera que ésta aparezca en el lado izquierdo de la pantalla del ordenador. Aquí se tiene en cuenta el sistema operativo en el que se está ejecutando el programa (Windows o MacOS) y el tamaño del monitor del ordenador.

```
% Code that executes after component creation
function startupFcn(app)
    %Variable that enables closing the app by pressing the CLOSE
    %APP button before an animation has been loaded
    global closenointeraction
    closenointeraction=1;
    %Variable to control slider interaction
    global slide
    slide=0;
    %Set UIFigure in the correct position depending on OS
    UIFigureSize=get(app.UIFigure,'Position');
    screensize=get(groot,'ScreenSize');
    if ispc
        windowstitlebar=28;
        windowstaskbar=40;
        %set(app.UIFigure,'Position',[1 screensize(1,4)-UIFigureSize(1,4)-screensize(1,4)*(7/270) UIFigureSize(1,3) UIFigureSize(1,4)])
        if screensize(1,4)-UIFigureSize(1,4)-windowstitlebar<windowstaskbar
            value2=windowstaskbar;
        else
            value2=screensize(1,4)-UIFigureSize(1,4)-windowstitlebar;
        end
        if UIFigureSize(1,4)>screensize(1,4)-windowstitlebar-windowstaskbar
            value4=screensize(1,4)-windowstitlebar-windowstaskbar;
        else
            value4=UIFigureSize(1,4);
        end
        set(app.UIFigure,'Position',[1 value2 UIFigureSize(1,3) value4]);
    elseif ismac
        mactitlebar=28;
        mactaskbar=40;
        if screensize(1,4)-UIFigureSize(1,4)-mactitlebar<mactaskbar
            value2=mactaskbar;
        else
            value2=screensize(1,4)-UIFigureSize(1,4)-mactitlebar;
        end
        if UIFigureSize(1,4)>screensize(1,4)-mactitlebar-mactaskbar
            value4=screensize(1,4)-mactitlebar-mactaskbar;
        else
            value4=UIFigureSize(1,4);
        end
        set(app.UIFigure,'Position',[1 value2 UIFigureSize(1,3) value4]);
    end
end
```

Figura 18. startupFcn

6.1.2. Cargar el modelo e inicialización de los parámetros necesarios

Esta sección del código se ejecuta cuando el usuario pulsa el botón *LOAD ANIMATION* del panel de control. Después de cargar todas las variables y parámetros del modelo 3D seleccionado, se muestra el primer fotograma, pero sin comenzar la animación.

Una vez el usuario pulsa el botón, se ejecuta el *callback* asociado a éste. Al principio de la función se llevan a cabo las siguientes operaciones:

- Declarar las variables globales necesarias para el correcto funcionamiento del software
- Cerrar la ventana de visualización 3D en caso de que haya una abierta
- Desactivar los *Warning*
- Resetear todos los valores del panel de control
- Declarar las variables de usuario (las cuales se explican en esta memoria más adelante)

Después, se llama a la función interna de Matlab *uigetfile*, como se puede ver en la Figura 19, la cual abre una ventana emergente que permite al usuario elegir el archivo de resultados deseado. Se guardan el nombre del archivo y su ruta.

```
%Get path and name of the .mat file
[name_matfile,partialpath_matfile] = uigetfile;

fullpath_matfile = fullfile(partialpath_matfile,name_matfile);
```

Figura 19. Llamada a la función uigetfile

Cuando ya se ha completado esto, se crea la ventana de visualización 3D en la que aparece el nombre de dicho archivo, se ajusta su tamaño según las características de la pantalla del ordenador y se sitúa a la derecha de ésta (ya que el panel de control aparece a la izquierda). Acto seguido, tanto esta ventana como el panel de control se traen al frente y se muestran ambas al usuario.

En este punto se leen los datos del archivo de resultados de formato .mat. En la Figura 20 se puede ver este paso.

```
%Load the .MAT file
global T_Segments
global T_ModelMarkers
global T_MeasuredMarkers
global Time

Matfile = load(fullpath_matfile);
T_Segments = Matfile.T_Segments;
T_ModelMarkers = Matfile.T_ModelMarkers;
T_MeasuredMarkers = Matfile.T_MeasuredMarkers;
Time = Matfile.Time;
R = Matfile.R;
GraphicsPath = Matfile.GraphicsPath;
ModelMarkers=Matfile.ModelMarkers;
ModelMarkersLocalCoords=ModelMarkers{1,1};
MeasuredMarkers=Matfile.MeasuredMarkers;
MeasuredMarkersLocalCoords=MeasuredMarkers{1,1};
WireframeLocalCoords=Matfile.WireframeLocalCoords;
SegmentNames=Matfile.Segments;
AuxWireData=Matfile.AuxWire;
Joints=Matfile.Joints;
maxd=Matfile.maxd;
```

Figura 20. Lectura del archivo de resultados de formato .mat

En la Figura 21 se puede observar cómo se ajustan algunos de los componentes que aparecen en el panel de usuario. Se indican los nombres de los segmentos en las listas pertinentes y en el menú desplegable. Se calculan el número de segmentos y de fotogramas que hay. También se establecen los límites de la barra slider y los campos de fotogramas.

```
%Print Segment Names on the List Boxes
app.ShowHideSegmentLCSListBox.Items=SegmentNames;
app.ShowHideSegmentsListBox.Items=SegmentNames;
app.SetSceneRotationOriginDropDown.Items=SegmentNames;

global nSegments
[nSegments,nSamples] = size(T_Segments);

%Set limits of Slider and Frame Field
app.FrameSlider.Limits = [0,nSamples-1];
app.FrameEditField.Limits = [0,nSamples-1];
app.DraweveryframesEditField.Limits = [1,nSamples-1];
```

Figura 21. Ajuste de parámetros del panel de control

A continuación, se procede a dibujar los siguientes objetos gráficos. Para dibujar cada uno de ellos se llama a una función diferente.

- Marcadores modelo (Ver Anexo A)
- Marcadores experimentales (Ver Anexo B)
- Cilindros auxiliares (Ver Anexo C)
- Segmentos y sus respectivos sistemas de coordenadas locales (Ver Anexo D)
- Articulaciones (Ver Anexo E)
- Sistema de coordenadas global (Ver Anexo F)

Una vez se han dibujado todos los objetos gráficos del modelo, se modifican los parámetros necesarios para que el punto de vista del conjunto completo mostrado al usuario sea el óptimo (Ver Anexo G).

Para calcular los límites de la ventana donde se muestra el modelo 3D se ha seguido el siguiente proceso.

1. Se ejecuta el comando de Matlab *axis image*, el cual hace que se ajusten los límites automáticamente a lo que se muestra por pantalla en ese instante (primer fotograma). Se guardan estos valores.

2. Se multiplican estos valores por los factores definidos en las variables de usuario, con el fin de aumentarlos en dicho porcentaje.

3. Se obtienen las coordenadas globales de los marcadores experimentales en todos los instantes capturados y se comparan las máximas de éstas con los límites actuales. En caso de que alguna resulte mayor (en valor absoluto) se reemplaza.

4. Se obtiene el punto del extremo del eje más largo de todos los sistemas de coordenadas locales y se comparan sus coordenadas con las actuales. En caso de resultar mayor (en valor absoluto), se reemplaza.

5. Se redondean los límites obtenidos a 2 decimales.

6. Se establecen estos valores como los límites de la ventana de visualización 3D y se muestran en sus respectivos campos en el panel de control.

Cuando todos estos pasos se han completado, la animación está cargada y lista para ser ejecutada, entonces la luz de al lado del botón cambia de rojo a verde.

El primer fotograma se muestra al usuario, y la ejecución se detiene momentáneamente porque se entra en un bucle infinito. De este solo se sale cuando el usuario lo decide, bien pulsando el botón de *Play*, avanzando de fotograma, moviendo la barra slider o cerrando la aplicación. En la Figura 22 de abajo se observa la parte del código que cumple estas funciones.

```
%Pausing code after loading the animation until user presses a
%button to start
while(1)
    pause(0.01);
    if PlayForward==1 && frame<nSamples-1
        slide=0;
        break;
    elseif PlayBackward==1 && frame>0
        slide=0;
        break;
    elseif FrameForward==1 && frame<nSamples-1
        i=1;
        frame=1;
        activation=1;
        slide=0;
        break;
    elseif FrameBackward==1 && frame>0
        activation=1;
        slide=0;
        break;
    elseif app.FrameEditField.Value~=0
        frame=app.FrameEditField.Value;
        i=frame;
        activation=1;
        slide=0;
        break;
        %Closing Code
    elseif StopAnimation==1
        slide=0;
        break;
        %End of Closing Code
    else
        PauseButtonValueChanged(app, event)
    end
end
```

Figura 22. Bucle de pausa antes de comenzar la animación

6.1.3. Bucle de animación

Esta sección del código se ejecuta cuando el usuario pulsa algún botón para comenzar la animación, una vez se ha cargado el modelo al completo. Tiene la siguiente estructura:

- Primer código de pausa de la animación
- Código de dibujo (actualización) de todos los objetos gráficos
- Segundo código de pausa de la animación
- Actualización de la variable que controla los fotogramas
- Código de pausa final

Los códigos de pausa consisten en bucles *while* infinitos en los que se ejecuta el comando *pause* en cada vuelta. La interacción del usuario con la aplicación, al pulsar algún

botón o mover la barra slider, hace que se ejecuten los *callback* de dichos controles y esto a su vez genera cambios en los valores de ciertas variables. En los bucles infinitos hay condiciones que, en caso de detectar cambios en estas variables, fuerzan la salida del bucle, continuando así con la animación (Ver Anexo H y Anexo I).

Se ha optado por insertar dos bucles de pausa, uno antes del código de dibujo de los objetos gráficos y otro después, debido a que esto optimiza el funcionamiento del programa. Dependiendo de qué línea se está ejecutando en el momento en el que el usuario pulsa el botón de *Pause*, el software podría no funcionar de manera óptima. Si el usuario pulsa el botón antes de haberse actualizado el fotograma y solo hay un bucle de pausa después del código de dibujo, la animación no se pararía hasta llegar a este bucle, por lo que se dibujaría un fotograma más antes de detenerse.

Como ya se explica en el apartado 6.4.2 más adelante, en el código de dibujo (o actualización) de los objetos gráficos simplemente se toma la matriz de transformación correspondiente ya calculada y se aplica al objeto en cuestión, actualizando de esta manera su posición. En la Figura 23 de abajo se puede ver el caso de los segmentos y sus sistemas de coordenadas.

```
%Graphic Objects Drawing Code
if nGraphFiles(j,1)>0
    T = T_Segments{j,(i+1)};    %Obtain the transformation matrix T from the .MAT file
    set(H{j,1},'Matrix',T);    % Update the object to new position using Transformation Matrix
    set(H{j,2},'Matrix',T);    % Update the object's LCS to new position using Transformation Matrix
    app.TimeTextArea.Value = num2str(Time{1,(i+1)});   %Update Time box
    app.FrameEditField.Value=frame; %Print frame number in frame box
    app.FrameSlider.Value=frame;    %Print frame number in frame slider
end
```

Figura 23. Actualización de los objetos gráficos de los segmentos

Después del segundo bucle de pausa, se encuentra una parte de código en la que se controla el valor de la variable *frame*. El bucle de animación es controlado por la variable *i*. Sin embargo, cuando el usuario desea cambiar de fotograma, mueve la barra slider o rebobina la animación, Matlab da un error si se modifica el valor de la variable que controla el bucle, en este caso *i*. Por esta razón, se ha decidido usar una variable auxiliar para solventar este problema.

En la Figura 24 de abajo, se pueden apreciar todas las diferentes situaciones en las que se modifica el valor de la variable *frame*, dependiendo de la voluntad del usuario.

La variable *slide* cambia su valor a 1 cuando el usuario utiliza la barra slider, pero ni no se ha llegado al primer o último fotograma inmediatamente cambia su valor a 0 otra vez. Así se consigue que la aplicación no dé ningún error cuando se llega al principio o final de la animación.

La variable *activation* cumple una función parecida, y sirve para decidir en qué bucle de pausa se debe parar la animación y cómo se debe aumentar el valor de la variable *frame*. Está relacionada con el cambio de fotograma mediante los botones de *Frame Backward* y *Frame Forward* y el campo editable de los fotogramas del panel de control.

```
%Frame increase/decrease depending on the situation
if activation~=2 && activation~=3 && slide==0
    if PlayForward==1
        i=i+app.DraweveryframesEditField.Value;
        frame=frame+app.DraweveryframesEditField.Value;
    elseif PlayBackward==1
        i=i-app.DraweveryframesEditField.Value;
        frame=frame-app.DraweveryframesEditField.Value;
    elseif FrameForward==1
        i=i+1;
        frame=frame+1;
    elseif FrameBackward==1
        i=i-1;
        frame=frame-1;
    end
elseif activation==3 && slide==0
    activation=0;
    if i==0
        i=i-1;
        frame=frame-1;
    elseif i==nSamples-1
        i=i+1;
        frame=frame+1;
    end
elseif slide==1 && frame==nSamples-1
    frame=frame+1;
    i=frame;
    slide=0;
elseif slide==1 && frame==0
    frame=frame-1;
    i=frame;
    slide=0;
end
```

Figura 24. Actualización de la variable frame

Por último, se llega al bucle de pausa final en el que solamente se entra cuando la animación ha llegado al último fotograma (o al primero si se está reproduciendo hacia atrás) que se puede ver en la Figura 25. Este bucle sirve para detener la animación una vez se ha llegado al final, pero permite volver a retomarla si el usuario cambia de fotograma. Esto se puede hacer mediante la barra slider, editando el campo de fotogramas del panel de control o simplemente pulsando al botón de *Play Forward* o *Play Backward* según el caso.

```
%Pausing Code after the Animation reaches first or last
%frame
%The Animation can be restarted from here by pressing a
%button, moving the slider or changing the frame box
if i==-1 || i==nSamples
    app.PauseButton.Value=true;
    PauseAnimation=1;
    PlayForward=0;
    PlayBackward=0;
    FrameForward=0;
    FrameBackward=0;
    comparevalue=app.FrameEditField.Value;
    while(1)
        %Closing Code
        if StopAnimation==1
            slide=0;
            break;
        end
        %End of Closing Code
        if PlayForward==1 && i==-1
            i=0;
            frame=0;
            slide=0;
            break;
        elseif PlayBackward==1 && i==nSamples
            i=nSamples-1;
            frame=nSamples-1;
            slide=0;
            break;
        elseif FrameForward==1 && i==-1
            i=1;
            frame=1;
            activation=1;
            slide=0;
            break;
        elseif FrameBackward==1 && i==nSamples
            i=nSamples-2;
            frame=nSamples-2;
            activation=1;
            slide=0;
            break;
        elseif app.FrameEditField.Value~=comparevalue
            frame=app.FrameEditField.Value;
            i=frame;
            activation=2;
            slide=0;
            break;
        else
            PauseButtonValueChanged(app, event)
        end
        pause(0.01);
    end
elseif i<=-1
    i=0;
    frame=0;
    activation=3;
elseif i>=nSamples
    i=nSamples-1;
    frame=nSamples-1;
    activation=3;
end
```

Figura 25. Bucle de pausa final

6.1.4. Callback de los botones del panel de control

Cada componente del panel de control tiene asociado un *callback*, es decir, una función a la que se llama únicamente cuando el usuario interactúa con dicho componente.

La función principal es la asociada al botón *LOAD ANIMATION* del panel de control, y es donde se ejecuta la parte principal del código.

Los demás componentes tienen asociadas funciones relativamente sencillas que cambian el valor (Ver Anexo J) o propiedades de ciertas variables (Ver Anexo K), las cuales afectan al modo de ejecución de la animación.

Sin embargo, la barra slider sí tiene asociada una función algo más compleja que consta de un código de dibujo (actualización) de los objetos gráficos análogo pero independiente al de la función principal (Ver Anexo L).

6.1.5. Detección de errores durante la ejecución

Tanto la segunda parte como la tercera parte del código, mencionadas anteriormente, pertenecen al *callback* del botón *LOAD ANIMATION*. En esta función se ha utilizado la funcionalidad de Matlab *try catch* que permite detectar errores y ejecutar ciertas líneas de código en caso de encontrar uno. Si se detecta algún error en el código, se ejecutan las líneas mostradas en la Figura 26 de abajo.

Como se puede observar, lo primero que se hace es cerrar la ventana de visualización 3D. Después, si el archivo de resultados no se ha seleccionado, se muestra un mensaje indicándolo. Lo mismo ocurre si éste no tiene el formato correcto. Si se detecta cualquier otro error, se muestra por pantalla un mensaje de error y se pide al usuario que vuelva a seleccionar un archivo de resultados. Por último, la luz de panel de control se vuelve roja y se eliminan todas las variables creadas hasta el momento, excepto *closenointeraction* y *slide* que se reinicializan.

Además, también detecta si el usuario cierra alguna de las ventanas utilizando el botón rojo de la X de la *titlebar* e impide que el código dé algún error.

```
catch ME
    if isvalid(app)==0
        if (exist('appfigure','var'))~=0
            if isvalid(appfigure)==1
                close(appfigure);
            end
        end
        return;
    else
        if (exist('appfigure','var'))~=0
            if isvalid(appfigure)==1
                close(appfigure);
            end
        end
        if isa(name_matfile,'double')==1
            if name_matfile==0 || partialpath_matfile==0
                msgbox('No file selected. Please select one.','Error');
            end
        elseif isa(name_matfile,'char')==1
            [FileName , Extension] = getFilenameAndExt(name_matfile);
            if strcmpi(Extension,'mat')==0
                msgbox('File extension is not correct. Please select another file.','Error');
            else
                msgbox('An error ocurred. Please load a model again.','Error');
            end
        else
            msgbox('An error ocurred. Please load a model again.','Error');
        end
        app.Lamp.Color='r'; %Turns lamp red which indicates that the animation is NOT ready to be played
        clearvars -except closenointeraction slide appfigure
        closenointeraction=1;
        slide=0;
        return;
    end
```

Figura 26. Errores en la ejecución de la aplicación

6.2. Variables de usuario del código

En el código existen ciertos parámetros que no es posible obtener de la reconstrucción del movimiento capturado, pero que sí que son necesarios para la visualización de los resultados. Para evitar que el código contenga valores numéricos fijos, se han definido unas variables con el valor de estos parámetros. En la Figura 27 mostrada abajo se puede apreciar cuáles son éstas.

```
% ========================================================================
% Declare user variables ================================================
extra1=1.5;  %Extra factor to multiply to the upper axis limit
extra2=0.5; %Extra factor to multiply to the lower axis limit
GCSlength=1;    %Length of the GCS
LCSlength=0.6;  %Length of the LCS
xoffset=0;  %Offset in x coordinate of the marker names
yoffset=2;    %Offset in y coordinate of the marker names
zoffset=0;  %Offset in z coordinate of the marker names
WireFacets=20;  %Number of Facets of the Wire Cylinders Draw Sequence
% Declare user variables ================================================
% ========================================================================
```

Figura 27. Variables de usuario

Estas variables se definen al principio del código. Son editables por un usuario con conocimiento para programar, ya que la aplicación no las muestra al ejecutarse, pero sí es

posible modificarlas si se decide reprogramar la aplicación. Su modificación afecta a la manera en que se muestra el modelo 3D por pantalla.

6.3. Funciones adaptadas

6.3.1. Stlread

Esta función ha sido tomada del foro File Exchange de Mathworks [7] y ha sido programada por Doron Harlev. Permite leer archivos stl de manera que Matlab los entienda y pueda representarlos (Ver Anexo M).

```
object = stlread(fullpath_stlfile);
```

Figura 28. Llamada a la función stlread

Se llama a la función como se indica en la Figura 28 de arriba. Se introduce como argumento de entrada el nombre del archivo de formato stl con su ruta (path) completa. La función devuelve una estructura (Figura 29) con dos campos: vertices y faces. El primero de ellos guarda las coordenadas de cada punto, y el segundo agrupa estos puntos de tres en tres de modo que se forman triángulos, es decir las caras que conforman el objeto gráfico poligonal.

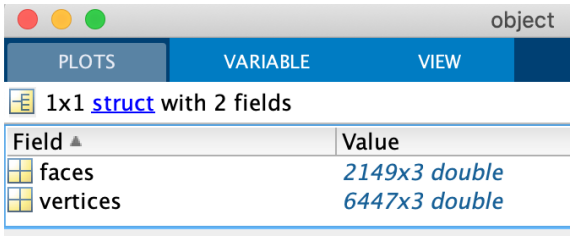


Figura 29. Stl struct

Una vez leído el archivo, resulta sencillo representarlo utilizando la función patch de Matlab [8]. Esta función permite representar regiones poligonales. Acepta como argumento de entrada una estructura con los campos vertices y faces, que es exactamente lo que se ha obtenido con la función stlread.

6.3.2. drawsphere

La función drawsphere se utiliza para representar los marcadores (modelo y experimentales) y las articulaciones.

Está basada en la función sphere, propia de Matlab [9]. Esta función, si no se le introduce ningún argumento de entrada, crea las coordenadas de una esfera de radio unidad y centrada en el origen, pero sin dibujarla.

Para adaptarla a las necesidades de la aplicación, se ha creado una nueva función en la que se introducen como argumentos de entrada el radio y las coordenadas del centro

de la esfera. Ésta llama internamente a la función sphere, después escala la esfera creada y la traslada (Ver Anexo N). Se llama a la función como se indica en la Figura 30.

```
%Marker sphere points are calculated
[X,Y,Z]=drawsphere(r,x0,y0,z0);
```

Figura 30. Llamada a la función drawsphere

Una vez obtenidas las coordenadas de la esfera se llama a la función surface [10] de Matlab que la dibuja. En la llamada a la función surface se especifica el color de la misma según el objeto que se quiera dibujar.

6.3.3. cylinder2P

La función cylinder2P ha sido diseñada por Per Sundqvist y se ha tomado del foro File Exchange de Mathworks [11]. Se introducen como argumentos de entrada el radio, el número de caras del polígono y los dos puntos entre los que se quiere dibujar el cilindro.

Matlab no dibuja un cilindro con superficie curva, sino que dibuja un polígono de n caras. Cuantas más caras tenga el polígono, más curva parecerá su superficie a simple vista.

En la función original hay que introducir las coordenadas de los dos puntos como vectores de 1x3. Sin embargo, el software MotRec los calcula como vectores 3x1. Esto inducía un error al normalizar los vectores. Por esta razón se ha añadido una pequeña modificación en el código de la función, que se muestra en la Figura 31.

```
%Transpose vectors
r1=r1';
r2=r2';
```

Figura 31. Modificación añadida en la función cylinder2P

Transponiendo los vectores, se consigue solventar el error y los cilindros se dibujan correctamente. En la Figura 32 se muestra cómo se llama a esta función.

```
[X, Y, Z]=cylinder2P(WireRadius,WireFacets,r1,r2);
```

Figura 32. Llamada a la función cylinder2P

6.3.4. LCSplot

La función LCSplot se ha basado en la función drawLCS ya existente en la librería del software MotRec. Sin embargo, ha sufrido varias modificaciones y poco tiene que ver con la original.

La nueva función toma como argumentos de entrada los ejes globales en los que se desea dibujar el sistema de coordenadas del segmento, la longitud de los ejes del

sistema de coordenadas a dibujar y el nombre del segmento (Ver Anexo O). El resultado obtenido es el que se puede ver en la Figura 33.

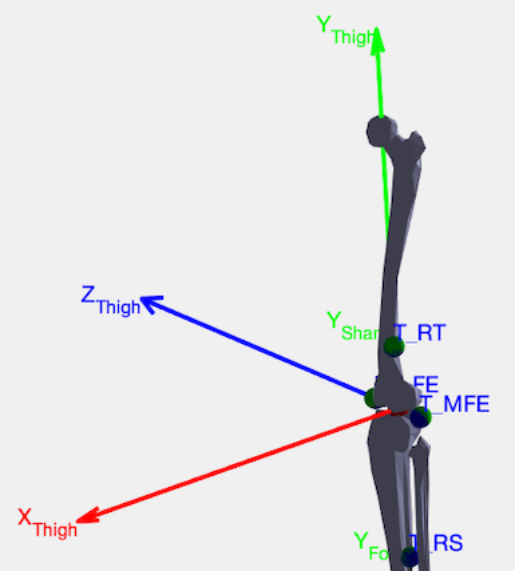


Figura 33. Sistema de coordenadas local del segmento Thigh

En la Figura 34 de abajo se puede observar cómo se llama a la función. En este caso *appax* es el nombre de los ejes en los que se dibujan tanto el segmento como su sistema de coordenadas, *LCSlength* es la longitud de los ejes del sistema de coordenadas y la variable *H{i,3}* contiene el nombre de dicho segmento. Los ceros se pueden ignorar.

```
LCSystems{i,1} = LCSplot(appax,LCSlength,0,0,H{i,3});
```

Figura 34. Llamada a la función LCSplot

6.3.5. writeFileMatrices

Esta función corresponde al código de reconstrucción de movimiento MotRec. En ella se calculan las matrices de transformación de los segmentos y marcadores para cada instante capturado. Se han añadido ciertas líneas de código en ella que permiten salvar información relevante, que es calculada a medida que se ejecuta la función, en un archivo de formato .mat. Este archivo es posteriormente utilizado para la visualización de los resultados en la aplicación desarrollada.

En la Figura 35 se pueden ver las variables creadas para guardar la información relevante calculada en la función. En cada iteración del bucle se van guardando los parámetros necesarios en estas variables, y cuando el bucle termina estas variables se salvan en un fichero .mat como se indica en la Figura 36.

```
% Initialize variables to save the T matrixes <=== Ignacio
T_Segments = cell(nSegments,nSamples);
T_ModelMarkers = cell(nMarkers,nSamples);
T_MeasuredMarkers = cell(nMarkers,nSamples);
Time = cell(1,nSamples);
maxCoord=[0 0;0 0;0 0]; %Variable to save Maximum and Minimum Coordinate to set it as the axis limit <===== Ignacio
first=1; %Variable to save first d vector in the maxCord vector, then we compare it with the next one <===== Ignacio
maxd=zeros(nSegments,1); %Variable to save max d vector coordinate from the T_Segments transformation matrix <=== Ignacio
```

Figura 35. Modificaciones en la función writeFileMatrices

```
% save data in .MAT 3D view in Matlab <===== Ignacio
save([Path,Filename],'T_Segments','T_ModelMarkers','T_MeasuredMarkers','Time','maxCoord','maxd','-append');
```

Figura 36. Salvado de la información en la función writeFileMatrices

Al lado de cada línea de ha insertado una breve explicación de cuál es su función seguida del siguiente texto "`<===== Ignacio`". Esto permite identificar fácilmente qué parte del código ha sido añadida a la versión original de la función.

6.3.6. writeFilePlayback

En esta función se han añadido modificaciones con un objetivo análogo a las realizadas en la función *writeFileMatrices*, mencionada en el punto anterior. La función *writeFilePlayback* calcula los parámetros necesarios para la correcta representación de los objetos gráficos como por ejemplo el color, las coordenadas locales, el radio de esferas y cilindros y el nombre de éstos.

Esta información se salva del mismo modo en el fichero en el que se guarda la información extraída de la función *writeFileMatrices*.

Al igual que en la función mencionada anteriormente, al lado de cada línea de ha insertado una breve explicación de cuál es su función seguida del siguiente texto "`<===== Ignacio`". Esto permite identificar fácilmente qué parte del código ha sido añadida a la versión original de la función.

6.4. Tratamiento de los archivos de formato stl

6.4.1. Conversión de los archivos de formato stl

El software MotRec ya disponía de archivos .x legibles por el anterior programa, Compamm. Mediante un conversor online de archivos [12] se han transformado estos archivos, obteniendo así una réplica en formato stl.

No obstante, este conversor altera los ejes del objeto gráfico, y por lo tanto las coordenadas también, dando como resultado un archivo simétrico al original. Para solucionar este problema, se han convertido todos los archivos y después se les ha cambiado el nombre por su simétrico. Es decir, el modelo .x de la pierna derecha ha pasado a ser el modelo .stl de la pierna izquierda. Se ha realizado un proceso análogo con el resto de archivos de formato .x.

6.4.2. Adecuación de los archivos de formato stl

Los modelos de los archivos stl tienen unas coordenadas y ejes locales por defecto. En los archivos de formato xml del modelo se definen los parámetros de translación, rotación y escalado necesarios para mover el origen del sistema de coordenadas local del archivo stl al punto deseado, como se indica en la Figura 37.

```
<Segment Name="Foot" >
  <Point Name="AJC" LocCoord="[0;0;0]"/>
  <Vector Name="XF" LocCoord="[1;0;0]" />
  <Vector Name="ZF" LocCoord="[0;0;1]" />
  <Vector Name="YF" LocCoord="[0;1;0]" />
  <Marker Name="T_FG" LocCoord="[0.09357;-0.00658;-0.03013]"/>
  <Marker Name="T_FT" LocCoord="[-0.06311;-0.03773;0.0054]"/>
  <Marker Name="T_FM" LocCoord="[0.0573;-0.01388;0.04684]"/>
  <Graphic File="talus-foot-toesL.stl" Tras="[0;-0.01;0]" Rot="[0;0;0]" Scal="[0.18/0.14;0.18/0.14;0.18/0.14]" />
```

Figura 37. Parámetros de translación, rotación y escalado de los archivos de formato stl

El software se ha programado de tal manera que aplica estos valores a los puntos del archivo, modificando así las coordenadas del mismo permanentemente, como se indica en la Figura 38. Una vez se ha completado este paso, con los ejes del sistema de coordenadas local situados correctamente, se procede a reproducir la animación.

```
for k=1:size(object.vertices,1)
    object.vertices(k,:)=object.vertices(k,:)+Trans;
    object.vertices(k,:)=object.vertices(k,:)*Rot;
    object.vertices(k,:)=object.vertices(k,:)*Scal;
end
```

Figura 38. Modificación de las coordenadas del archivo de formato stl

A partir de este punto, se utilizada un objeto de Matlab llamado hgtransform [13]. Este método permite realizar una animación de manera más fluida y rápida. En vez de modificar las coordenadas de cada punto del objeto gráfico una a una como se ha realizado para aplicar la translación, rotación y escalado, simplemente se actualiza el campo de *Matriz de Transformación* (que inicialmente es la matriz identidad) de cada segmento en cada fotograma. Las matrices de transformación de cada instante capturado se precalculan en la función *writeFileMatrices* (explicada en el punto 6.3.5) del software MotRec, y se guardan en las variables *T_Segments*, *T_ModelMarkers* y *T_MeasuredMarkers* que se encuentran en el fichero de resultados de formato .mat (Figura 39).

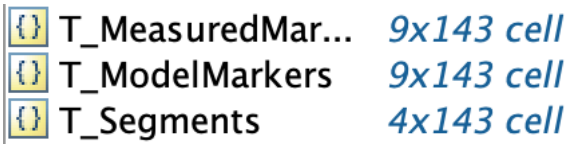


Figura 39. Variables que contienen las matrices de transformación

6.5. Funcionamiento en sistemas operativos Windows y Mac

Uno de los grandes logros de este proyecto ha sido conseguir que la nueva aplicación funcione tanto en los sistemas operativos de Windows como en los de Mac. Además, también se ha añadido una pequeña modificación en el código del software de reconstrucción de movimiento MotRec de tal manera que funcione en MacOS.

A continuación, se explican las tres diferencias fundamentales para el correcto funcionamiento en los sistemas operativos Windows y MacOs.

1. Barras para marcar las rutas (*path*)

Los sistemas operativos Windows utilizan la barra invertida \ para marcar la ruta. Por su parte, MacOS utiliza la barra normal /. Esto supone un problema a la hora de leer rutas, nombres de archivos o buscar funciones.

Para eliminar el problema, se han creado dos variables globales en la función startupBiomech, que debe ser ejecutada nada más iniciar la sesión en Matlab, llamadas PathBar y PathBar2, como se indica en la Figura 40. Por otra parte, se han sustituido todas las barras que aparecían en el código de Matlab por estas variables.

```
global PathBar PathBar2
if ismac
    % Code to run on Mac platform
    PathBar  = '/';
    PathBar2 = '/';
elseif ispc
    % Code to run on Windows platform
    PathBar  = '\';
    PathBar2 = '\\';
end
```

Figura 40. Variables globales PathBar y PathBar2

2. Archivos .C3D y .MAT

En la versión anterior del software, los patrones de movimiento se definían en archivos de formato .C3D. Para leer este tipo de archivos es necesario instalar el software C3DServer, el cual no es compatible con el sistema operativo de Mac.

Para la nueva versión, se han convertido todos los archivos .C3D a archivos .mat, que son legibles por Matlab, mediante la función C3D2MAT (Ver Anexo P). Sin embargo, esta conversión hay que hacerla desde un ordenador con sistema operativo Windows, ya que a su vez ésta llama al lector C3D, el cual solo es compatible con este sistema. Es decir, los archivos se deben convertir en el sistema operativo Windows y después ya pueden ser utilizados en ambos sistemas operativos.

En el archivo main de formato xml de cada modelo, es necesario introducir este cambio cuando se especifica el archivo con los patrones de movimiento, como señala la flecha naranja en la Figura 41 de abajo.

```
<Experiment>
  <!-- ExpPath. Path for the experimental data. Options:
       1) full path e.g. C:\Users\MotRec\Experiments\ExpPend\
       2) a path relative to this file  e.g. .\Experiments\ExpPend\ -->
  <ExpPath>./Experiments/ExpRightLeg/</ExpPath>
  <!-- GuidedVarsFile. File that defines the model guided variables. Must
  <GuidedVarsFile>RightLeg_guidedVars.m</GuidedVarsFile>
  <Motion>Dorsiflexion3.mat</Motion>
  <Motion>Extension2.mat</Motion>
  <Motion>Inversion3.mat</Motion>
  <Motion>Static1.mat</Motion>
```

Figura 41. Archivo main de formato xml

3. Archivos .xml main

El último cambio significativo ha sido duplicar los archivos main de formato xml. Se han creado dos versiones de cada archivo con los nombres NombreDelArchivo_Win.xml y NombreDelArchivo_MAC.xml.

La diferencia entre las dos versiones radica en las barras para marcar las rutas de los archivos que deben leerse. En la versión de Windows se utiliza la barra invertida \ y en la versión de Mac la barra normal /.

El usuario debe tener en cuenta este punto, ya que a la hora de ejecutar el software MotRec para reconstruir el movimiento, debe seleccionar la versión apropiada dependiendo del sistema operativo en el que se esté trabajando.

Se han adecuado los siguientes modelos para que sean ejecutables tanto en el software MotRec como en la aplicación desarrollada en los sistemas operativos Windows y MacOS:

- Pendulum
- DoublePendulum
- Forearm
- RightArm
- RightLeg
- LLegRLeg

6.6. Colores disponibles en el software

Como se ha establecido anteriormente, los objetos gráficos de los segmentos pueden ser bien archivos stl o cilindros (*wireframe*) que simulen el hueso. En el primero de los casos, el color del modelo está predeterminado. En el segundo en cambio, el usuario puede definir el color que desee.

En el archivo de definición del modelo de formato xml, se puede establecer el color del cilindro como se indica en la línea de código de abajo.

```
<Graphic DrawSeq="GHJC,EJC" Radius="0.008" Colour="Black"/>
```

En el archivo de formato xml se define el nombre del color. Después, la función *getColor* lo lee y devuelve su código RGB legible por Matlab (Ver Anexo Q).

A continuación, se presentan todos los colores que el nuevo software es capaz de interpretar (Tabla 1).

NAME	RGB/SHORTCUT (MATLAB)	COLOR
Black	k	
Dark Red	0.5 0 0	
Red	r	
Pink	m	
Rose	1 0.6 0.8	
Brown	0.6 0.2 0	
Orange	1 0.4 0	
Light Orange	1 0.6 0	
Gold	1 0.8 0	
Tan	1 0.8 0.6	
Olive Green	0.2 0.2 0	
Dark Yellow	0.5 0.5 0	
Lime	0.6 0.8 0	
Yellow	y	
Light Yellow	1 1 0.6	
Dark Green	0 0.2 0	
Green	0 0.5 0	
Sea Green	0.2 0.6 0.4	
Bright Green	g	
Light Green	0.8 1 0.8	
Dark Teal	0 0.2 0.4	
Teal	0 0.5 0.5	
Aqua	0.2 0.8 0.8	
Turquoise	c	
Light Turquoise	0.8 1 1	
Dark Blue	0 0 0.5	
Blue	b	
Light Blue	0.2 0.4 1	
Sky Blue	0 0.8 1	
Pale Blue	0.6 0.8 1	
Indigo	0.2 0.2 0.6	

Blue-Gray	0.4 0.4 0.6	
Violet	0.5 0 0.5	
Plum	0.6 0.2 0.4	
Lavender	0.8 0.6 1	
Gray-80%	0.2 0.2 0.2	
Gray-50%	0.5 0.5 0.5	
Gray-40%	0.588 0.588 0.588	
Gray-25%	0.753 0.753 0.753	
White	w	

Tabla 1. Colores disponibles en la aplicación

7. CONCLUSIONES

Tras la realización del proyecto y teniendo en cuenta los objetivos marcados al inicio del mismo, es posible extraer las siguientes conclusiones.

- El software desarrollado muestra correctamente el movimiento capturado y permite la interacción con el usuario. Esto posibilita la sustitución del software empleado anteriormente (Compamm) por el nuevo, trabajando así siempre en el entorno Matlab. Así se evitan posibles problemas de incompatibilidad por antigüedad del software de visualización 3D con futuras versiones de Windows y OpenGL.

- La aplicación funciona correctamente en la versión 2020a de Matlab, lo cual abre la posibilidad de ejecutarla en cualquier ordenador de la escuela, ya que todos tienen instalado esta versión durante el curso 2020-2021. También puede ser ejecutada en los ordenadores personales de los alumnos y/o profesores si estos han descargado el software, cuya licencia sí tienen disponible por pertenecer a la Escuela.

- Tanto los sistemas operativos de Windows como los de Mac son capaces de ejecutar el software desarrollado. Esto es importante ya que elimina cualquier restricción para que los alumnos trabajen con él en sus ordenadores personales, al contrario de lo que ocurría con Compamm, que solo estaba disponible para Windows.

- Los controles para la manipulación de la aplicación por el usuario son sencillos y fáciles, como se explica en el manual del software MotRec (Ver Anexo R). Esto acelera el proceso de dominio del software por parte del usuario y lo hace más práctico.

- A diferencia de Compamm, que leía archivos de objetos gráficos de formato .x, el nuevo software es capaz de leer y reproducir archivos de formato stl, los cuales son notablemente más comunes y compatibles con otros programas. Esto aumenta en gran medida la versatilidad del software.

- La aplicación muestra al usuario el fotograma y tiempo actual de la animación, lo cual puede ayudar a visualizar fotogramas específicos de más interés y facilita la detección del momento concreto de una anomalía en el patrón de movimiento, si la hubiera.

8. PRESUPUESTO

A continuación, se detalla el presupuesto correspondiente al desarrollo del proyecto. Se encuentra dividido en varias partidas.

- Inmovilizado material: Se detallan todos componentes fabricados o comprados a terceros que se han utilizado durante la realización del proyecto.
- Material fungible (consumible): Se detallan todos los fungibles consumidos durante la realización del proyecto.
- Equipamiento: Se incluye todo el gasto por uso de máquinas. Para calcular estos gastos se tiene en cuenta el coste medio de amortización y el tiempo de uso de dicho equipamiento durante el proyecto.
- Software: En él, se incluye todo el software utilizado y necesario para el desarrollo del proyecto. Se tiene en cuenta el coste de amortización de las licencias.
- Mano de obra: Esta partida corresponde a los costes necesarios de los recursos humanos involucrados en cada una de las fases del proyecto.

Presupuesto de inmovilizado

Cantidad	Referencia	Descripción	Precio (€)	
			Unitario	Total
-	-	-	-	-
Total inmovilizado				0

Tabla 2. Presupuesto inmovilizado material

Presupuesto de material fungible

Cantidad	Referencia	Descripción	Precio (€)	
			Unitario	Total
3	8373602	Bolígrafos	0.35	1.05
50	-	Folios Din A4	0.01	0.5
Total fungibles				1.55

Tabla 3. Presupuesto fungibles

Presupuesto de equipamiento

Equipamiento	Cuota adquisición (€)	Tiempo amortización (años)	Cuota amortización mensual (€)	Tiempo de uso (mes)	Amortización (€)
Ordenador Ceit	800	4	17	5	85
Ordenador personal	1.200	10	10	5	50
Total equipamiento					135

Tabla 4. Presupuesto equipamiento

Presupuesto de software

Software	Cuota adquisición (€)	Tiempo amortización (años)	Cuota amortización mensual (€)	Tiempo de uso (mes)	Amortización (€)
Matlab 2020a	69	1	5.75	5	28.75
Windows 10	59.99	1	5	5	25
MacOS Catalina	(incluido en el precio del ordenador personal)	-	-	-	-
Total software					53.75

Tabla 5. Presupuesto equipamiento

Presupuesto de mano de obra

Tarea	Duración (horas)	Precio (€)	
		Unitario	Total
Diseño de la aplicación	230	-	-

Redactar la memoria y el manual de usuario	70	-	-
Total mano obra	300 horas		

Tabla 6. Presupuesto Mano obra

Resumen del presupuesto

Se presenta a continuación el presupuesto global del proyecto, el cual asciende a doscientos cincuenta y tres con 29 euros y 300 horas de mano de obra no remuneradas.

Partida	Importe (€)	
	Parcial	Acumulado
Inmovilizado	-	-
Fungibles	1.55	1.55
Equipamiento	135	136.55
Software	53.75	190.3
Mano de obra	300 horas	300 horas
Costes indirectos (10%)		19.03
Total sin IVA		209.33
Total con IVA		253.29

Tabla 7. Presupuesto total del proyecto

9. REFERENCIAS

[1] Teseo, «¿Qué es y cómo funciona la captura de movimiento?,» 02 11 2020. [En línea]. Available: https://teseo.es/noticias/que-es-y-como-funciona-la-captura-de-movimiento/.

[2] E. R. TRANSPORT, «CORDIS EU Research Results,» 25 05 2017. [En línea]. Available: https://cordis.europa.eu/project/id/218525.

[3] «SimTK,» 18 08 2020. [En línea]. Available: https://simtk.org/projects/opensim/.

[4] O. Heidari, «MathWorks,» 12 04 2017. [En línea]. Available: https://www.mathworks.com/matlabcentral/fileexchange/62513-mocap_toolbox.

[5] E. B. Javier Garcia de Jalón, Kinematic and Dynamic Simulation of Multibody Systems, NY: Springer, 1993.

[6] I. The MathWorks, «MathWorks,» 2020. [En línea]. Available: https://www.mathworks.com/help/matlab/creating_plots/view-control-with-the-camera-toolbar.html.

[7] D. Harlev, «MathWorks,» 07 01 2005. [En línea]. Available: https://www.mathworks.com/matlabcentral/fileexchange/6678-stlread.

[8] I. The MathWorks, «MathWorks,» 2020. [En línea]. Available: https://www.mathworks.com/help/matlab/ref/patch.html.

[9] I. The MathWorks, «MathWorks,» 2020. [En línea]. Available: https://www.mathworks.com/help/matlab/ref/sphere.html.

[10] I. The MathWorks, «MathWorks,» 2020. [En línea]. Available: https://www.mathworks.com/help/matlab/ref/surf.html.

[11] L. Barone, «MathWorks,» 16 10 2008. [En línea]. Available: https://www.mathworks.com/matlabcentral/fileexchange/21758-cylinder-surface-connecting-2-points?focused=5104454&tab=function&s_tid=gn_loc_drop.

[12] «MESHCONVERTER.COM,» 2020. [En línea]. Available: https://www.meshconvert.com/es.html.

[13] I. The MathWorks, «MathWorks,» 2020. [En línea]. Available: https://www.mathworks.com/help/matlab/ref/hgtransform.html.

10. ANEXOS

ANEXO A: Código de la función DrawModelMarkers

```
function
[ModelMarkersSphere,MODS,nMarkers]=DrawModelMarkers(app,appax,ModelMarkers,ModelMarkersLocalCoords,nSegments,T_S
egments,xoffset,yoffset,zoffset)
ModelMarkersSphere=cell(nSegments,1);   %Variable where Model Markers graphic objects are saved
MODS=cell(nSegments,1); %Variable that is the Parent of the Model Markers graphic objects

nMarkers=zeros(nSegments,1);

for i = 1: nSegments
    nMarkers(i,1)=size(ModelMarkersLocalCoords{i,1},1);
    if nMarkers(i,1)>0
        %First column for the Marker garphic object
        %Second column for the Marker Name
        ModelMarkersSphere{i,1}=gobjects(nMarkers(i,1),2);
        for j=1:nMarkers(i,1)
            r=ModelMarkers{2,1};    %radius of the marker
            %Center of the marker sphere
            x0=ModelMarkersLocalCoords{i,1}{j,1}(1,1);
            y0=ModelMarkersLocalCoords{i,1}{j,1}(2,1);
            z0=ModelMarkersLocalCoords{i,1}{j,1}(3,1);

            %Marker sphere points are calculated
            [X,Y,Z]=drawsphere(r,x0,y0,z0);

            %Marker sphere and text are drawn
            ModelMarkersSphere{i,1}{j,1}=surface(appax,X,Y,Z,'EdgeColor','none','FaceColor', ModelMarkers{3,1});
            ModelMarkersSphere{i,1}{j,2}=text(appax,(x0+xoffset*r),(y0+yoffset*r),(z0+zoffset*r),ModelMarkersLocalCoords{i,1}{j,2},'inter
preter','none','Color',ModelMarkers{3,1});
            hold(appax, 'on');
        end
        %Transform is created and set as parent
        MODS{i,1}=hgtransform('Parent',appax);
        set(ModelMarkersSphere{i,1},'Parent',MODS{i,1});

        %First frame transformation matrix is applied
        set(MODS{i,1},'Matrix',T_Segments{i,1});
    end
end
end
```

ANEXO B: Código de la función DrawMeasuredMarkers

```
function
[MeasuredMarkersSphere,MEDS,nMeasuredMarkers]=DrawMeasuredMarkers(app,appax,MeasuredMarkers,MeasuredMarkersLoc
alCoords,T_MeasuredMarkers)
%Number of total Measured Markers
nMeasuredMarkers=size(MeasuredMarkersLocalCoords,1);

MeasuredMarkersSphere=gobjects(nMeasuredMarkers,1); %Variable where Measured Markers graphic objects are saved
MEDS=cell(nMeasuredMarkers,1);  %Variable that is the Parent of the Measured Markers graphic objects

for i = 1: nMeasuredMarkers
    r=MeasuredMarkers{2,1}; %radius of the marker
    %Center of the marker sphere
    x0=MeasuredMarkersLocalCoords{i,1}(1,1);
    y0=MeasuredMarkersLocalCoords{i,1}(2,1);
    z0=MeasuredMarkersLocalCoords{i,1}(3,1);

    %Marker sphere points are calculated
    [X,Y,Z]=drawsphere(r,x0,y0,z0);
    %Marker sphere is drawn
    MeasuredMarkersSphere(i,1)=surface(appax,X,Y,Z,'EdgeColor','none','FaceColor', MeasuredMarkers{3,1});
    hold(appax, 'on');

    %Transform is created and set as parent
    MEDS{i,1}=hgtransform('Parent',appax);
    set(MeasuredMarkersSphere(i,1),'Parent',MEDS{i,1});

    %First frame transformation matrix is applied
    set(MEDS{i,1},'Matrix',T_MeasuredMarkers{i,1});
end
end
```

ANEXO C: Código de la función DrawAuxWire

```
function
[AuxWire,ModelMarkerCenter]=DrawAuxWire(app,appax,AuxWireData,MeasuredMarkersLocalCoords,ModelMarkersLocalCoords,n
Samples,nSegments,nMeasuredMarkers,T_Segments,T_MeasuredMarkers)
AuxWire=gobjects(nMeasuredMarkers,1);
MarkerOrder=cell(nMeasuredMarkers,2);   %Variable to save which model marker must be taken
%First colum indicates de segment where the model marker is
%attached
%Second colum indicates the number of model marker of that
%particular segment
for i=1:nMeasuredMarkers
    for j=1:nSegments
        if ~isempty(ModelMarkersLocalCoords{j,1})
            for k=1:size(ModelMarkersLocalCoords{j,1},1)
                if strcmp(ModelMarkersLocalCoords{j,1}{k,2},MeasuredMarkersLocalCoords{i,2})==1
                    MarkerOrder{i,1}=j; %Save the segment where the rigid marker is attached
                    MarkerOrder{i,2}=k; %Save the number of that marker in the j segment
                    break;
                end
            end
        end
    end
end
%Aux Wires for the first frame are drawn
ModelMarkerCenter=cell(nMeasuredMarkers,nSamples);
for i=1:nMeasuredMarkers
    segment=MarkerOrder{i,1};
    numberofmarker=MarkerOrder{i,2};
    for j=1:nSamples
        ModelMarkerCenter{i,j}=ones(4,1);
        ModelMarkerCenter{i,j}(1:3,1)=ModelMarkersLocalCoords{segment,1}{numberofmarker,1};
        ModelMarkerCenter{i,j}=T_Segments{segment,j}*ModelMarkerCenter{i,j};
    end
    x=[ModelMarkerCenter{i,1}(1,1) T_MeasuredMarkers{i,1}(1,4)];
    y=[ModelMarkerCenter{i,1}(2,1) T_MeasuredMarkers{i,1}(2,4)];
    z=[ModelMarkerCenter{i,1}(3,1) T_MeasuredMarkers{i,1}(3,4)];
    hold(appax, 'on');
    AuxWire(i,1)=line(appax,x,y,z,'LineWidth',AuxWireData{1,1}*1000,'Color',AuxWireData{1,3});
    %If experimental Marker is at the origin it means it
    %has not been read correctly so no line is drawn
    if T_MeasuredMarkers{i,1}(1,4)==0 && T_MeasuredMarkers{i,1}(2,4)==0 && T_MeasuredMarkers{i,1}(3,4)==0
        set(AuxWire(i,1),'Visible','off');
    else
    end
end
end
```

ANEXO D: Código de dibujo de los segmentos y sus sistemas de coordenadas

```
% Draw the Graphic Object for each segment
for i=1:nSegments

    %     Write Segment name
    H{i,3}=SegmentNames(i,1);

    %     Check if the Graphic Object is a cylinder
    if ~isempty(R.Subject.Segments(i).WireFrame)
        DrawSeq = R.Subject.Segments(i).WireFrame.DrawSeq;
        if ~isempty(R.Subject.Segments(i).WireFrame.Color)
            WireColor = R.Subject.Segments(i).WireFrame.Color;
            WireColorrgb=getColor(app,WireColor);
        end
        if ~isempty(R.Subject.Segments(i).WireFrame.Radius)
            WireRadius = str2num(R.Subject.Segments(i).WireFrame.Radius);
        end

        nDrawSeq=size(DrawSeq,1);
        nGraphFiles(i,1)=nDrawSeq-1;
        h{i,1}=gobjects(nGraphFiles(i,1),1);
        h{i,2}=hggroup(appax);

        %         Draw Cylinders
        for j=1:nGraphFiles(i,1)
            r1=WireframeLocalCoords{i,1}{j,1};
            r2=WireframeLocalCoords{i,1}{j+1,1};

            [X, Y, Z]=cylinder2P(WireRadius,WireFacets,r1,r2);

            h{i,1}(j,1)=surface(appax,X,Y,Z,'FaceColor',WireColorrgb,'EdgeColor','none');
            hold(appax, 'on');
            %             Draw LCS
            if j==1 && strcmp(H{i,3},'Ground')~=1
                LCSystems{i,1} = LCSplot(appax,X,Y,Z,H{i,3});
                set(LCSystems{i,1},'Parent',h{i,2});
                %                 Get maximum length of all LCS
                maxtmp=max(abs((LCSystems{i,1}(1,2).Position)));
                if maxtmp>maxLCS
                    maxLCS=maxtmp;
                    longestsegment=i;
                end
            end
        end
        %         Create an hgtransform group and set it as a parent of the patch
        H{i,1}=hgtransform(appax);
        set(h{i,1},'Parent',H{i,1});
        H{i,2}=hgtransform(appax);
        set(h{i,2},'Parent',H{i,2});

        %         Convert to global coordinates
        set(H{i,1},'Matrix',T_Segments{i,1});
        set(H{i,2},'Matrix',T_Segments{i,1});
        hold(appax, 'on');
    end

    %     Check if the Graphic Object is an .stl file
    GraphFile = R.Subject.Segments(i).Graphic;
    if iscell(GraphFile)  &&  ~isempty(GraphFile)
        nGraphFiles(i,1) = size(GraphFile,1);
        h{i,1} = gobjects(nGraphFiles(i,1),1);
        h{i,2}=hggroup(appax);

        for j=1:nGraphFiles(i,1)
            %             Read the stl file
            stlfilename = GraphFile{j,1};
            fullpath_stlfile = fullfile(GraphicsPath,stlfilename);
            object = stlread(fullpath_stlfile);

            %             Translate, rotate and scale the patch
            Trans=GraphFile{1,2}';
```

```
        XRot=makehgtform('xrotate',GraphFile{1,3}(1,1));
        XRot=XRot(1:3,1:3);
        YRot=makehgtform('yrotate',GraphFile{1,3}(2,1));
        YRot=YRot(1:3,1:3);
        ZRot=makehgtform('zrotate',GraphFile{1,3}(3,1));
        ZRot=ZRot(1:3,1:3);
        Rot=XRot*YRot*ZRot;
        Scal=makehgtform('scale',GraphFile{1,4});
        Scal=Scal(1:3,1:3);

        for k=1:size(object.vertices,1)
            object.vertices(k,:)=object.vertices(k,:)+Trans;
            object.vertices(k,:)=object.vertices(k,:)*Rot;
            object.vertices(k,:)=object.vertices(k,:)*Scal;
        end

        %           Draw the stl file
        h{i,1}(j,1) = patch(appax,object,'FaceColor',       [0.8 0.8 1.0], ...
            'EdgeColor',       'none',        ...
            'AmbientStrength', 0.7);
        hold(appax, 'on');
        %           Draw LCS
        if j==1 && strcmp(H{i,3},'Ground')~=1
            X=h{i,1}(j,1).Vertices(:,1);
            Y=h{i,1}(j,1).Vertices(:,2);
            Z=h{i,1}(j,1).Vertices(:,3);
            LCSystems{i,1} = LCSplot(appax,X,Y,Z,H{i,3});
            set(LCSystems{i,1},'Parent',h{i,2});
            %               Get maximum length of all LCS
            maxtmp=max(abs((LCSystems{i,1}(1,2).Position)));
            if maxtmp>maxLCS
                maxLCS=maxtmp;
                longestsegment=i;
            end
        end
    end
    %       Create an hgtransform group and set it as a parent of the patch
    H{i,1}=hgtransform(appax);
    set(h{i,1},'Parent',H{i,1});
    H{i,2}=hgtransform(appax);
    set(h{i,2},'Parent',H{i,2});

    %       Convert to global coordinates
    set(H{i,1},'Matrix',T_Segments{i,1});
    set(H{i,2},'Matrix',T_Segments{i,1});
  end
end
% Convert maxLCS to GLobal Coordinates
maxLCS=maxLCS+maxd(longestsegment,1);
```

ANEXO E: Código de la función DrawJoints

```
function [JointsSphere,JNT,nJoints]=DrawJoints(app,appax,Joints,nSegments,T_Segments)
JointsSphere=cell(nSegments,1);   %Variable where Joints graphic objects are saved
JNT=cell(nSegments,1); %Variable that is the Parent of the Joints graphic objects

nJoints=zeros(nSegments,1);

for i = 1: nSegments
    if ~isempty(Joints{1,1}{i,1})
        %First column for the Marker garphic object
        %Second column for the Marker Name
        nJoints(i,1)=size(Joints{1,1}{i,1},1);
        JointsSphere{i,1}=gobjects(nJoints(i,1),1);
        for j=1:nJoints(i,1)
            r=Joints{2,1};    %radius of the joint
            %Center of the marker sphere
            x0=Joints{1,1}{i,1}{j,1}(1,1);
            y0=Joints{1,1}{i,1}{j,1}(2,1);
            z0=Joints{1,1}{i,1}{j,1}(3,1);

            %Joint sphere points are calculated
            [X,Y,Z]=drawsphere(r,x0,y0,z0);

            %Joint sphere and text are drawn
            JointsSphere{i,1}(j,1)=surface(appax,X,Y,Z,'EdgeColor','none','FaceColor', Joints{3,1});
            %                       JointsSphere{i,1}(j,2)=text(appax,x0,(y0+1.5*r),z0,JointName,'Color',Joints{3,1});
            hold(appax, 'on');
        end
        %Transform is created and set as parent
        JNT{i,1}=hgtransform('Parent',appax);
        set(JointsSphere{i,1},'Parent',JNT{i,1});

        %First frame transformation matrix is applied
        set(JNT{i,1},'Matrix',T_Segments{i,1});
    end
end
end
```

ANEXO F: Código de dibujo del sistema de coordenadas global

```
% Draw GCS
global GCS
GCS=hggroup(appax);
for i=1:nSegments
    if strcmp(H{i,3},'Ground')==1
        GCSystem=LCSplot(appax,GCSlength,0,0,H{i,3});
    end
end
set(GCSystem,'Parent',GCS);
```

ANEXO G: Ajuste de los parámetros que afectan al punto de vista del modelo 3D

```
%Adjust camera, light, axis
c=camlight(appax,'headlight');
material(appax,'dull');
axis(appax,'on');
axis(appax,'equal');
%Set axis limit to fit all the objects drawn
axis(appax,'image');
axislimit=zeros(3,2);
axislimit(1,1)=appax.XLim(1,1);
axislimit(1,2)=appax.XLim(1,2);
axislimit(2,1)=appax.YLim(1,1);
axislimit(2,2)=appax.YLim(1,2);
axislimit(3,1)=appax.ZLim(1,1);
axislimit(3,2)=appax.ZLim(1,2);
for i=1:3
    if axislimit(i,1)<0
        axislimit(i,1)=extra1*axislimit(i,1);
    else
        axislimit(i,1)=extra2*axislimit(i,1);
    end
    if axislimit(i,2)>0
        axislimit(i,2)=extra1*axislimit(i,2);
    else
        axislimit(i,2)=extra2*axislimit(i,2);
    end
end
%Get Measured Markers Limits
markerslimit=zeros(3,2);
for i=1:3
    if Matfile.maxCoord(i,1)<0
        markerslimit(i,1)=extra1*Matfile.maxCoord(i,1);
    else
        markerslimit(i,1)=extra2*Matfile.maxCoord(i,1);
    end
    if Matfile.maxCoord(i,2)>0
        markerslimit(i,2)=extra1*Matfile.maxCoord(i,2);
    else
        markerslimit(i,2)=extra2*Matfile.maxCoord(i,2);
    end
end
%Compare Measured Markers limits to actual axis limits
for i=1:3
    if markerslimit(i,1)<axislimit(i,1)
        axislimit(i,1)=markerslimit(i,1);
    end
    if markerslimit(i,2)>axislimit(i,2)
        axislimit(i,2)=markerslimit(i,2);
    end
end
%Compare Longest LCS limit to actual axis limits
for i=1:3
    if -maxLCS<axislimit(i,1)
        axislimit(i,1)=-maxLCS;
    end
    if maxLCS>axislimit(i,2)
        axislimit(i,2)=maxLCS;
    end
end

%Round axis limits to 2 decimal places
for i=1:3
    axislimit(i,1)=round(axislimit(i,1),2);
    axislimit(i,2)=round(axislimit(i,2),2);
end
%Print axis limits on boxes
app.XLimminEditField.Value=axislimit(1,1);
app.XLimmaxEditField.Value=axislimit(1,2);
app.YLimminEditField.Value=axislimit(2,1);
app.YLimmaxEditField.Value=axislimit(2,2);
app.ZLimminEditField.Value=axislimit(3,1);
```

```
app.ZLimmaxEditField.Value=axislimit(3,2);
%Set axis limits
axis(appax,([axislimit(1,1) axislimit(1,2),axislimit(2,1) axislimit(2,2),axislimit(3,1) axislimit(3,2)]));
axis(appax,'manual');
grid(appax,'off');
%Delete axis ticks
set(appax,'XTickMode','manual');
set(appax,'YTickMode','manual');
set(appax,'ZTickMode','manual');
set(appax,'XTick',[]);
set(appax,'YTick',[]);
set(appax,'ZTick',[]);
%Set the axis as tight as possible in the figure
InSet = get(appax, 'TightInset');
set(appax, 'Position', [InSet(1:2), 1-InSet(1)-InSet(3), 1-InSet(2)-InSet(4)]);
set(appax,'CameraUpVector',[0 1 0],'CameraUpVectorMode','manual');
%Display Cameratoolbar
cameratoolbar(appfigure,'Show');
axis(appax,'off');
drawnow;
%End of loading code
```

ANEXO H: Código del primer bucle de pausa del callback del botón LOAD ANIMATION

```
%Pausing code
if PauseAnimation == 1 && activation == 0
   while(1)
      pause(0.01);

      %Closing Code
      if StopAnimation==1
         slide=0;
         break;
      end
      %End of Closing Code

      if PauseAnimation==0
         slide=0;
         break;
      end

      if FrameForward==1
         i=i+1;
         frame=frame+1;
         activation=1;
         slide=0;
         break;
      elseif FrameBackward==1
         i=i-1;
         frame=frame-1;
         activation=1;
         slide=0;
         break;
      end

      if app.FrameEditField.Value~=frame+1
         frame=app.FrameEditField.Value;
         i=frame;
         slide=0;
         break;
      end

   end
end
%End of pausing code
```

ANEXO I: Código del segundo bucle de pausa del callback del botón LOAD ANIMATION

```
%Second Pausing Code
if PauseAnimation==1
    while (1)

        pause(0.01);

        %Closing Code
        if StopAnimation==1
            slide=0;
            break;
        end
        %End of Closing Code

        %Quits loop if it has reached first or last frame
        %using the slider
        if slide==1 && frame==nSamples-1
            break;
        elseif slide==1 && frame==0
            break;
        end

        if activation == 1
            FrameForward=0;
            FrameBackward=0;
            activation=0;
        elseif activation==2
            activation=0;
        end

        if PauseAnimation==0
            slide=0;
            break;
        end

        if FrameForward==1 || FrameBackward==1
            activation=1;
            slide=0;
            break;
        end

        if app.FrameEditField.Value~=frame
            frame=app.FrameEditField.Value;
            i=frame;
            activation=2;
            slide=0;
            break;
        end
    end
end
%End of Second Pausing Code
```

ANEXO J: Callback asociado al botón "Play Forward"

```
function PlayForwardButtonPushed(app, event)
try
    global PlayForward
    global PlayBackward
    global PauseAnimation
    PlayForward = 1;
    PlayBackward = 0;
    app.PauseButton.Value=false;
    PauseAnimation = 0;
catch ME
    return;
end
end
```

ANEXO K: Callback asociado al checkbox “show/hide all LCS”

```
function ShowHideallLCSCheckBoxValueChanged(app, event)
try
    %Declare global variables
    global H
    global nSegments

    value = app.ShowHideallLCSCheckBox.Value;
    for i=1:nSegments
        if ~isempty(H{i,2})
            if value == true
                set(H{i,2},'Visible','on');
            elseif value == false
                set(H{i,2},'Visible','off');
            end
        end
    end
catch ME
    return;
end
end
```

ANEXO L: Callback asociado a la barra slider

```
function FrameSliderValueChanging(app, event)
try
    rounded=round(event.Value,0);

    global H
    global MODS MEDS JNT
    global AuxWire
    global ModelMarkerCenter
    global T_Segments T_MeasuredMarkers
    global Time
    global frame

    global slide
    slide=1;

    nSegments = size(T_Segments,1);
    nMeasuredMarkers = size(T_MeasuredMarkers,1);

    i = rounded;
    frame = i;
    for j = 1:nMeasuredMarkers
        %If experimental Marker is at the origin it means it
        %has not been read correctly so no line is drawn
        if T_MeasuredMarkers{j,i+1}(1,4)==0 && T_MeasuredMarkers{j,i+1}(2,4)==0 && T_MeasuredMarkers{j,i+1}(3,4)==0
            set(MEDS{j,1},'Visible','off');
        else
            T = T_MeasuredMarkers{j,(i+1)};
            set(MEDS{j,1},'Matrix',T);
            if strcmp(MEDS{j,1}.Visible,'off')==1
                set(MEDS{j,1},'Visible','on');
            end
        end

        %If experimental Marker is at the origin it means it
        %has not been read correctly so no line is drawn
        if T_MeasuredMarkers{j,i+1}(1,4)==0 && T_MeasuredMarkers{j,i+1}(2,4)==0 && T_MeasuredMarkers{j,i+1}(3,4)==0
            set(AuxWire(j,1),'Visible','off');
        else
            x=[ModelMarkerCenter{j,i+1}(1,1) T_MeasuredMarkers{j,i+1}(1,4)];
            y=[ModelMarkerCenter{j,i+1}(2,1) T_MeasuredMarkers{j,i+1}(2,4)];
            z=[ModelMarkerCenter{j,i+1}(3,1) T_MeasuredMarkers{j,i+1}(3,4)];
            set(AuxWire(j,1),'XData',x,'YData',y,'ZData',z);
            if strcmp(AuxWire(j,1).Visible,'off')==1
                set(AuxWire(j,1),'Visible','on');
            end
        end
    end
    for j = 1:nSegments
        T = T_Segments{j,(i+1)};
        set(MODS{j,1},'Matrix',T);

        set(JNT{j,1},'Matrix',T);

        T = T_Segments{j,(i+1)};    %Obtain the transformation matrix T from the .MAT file
        set(H{j,1},'Matrix',T);  % Update the object
        set(H{j,2},'Matrix',T);    % Update the object's LCS
        app.TimeTextArea.Value = num2str(Time{1,(i+1)});
        app.FrameEditField.Value=frame; %Print frame number in frame box
        app.FrameSlider.Value=frame; %Print frame number in frame box
    end
catch ME
    return;
end
end
```

ANEXO M: Código de la función stlread

```
function varargout = stlread(file)
% STLREAD imports geometry from an STL file into MATLAB.
%    FV = STLREAD(FILENAME) imports triangular faces from the ASCII or binary
%    STL file idicated by FILENAME, and returns the patch struct FV, with fields
%    'faces' and 'vertices'.
%
%    [F,V] = STLREAD(FILENAME) returns the faces F and vertices V separately.
%
%    [F,V,N] = STLREAD(FILENAME) also returns the face normal vectors.
%
%    The faces and vertices are arranged in the format used by the PATCH plot
%    object.
% Copyright 2011 The MathWorks, Inc.
    if ~exist(file,'file')
        error(['File "%s" not found. If the file is not on MATLAB"s path' ...
              ', be sure to specify the full path to the file.'], file);
    end

    fid = fopen(file,'r');
    if ~isempty(ferror(fid))
        error(lasterror); %#ok
    end

    M = fread(fid,inf,'uint8=>uint8');
    fclose(fid);

    [f,v,n] = stlbinary(M);
    %if( isbinary(M) ) % This may not be a reliable test
    %    [f,v,n] = stlbinary(M);
    %else
    %    [f,v,n] = stlascii(M);
    %end

    varargout = cell(1,nargout);
    switch nargout
        case 2
            varargout{1} = f;
            varargout{2} = v;
        case 3
            varargout{1} = f;
            varargout{2} = v;
            varargout{3} = n;
        otherwise
            varargout{1} = struct('faces',f,'vertices',v);
    end
end
function [F,V,N] = stlbinary(M)
    F = [];
    V = [];
    N = [];

    if length(M) < 84
        error('MATLAB:stlread:incorrectFormat', ...
              'Incomplete header information in binary STL file.');
    end

    % Bytes 81-84 are an unsigned 32-bit integer specifying the number of faces
    % that follow.
    numFaces = typecast(M(81:84),'uint32');
    %numFaces = double(numFaces);
    if numFaces == 0
        warning('MATLAB:stlread:nodata','No data in STL file.');
        return
    end

    T = M(85:end);
    F = NaN(numFaces,3);
    V = NaN(3*numFaces,3);
    N = NaN(numFaces,3);

    numRead = 0;
    while numRead < numFaces
```

```
        % Each facet is 50 bytes
        %  - Three single precision values specifying the face normal vector
        %  - Three single precision values specifying the first vertex (XYZ)
        %  - Three single precision values specifying the second vertex (XYZ)
        %  - Three single precision values specifying the third vertex (XYZ)
        %  - Two unused bytes
        i1    = 50 * numRead + 1;
        i2    = i1 + 50 - 1;
        facet = T(i1:i2)';

        n  = typecast(facet(1:12),'single');
        v1 = typecast(facet(13:24),'single');
        v2 = typecast(facet(25:36),'single');
        v3 = typecast(facet(37:48),'single');

        n = double(n);
        v = double([v1; v2; v3]);

        % Figure out where to fit these new vertices, and the face, in the
        % larger F and V collections.
        fInd  = numRead + 1;
        vInd1 = 3 * (fInd - 1) + 1;
        vInd2 = vInd1 + 3 - 1;

        V(vInd1:vInd2,:) = v;
        F(fInd,:)        = vInd1:vInd2;
        N(fInd,:)        = n;

        numRead = numRead + 1;
    end

end
function [F,V,N] = stlascii(M)
    warning('MATLAB:stlread:ascii','ASCII STL files currently not supported.');
    F = [];
    V = [];
    N = [];
end
% TODO: Change the testing criteria! Some binary STL files still begin with
% 'solid'.
function tf = isbinary(A)
% ISBINARY uses the first line of an STL file to identify its format.
    if isempty(A) || length(A) < 5
        error('MATLAB:stlread:incorrectFormat', ...
              'File does not appear to be an ASCII or binary STL file.');
    end
    if strcmpi('solid',char(A(1:5)'))
        tf = false; % ASCII
    else
        tf = true;  % Binary
    end
end
```

ANEXO N: Código de la función drawsphere

```
function [X,Y,Z] = drawsphere(r,x0,y0,z0)

%Create a unit sphere centered in the origin
[x1,y1,z1] = sphere;

%Scale my multiplying by the radius
x2 = x1*r;
y2 = y1*r;
z2 = z1*r;

X = x2+x0;
Y = y2+y0;
Z = z2+z0;

end
```

ANEXO O: Código de la función LCSplot

```
function h=LCSplot(ax,X,Y,Z,SegmentName)

h=gobjects(3,2);

% find scale
MaximumCoords=zeros(1,3);
MaximumCoords(1,1)  = max(max(abs(X(:,:))));
MaximumCoords(1,2)  = max(max(abs(Y(:,:))));
MaximumCoords(1,3)  = max(max(abs(Z(:,:))));
MaxCoord = max(MaximumCoords);
LCSlength = 1.3 * MaxCoord;

% plot axes vectors
h(1,1)=quiver3(ax,0,0,0,LCSlength,0,0,0,'r','LineWidth',2);
h(2,1)=quiver3(ax,0,0,0,0,LCSlength,0,0,'g','LineWidth',2);
h(3,1)=quiver3(ax,0,0,0,0,0,0,LCSlength,0,'b','LineWidth',2);

% plot axes names
axisname=strings(1,3);
if strcmp(SegmentName,'Ground')==1 %Enters here if it is drawing the GCS
    axisname(1,1)=strcat('X_{',SegmentName,'}   ');
    axisname(1,2)=strcat('Y_{',SegmentName,'}   ');
    axisname(1,3)=strcat('Z_{',SegmentName,'}   ');
    h(1,2)=text(ax,LCSlength,0,0,axisname(1,1),'FontSize',12,'FontWeight','Bold','Color','r','HorizontalAlignment','right','BackgroundCol
or','none'); % X-axis
    h(2,2)=text(ax,0,LCSlength,0,axisname(1,2),'FontSize',12,'FontWeight','Bold','Color','g','HorizontalAlignment','right','BackgroundCo
lor','none'); % Y-axis
    h(3,2)=text(ax,0,0,LCSlength,axisname(1,3),'FontSize',12,'FontWeight','Bold','Color','b','HorizontalAlignment','right','BackgroundCo
lor','none'); % Z-axis
else %Enters here if it is drawing a LCS
    axisname(1,1)=strcat('X_{',SegmentName,'}   ');
    axisname(1,2)=strcat('Y_{',SegmentName,'}   ');
    axisname(1,3)=strcat('Z_{',SegmentName,'}   ');
    h(1,2)=text(ax,LCSlength,0,0,axisname(1,1),'FontSize',10,'FontWeight','normal','Color','r','HorizontalAlignment','right','Background
Color','none'); % X-axis
    h(2,2)=text(ax,0,LCSlength,0,axisname(1,2),'FontSize',10,'FontWeight','normal','Color','g','HorizontalAlignment','right','Background
Color','none'); % Y-axis
    h(3,2)=text(ax,0,0,LCSlength,axisname(1,3),'FontSize',10,'FontWeight','normal','Color','b','HorizontalAlignment','right','Background
Color','none'); % Z-axis
end
```

ANEXO P: Código de la función C3D2MAT

```
function C3D2MAT(InputPath,InputFile)

% receives a C3D file, extracts all data and saves part of it into a MAT file.
% It is usefull when C3DServer software is not available. For example for
% Apple computers


% read C3D file
[Markers, Frequency, IndexAllVisible, MarkerNames, MarkerCoords] = readC3D(InputPath, InputFile);

% get only filename
[Filename, ~] = getFilenameAndExt(InputFile);

% save data in a MAT file.
disp(['Saving motion data in file ',InputFile,' ...'])
save([InputPath,Filename],'Frequency','MarkerNames','MarkerCoords');
disp(['Process finished!'])
```

ANEXO Q: Código de la función getColor

```
%Function to convert the color read from .xml to rgb color which Matlab
%can read
function colorrgb = getColor(app,color)
if strcmp(color,'Black')==1
    colorrgb='k';
elseif strcmp(color,'Dark Red')==1
    colorrgb=[0.5 0 0];
elseif strcmp(color,'Red')==1
    colorrgb='r';
elseif strcmp(color,'Pink')==1
    colorrgb='m';
elseif strcmp(color,'Rose')==1
    colorrgb=[1 0.6 0.8];
elseif strcmp(color,'Brown')==1
    colorrgb=[0.6 0.2 0];
elseif strcmp(color,'Orange')==1
    colorrgb=[1 0.4 0];
elseif strcmp(color,'Light Orange')==1
    colorrgb=[1 0.6 0];
elseif strcmp(color,'Gold')==1
    colorrgb=[1 0.8 0];
elseif strcmp(color,'Tan')==1
    colorrgb=[1 0.8 0.6];
elseif strcmp(color,'Olive Green')==1
    colorrgb=[0.2 0.2 0];
elseif strcmp(color,'Dark yellow')==1
    colorrgb=[0.5 0.5 0];
elseif strcmp(color,'Lime')==1
    colorrgb=[0.6 0.8 0];
elseif strcmp(color,'Yellow')==1
    colorrgb='y';
elseif strcmp(color,'Light Yellow')==1
    colorrgb=[1 1 0.6];
elseif strcmp(color,'Dark Green')==1
    colorrgb=[0 0.2 0];
elseif strcmp(color,'Green')==1
    colorrgb=[0 0.5 0];
elseif strcmp(color,'Sea Green')==1
    colorrgb=[0.2 0.6 0.4];
elseif strcmp(color,'Bright Green')==1
    colorrgb='g';
elseif strcmp(color,'Light Green')==1
    colorrgb=[0.8 1 0.8];
elseif strcmp(color,'Dark Teal')==1
    colorrgb=[0 0.2 0.4];
elseif strcmp(color,'Teal')==1
    colorrgb=[0 0.5 0.5];
elseif strcmp(color,'Aqua')==1
    colorrgb=[0.2 0.8 0.8];
elseif strcmp(color,'Turquoise')==1
    colorrgb='c';
elseif strcmp(color,'Light Turquoise')==1
    colorrgb=[0.8 1 1];
elseif strcmp(color,'Dark Blue')==1
    colorrgb=[0 0 0.5];
elseif strcmp(color,'Blue')==1
    colorrgb='b';
elseif strcmp(color,'Light Blue')==1
    colorrgb=[0.2 0.4 1];
elseif strcmp(color,'Sky Blue')==1
    colorrgb=[0 0.8 1];
elseif strcmp(color,'Pale Blue')==1
    colorrgb=[0.6 0.8 1];
elseif strcmp(color,'Indigo')==1
    colorrgb=[0.2 0.2 0.6];
elseif strcmp(color,'Blue-Gray')==1
    colorrgb=[0.4 0.4 0.6];
elseif strcmp(color,'Violet')==1
    colorrgb=[0.5 0 0.5];
elseif strcmp(color,'Plum')==1
    colorrgb=[0.6 0.2 0.4];
elseif strcmp(color,'Lavender')==1
```

```
    colorrgb=[0.8 0.6 1];
elseif strcmp(color,'Gray-80%')==1
    colorrgb=[0.2 0.2 0.2];
elseif strcmp(color,'Gray-50%')==1
    colorrgb=[0.5 0.5 0.5];
elseif strcmp(color,'Gray-40%')==1
    colorrgb=[0.588 0.588 0.588];
elseif strcmp(color,'Gray-25%')==1
    colorrgb=[0.753 0.753 0.753];
elseif strcmp(color,'White')==1
    colorrgb='w';
end
end
```

ANEXO R: MANUAL DE USUARIO DE LA APLICACIÓN

2. Model definition

`Graphic` defines a graphic object associated to the segment. If you define a drawing sequence (`DrawSeq`) the points defined in `DrawSeq` are connected to a cylinder of radius `Radius="0.01"` and colour `Colour="gray"`. The possible values for `Colour` are the following:

Black	Dark Teal
Dark Red	Teal
Red	Aqua
Pink	Turquoise
Rose	Light Turquoise
Brown	Dark Blue
Orange	Blue
Light Orange	Light Blue
Gold	Sky Blue
Tan	Pale Blue
Olive Green	Indigo
Dark Yellow	Blue-Gray
Lime	Violet
Yellow	Plum
Light Yellow	Lavender
Dark Green	Gray-80%
Green	Gray-50%
Sea Green	Gray-40%
Bright Green	Gray-25%
Light Green	White

7. Results visualization with MATLAB

The MoCap results can be viewed using Matlab by calling the app (MotVis3D) from the command window. Once the app is opened just press the LOAD ANIMATION button located at the top and select the .MAT results file.

Play/pause the motion

To play or pause the motion use the buttons show in the figure below. The middle button pauses the animation. The buttons located on the left and right side of the pause button play the motion backwards or forwards respectively. The buttons located on both ends can be used to animate the motion frame by frame.

The playing speed can be changed by editing the field shown below on the control panel.

The slider bar can be used to play the motion manually.

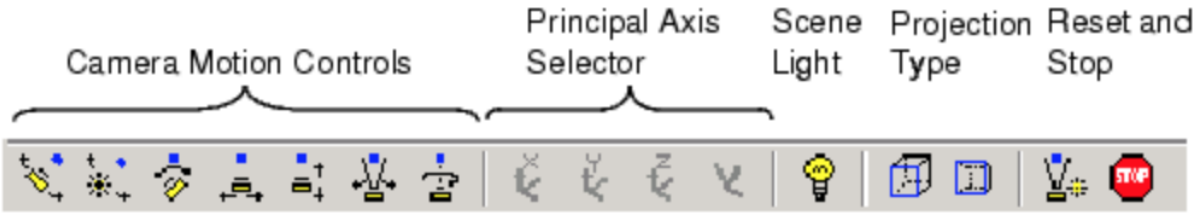
It can be chosen what frame to display by editing the *Frame* field shown below. The *Time* field shows the actual time but it is not editable.

Fit the model in the window

The model automatically fits in the window. However, the limits of the axes can be changed by editing the maximum and minimum fields of each axis.

Interact with the viewing of the model

Use the camera toolbar to interact with the animation.



From left to right the camera motion controls perform the following function:

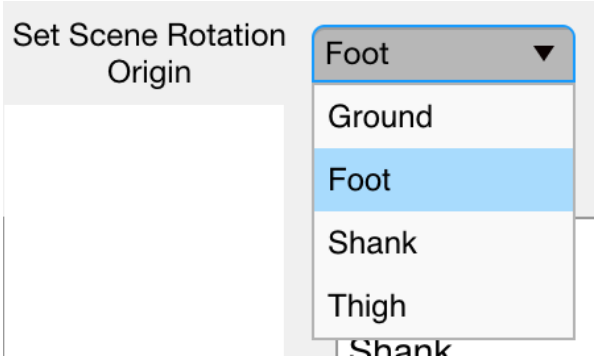
- Orbit Camera: rotates the camera about the selected axis.
- Orbit Scene Light
- Pan/Tilt Camera: moves the point in the scene that the camera points to while keeping the camera fixed.
- Move Camera Horizontally/Vertically

- Move Camera Forward and Backward
- Zoom Camera
- Camera Roll: rotates the camera about the viewing axis, thereby rotating the view on the screen.

Principal Axis Selector is used to select the rotation axis (x, y, z or no principal axis).

Select the scene rotation point

Select the scene rotation origin on the display menu. The model rotates with respect to the origin of the axes of the segment selected. The Ground's axes are the Global axes.



Show and hide segments, axes and markers

Select or deselect the checkbox corresponding to each graphic object to show or hide it. The list boxes have the same function but offer the possibility to perform it individually. These components show the name of all the segments of the model. If a segment's name is clicked on the list box of the left-hand side, its Local Coordinate System's visibility changes, if it is hidden it becomes visible and vice versa.

The list box on the right has an analogous operation principle, but it hides and shows the segment itself.

Stop the motion and close the app

To close the app just press the red button CLOSE ANIMATION located at the bottom.