CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Creación de personajes autónomos animados en la herramienta de desarrollo de entornos virtuales *Unity 3D*

Autor: Carlos Castillo Venegas

Directores: Angélica de Antonio Jiménez
Daniel Klepel

MADRID, JUNIO DE 2013

Abstract

Se presenta el desarrollo y resultados que se obtuvieron en la realización del TFG. Este trabajo se centra en la construcción de la parte física del personaje virtual. El desarrollo muestra técnicas de modelado 3D, cinemática y animación usadas para la creación de personajes virtuales. Se incluye además una implementación que está dividida en: modelado del personaje virtual, creación de un sistema de cinemática inversa y la creación de animaciones utilizando el sistema de cinemática. Primero, crear un modelo 3D exacto al diseño original, segundo, el desarrollo de un sistema de cinemática inversa que resuelva con exactitud las posiciones de las partes articuladas que forman el personaje virtual, y tercero, la creación de animaciones haciendo uso del sistema de cinemática para conseguir animaciones fluidas y depuradas. Como consecuencia, se ha obtenido un componente 3D animado, reutilizable, ampliable, y exportable a otros entornos virtuales.

The obtained development and results are presented in this TFG. This article is pointed in the making of the physical part of the virtual character. Development shows modeling 3D, kinematic and animation techniques used for create the virtual character. In addition, an implementation is included, and it is divided in: to model the 3D character, to create an inverse kinematics system, and to create animations using a kinematic system. First, creating an exact 3D model from the original design, second, developing an inverse kinematics system that resolves the positions of the articulated pieces that compose the virtual character, and third, creating animation using the inverse kinematics system to get fluid and refined animations in realtime. As consequence, a 3D animated, reusable, extendable and to other virtual environments exportable component has been obtained.

Dedicatoria

El presente trabajo se lo dedico a mi familia ya que gracias a su apoyo he podido cursar mi carrera, especialmente a mis padres y hermana por darme su apoyo y confianza en todos mis proyectos.

A Irene Escribano quién a estado siempre a mi lado ayudándome y guiándome a lo largo de estos años. También a mis amigos quienes han sido un punto de apoyo en los momentos de dificultad.

En especial a mis abuelos Pascual y Tiburcio y mi primo José que aunque no están aquí siempre formarán parte de mí. En general a aquellas todas personas que me han proporcionado su apoyo y comprensión a lo largo de mi vida.

Agradecimientos

En primer lugar quiero dar las gracias a mis tutores del trabajo fin de grado, Angélica de Antonio Jiménez y Daniel Klepel, ya que sin su ayuda este trabajo no hubiese salido adelante. En especial a Daniel Klepel por su ayuda y su tiempo dedicado en enseñarme una parte de la informática que desconocía totalmente.

A mis compañeros que han compartido conmigo los años de estudio y con quienes he compartido trabajos y horas de estudio. Por último, a mi familia, novia y amigos por apoyarme y darme fuerzas cada día.

Índice general

Índice de figuras

Indice de tablas

Capítulo 1

Introducción

1.1. Motivación

En el presente trabajo, *Creación de personajes autónomos animados en la herramienta de desarrollo de entornos virtuales Unity 3D*, se busca la implementación de un personaje virtual a través de un componente 3D con animaciones propias que puedan ser utilizadas por un módulo agente que implementa la mente del personaje. La necesidad de crear un componente animado surge al usar entornos virtuales donde los elementos 3D realizan una interacción con los usuarios que visitan el entorno virtual. Además, y debido a la forma de integración de las animaciones, este trabajo permite realizar un estudio de los diversos métodos existentes en el sector de la animación. La animación por cinemática inversa, en concreto, es un concepto estudiado mayoritariamente por la robótica y se busca aplicarlo en entornos virtuales.

1.2. Necesidades

Mientras se procedía a la realización del *prácticum* en el laboratorio Decoroso Crespo de la Facultad de Informática de la Universidad Politécnica de Madrid, surgió la necesidad de animar elementos insertados dentro de entornos virtuales, concretamente, en el entorno virtual creado para el proyecto Miles.

Existen diversas técnicas de animación, pero este proyecto se centrará en la animación de componentes articulados mediante cinemática. La cinemática se divide en dos grupos: [5]

- Cinemática inversa: Se basa en solucionar la posición final de los ángulos de las uniones sin manipulación directa de las mismas .
- Cinemática directa: Se basa en solucionar la posición final a partir de manipulación directa de las articulaciones.

En lo referente a este TFG se centrará en la animación mediante cinemática inversa, la cuál será expuesta más adelante.

El trabajo a realizar deberá integrarse en la plataforma de desarrollo *Unity3D*. Esta plataforma en su cuarta versión proporciona un nuevo sistema de desarrollo de animaciones proporcionando una tecnología que ha sido usada para la consecución de los objetivos.

Además, la herramienta *software Unity3D* proporciona un sistema de cinemática inversa, pero solo para modelos antropomórficos basados en tecnología de *biped* o *bones*[1].

El problema de este sistema de cinemática inversa es válido para modelos antropomórficos y por tanto es inútil para otros modelos, como es el caso del avatar generado (Ver figuras 1.1 y 1.2).

Como solución a este problema se propuso la creación de un sistema de cinemática inversa para el avatar modelado en particular y una interfaz reutilizable para otros tipos de modelos.

Referente a los comportamientos o animaciones se estableció que para dar uso al sistema de cinemática inversa se implementarían ciertos comportamientos accesibles mediante una interfaz. La activación de la ejecución de las animaciones será responsabilidad de un módulo agente desarrollado por un tercero.

1.3. Objetivos

El proyecto tiene diversos objetivos que fueron propuestas inicialmente. Se intentará demostrar su consecución a lo largo de este documento. Los objetivos son:

1. Construir un modelo 3D apropiado para el personaje y sus posibilidades de animación previstas.

2. Explorar los mecanismos de animación ofrecidos por la herramienta *Unity3D*.

3. Diseñar una serie de comportamientos básicos para el personaje, con posibilidades de parametrización y adaptación dinámica, incluyendo al menos la capacidad de desplazarse por el entorno 3D y de interactuar con los usuarios.

4. Especificar una interfaz de control para el personaje que pueda ser utilizada desde un agente *software* inteligente construido en Jade.

5. Realizar pruebas de funcionamiento.

6. Documentar el proceso de desarrollo.

7. Documentar los mecanismos para la creación de nuevos comportamientos sobre la base de lo desarrollado.

También y debido al proceso de realización del TFG se pueden extraer objetivos inherentes al proyecto. Estos son:

- Formación en entornos virtuales.

- Aprendizaje de técnicas de modelado.

- Comprensión de métodos existentes en la actualidad para creación de sistemas mediante cinemática inversa.

- Implementación de un método de cinemática inversa.

[1] `http://docs.unity3d.com/Documentation/Manual/MecanimAnimationSystem.html`

Figura 1.1: Muestra el avatar creado en la realización del TFG.

- Revisión, aprendizaje y uso de elementos matemáticos.

Por tanto, el objeto del trabajo es la solución al problema específico de creación de un componente 3D animado, y por otro lado, aportar una guía para futuras personas con necesidades similares.

1.4. Nomenclatura

A lo largo del TFG se utilizarán diferentes formas de reconocimiento para las partes del componente 3D. Por tanto, las distintas partes con sus correspondientes identificadores son:

- Robot_Ayuda: Referencia al avatar en su completud, es decir, es la agrupación de todas las partes que forman el modelo 3D (Ver figura 1.2).

- Seccion_Esfera: Referencia a la esfera del Robot_Ayuda (Ver figura 1.3).

- Proyector: Referencia al modelo 3D del proyector que pertenece al Robot_Ayuda (Ver figura 1.4).

- Union_Y: Referencia a la unión de las secciones, Seccion-1_Y y Seccion-2_Y, que se conectan a la Seccion_Esfera del Robot_Ayuda (Ver figura 1.5). Por otro lado, el valor de Y identifica cada una de las cuatro extremidades que forman el Robot_Ayuda.

- Seccion-X_Y: Referencia a cada una de las secciones que forman las extremidades móviles del Robot_Ayuda, siendo $X \in [1, 2]$ y $Y \in [1, 4]$. Es decir, el valor de X identifica a cada una de las partes de la extremidad. En caso de $X = 1$ identificará a la primera sección (Ver figura 1.6) y $X = 2$ identificará a la segunda sección (Ver figura 1.7). Por otro lado, el valor de Y identifica cada una de las cuatro extremidades que forman el Robot_Ayuda.

- PR_Y: Referencia al extremo no conectado a la Seccion_Esfera de la Union_Y (Ver figuras 1.2,1.3 y 1.5). Además, Y en este caso puede tomar el valor P para referenciar el extremo del Proyector.

- TG_Y: Referencia a los objetivos a alcanzar por las distintas Union_Y. Además, Y en este caso puede tomar el valor P para referenciar el objetivo a alcanzar por el Proyector.

También se usara el acrónimo IK para referirse a cinemática inversa (proviene del nombre anglosajón *Inverse Kinematic* y por el cual se identifica académicamente dicho método).

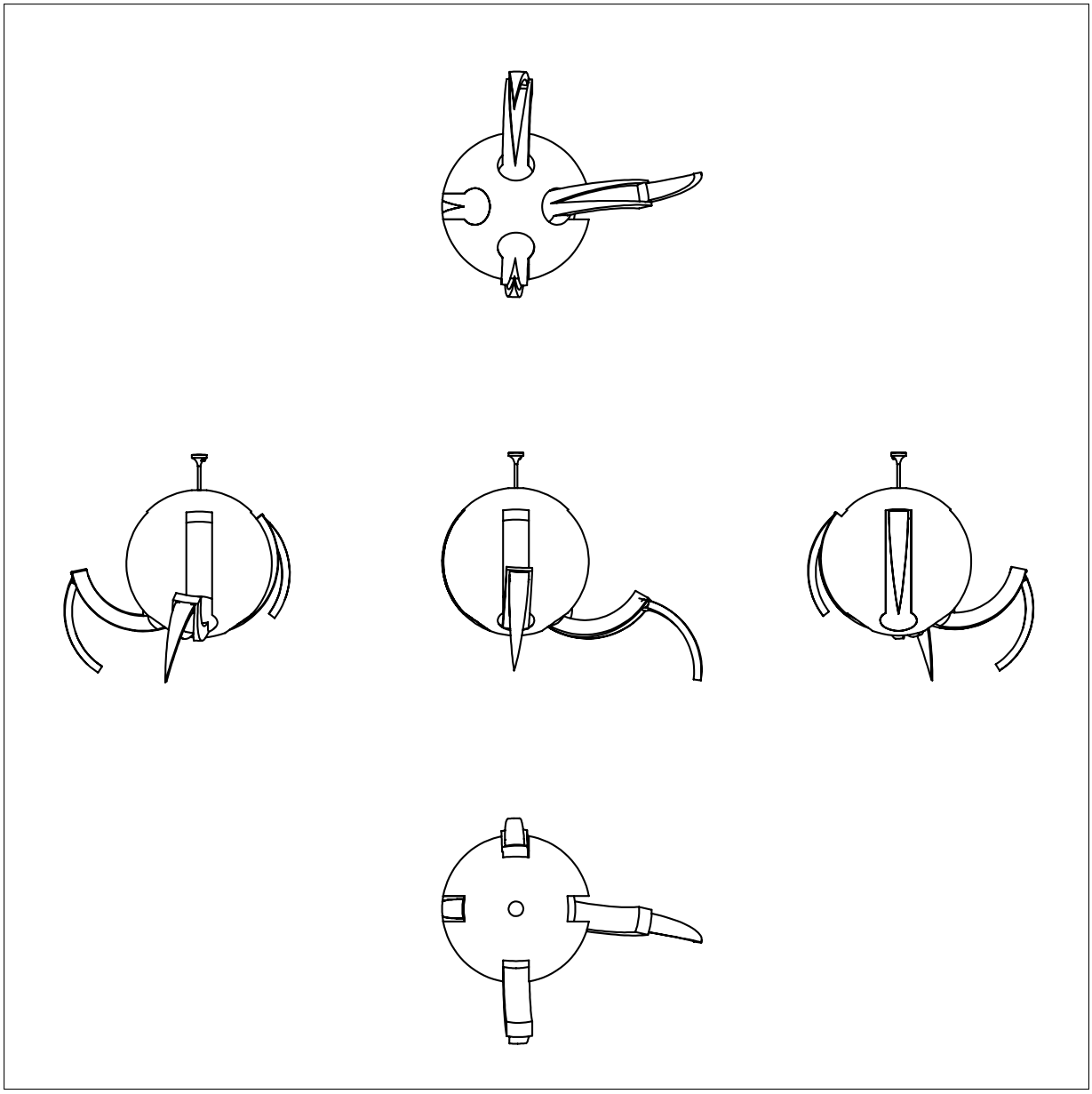


Figura 1.2: Plano del Robot_Ayuda. Incluye archivo multimedia visible con Acrobat versión 7 o superior.

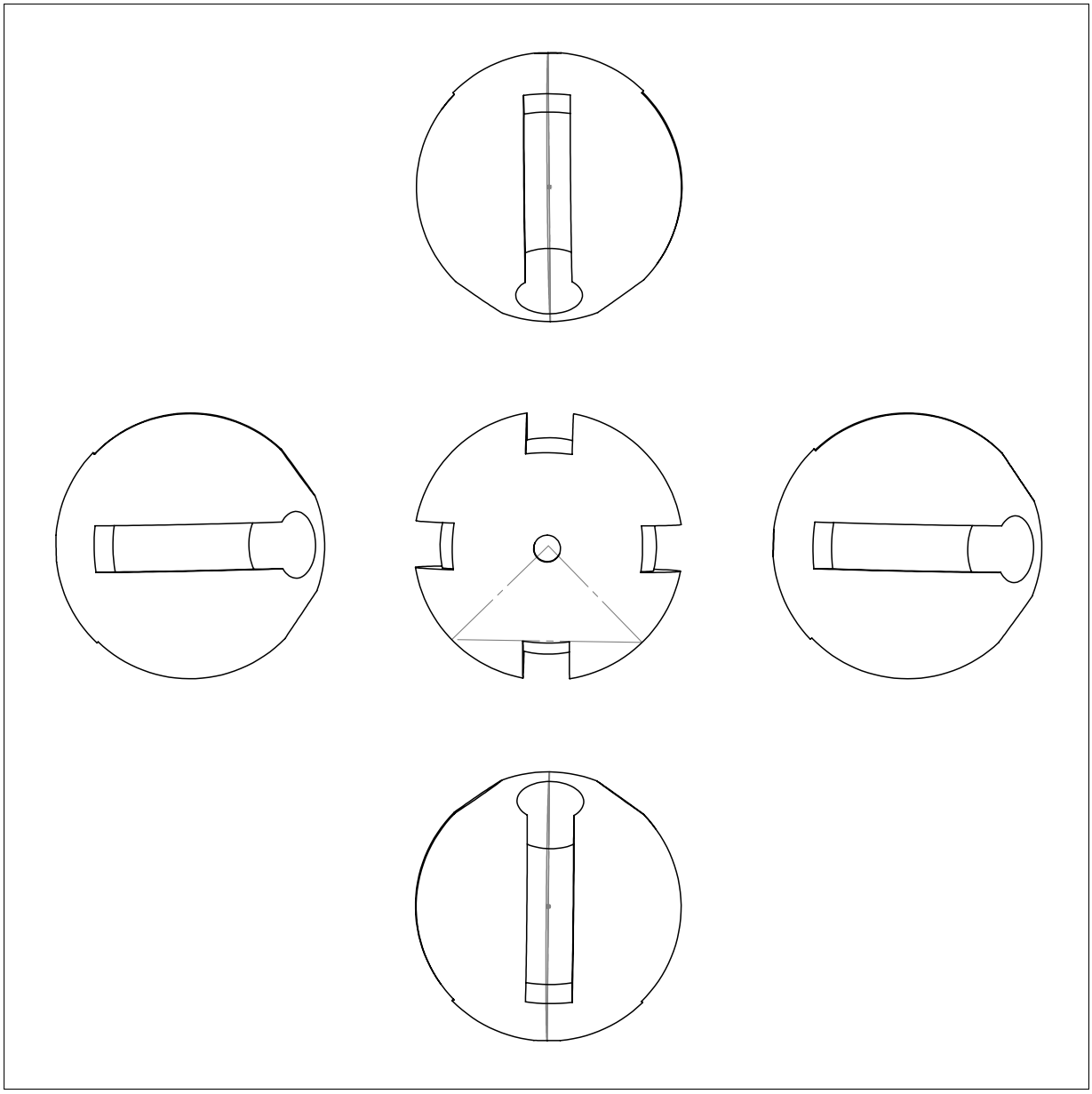


Figura 1.3: Seccion_Esfera. Incluye archivo multimedia visible con Acrobat versión 7 o superior.

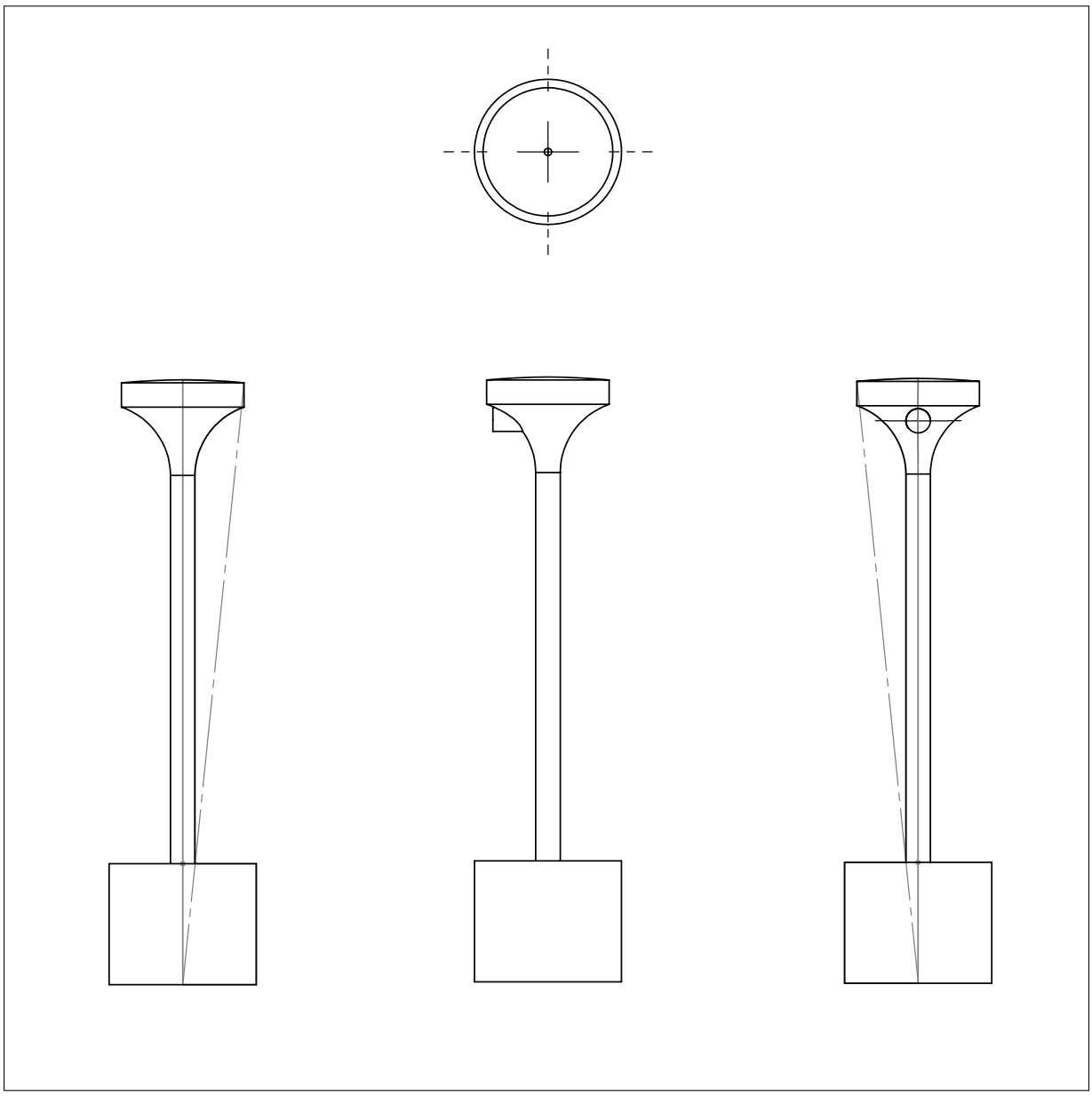


Figura 1.4: Proyector. Incluye archivo multimedia visible con Acrobat versión 7 o superior.

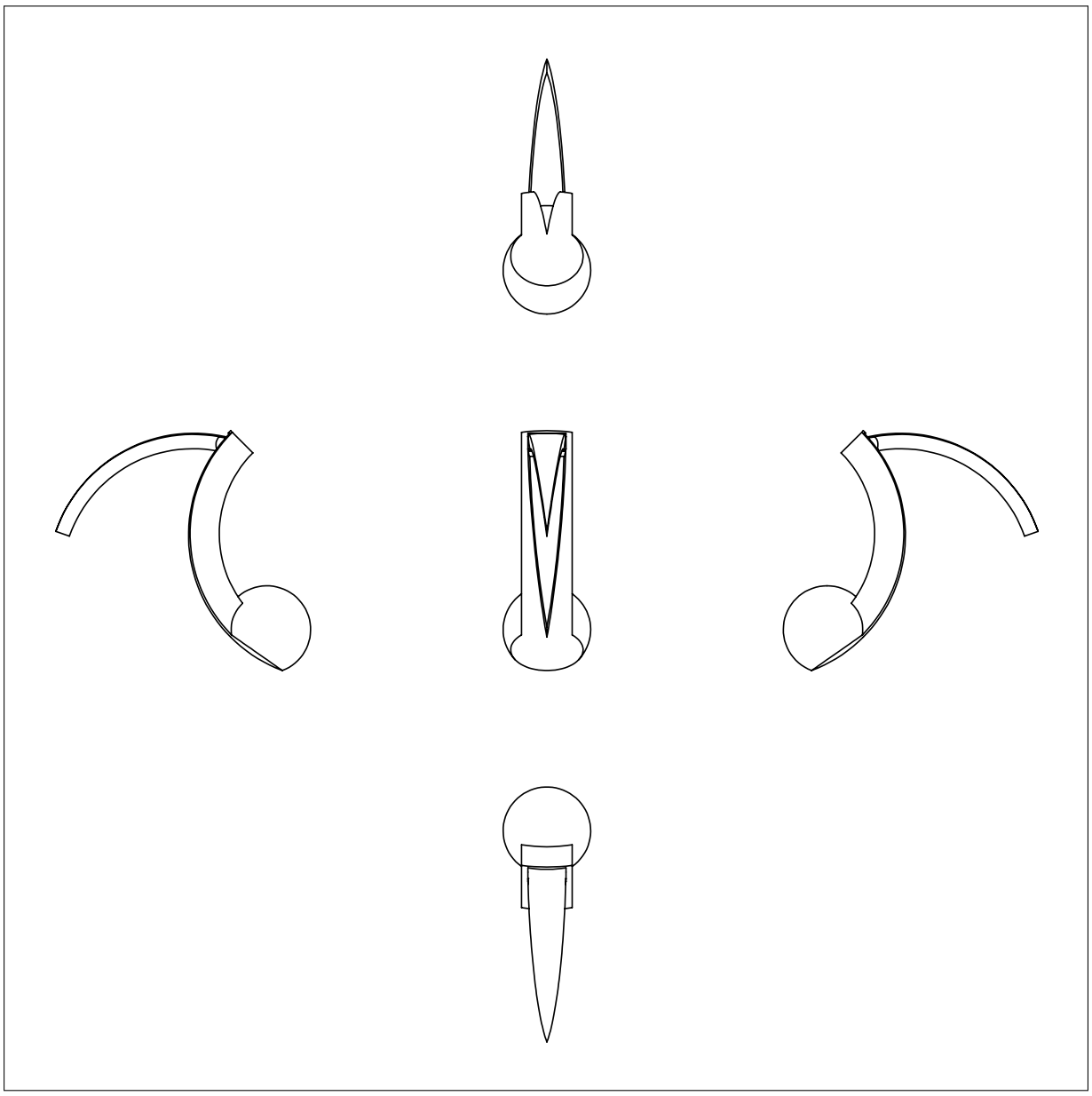


Figura 1.5: Union_Y. Incluye archivo multimedia visible con Acrobat versión 7 o superior.

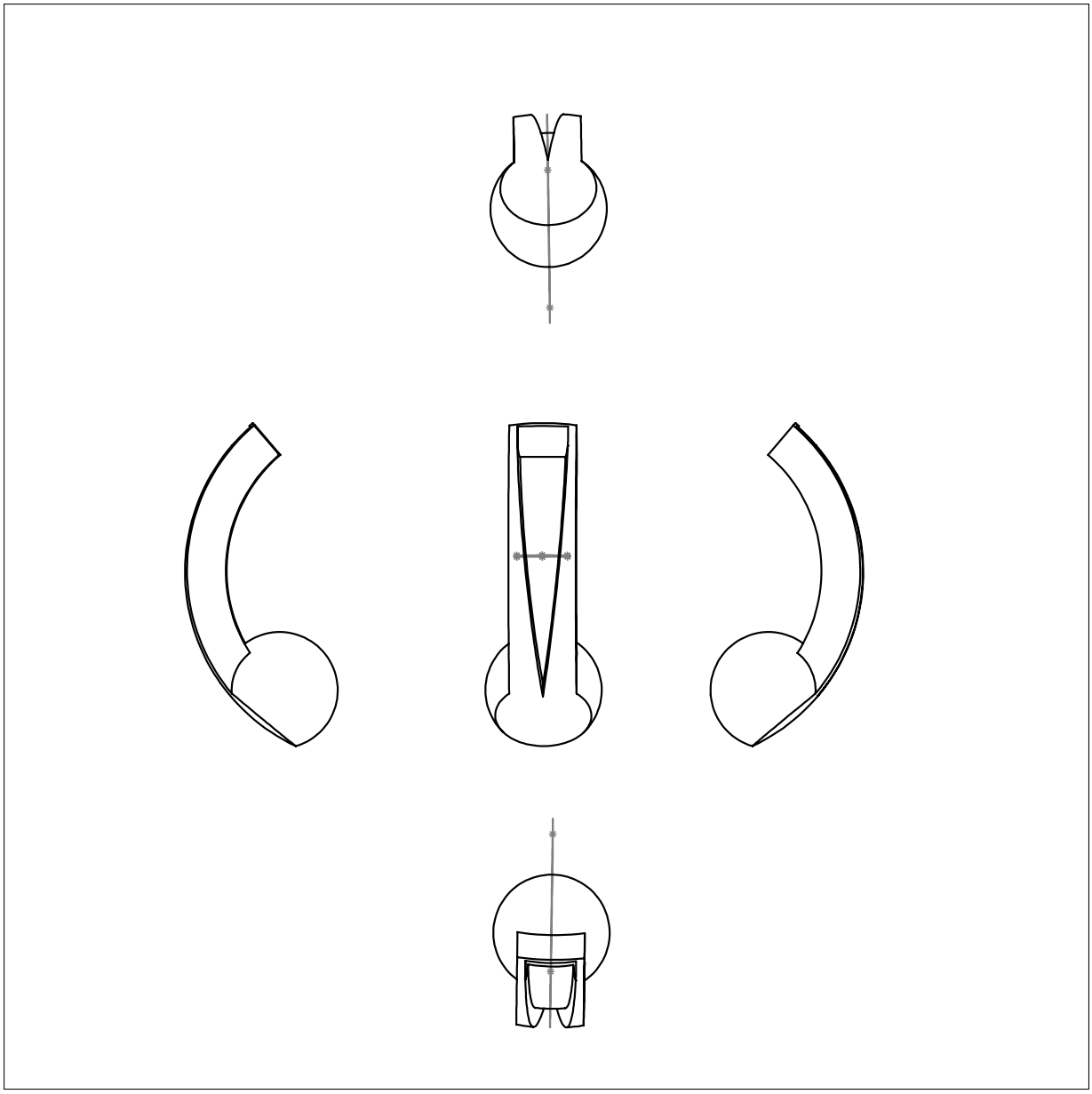


Figura 1.6: Seccion-1_Y. Incluye archivo multimedia visible con Acrobat versión 7 o superior.

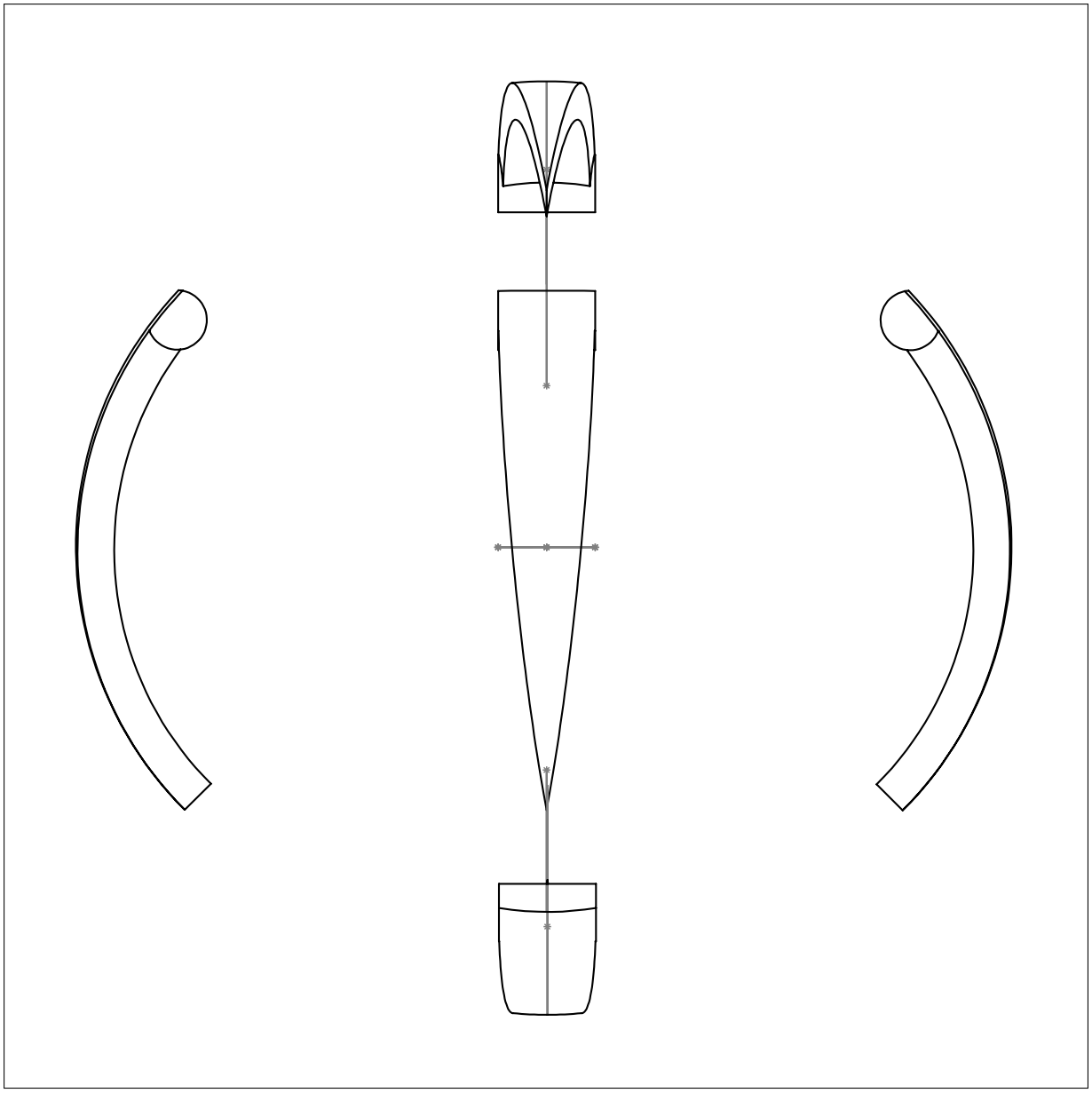


Figura 1.7: Seccion-2_Y. Incluye archivo multimedia visible con Acrobat versión 7 o superior.

1.5. Contenido del documento

El documento está dividido en los siguientes capítulos:

- *Antecedentes y estado del arte*: Se mostrará la situación actual en lo concerniente al TFG y conceptos necesarios para llevar a cabo el mismo.
- *Desarrollo*: Se expondrán los pasos seguidos para conseguir cumplir los objetivos iniciales del TFG. En este capítulo, se tratará el desarrollo del modelado 3D, la cinemática inversa, los comportamientos, la interfaz de acceso y pruebas.
- *Resultados y conclusiones*: Se tratará los resultados obtenidos, conclusiones alcanzadas y conocimientos adquiridos en la realización de este TFG.
- *Anexo*: Se añadirá información relevante relacionada con este TFG.

Capítulo 2

Antecedentes y estado del arte

2.1. Introducción

En este capítulo se mostrará la situación actual de los distintos apartados técnicos relacionados con el TFG. Se empezará exponiendo conceptos matemáticos utilizados. A continuación se tratará el tema del modelado 3D y se continuará con las técnicas de animación, sin entrar en detalles específicos debido a que esto se explicará con más concreción en el tercer capítulo, *Desarrollo*.

2.2. Conceptos matemáticos

2.2.1. Vector que une dos puntos

Dado dos puntos P y P', podemos trazar el vector v con inicio en P y final en P', donde escribimos $P\vec{P}'$ en lugar de v (Ver figura 2.1). [7]

Si $P = (x, y, z)$ y $P' = (x', y', z')$, entonces los vectores que van del origen a P y P' son $a = xi + yj + zk$ y $a' = x'i + y'j + z'k$, respectivamente, de modo que el vector $P\vec{P}'$ es la diferencia entre a y a' siendo:

$$a' - a = (x' - x)i + (y' - y)j + (z' - z)k \tag{2.1}$$

2.2.2. Longitud del vector

Se deduce del teorema de Pitágoras que la longitud del vector $a = a_1i + a_2j + a_3k$ es $\sqrt{a_1^2 + a_2^2 + a_3^2}$ (Ver figura 2.2). La longitud del vector a se denota por $||a||$. Esta cantidad se llama frecuentemente la norma de a. Como $a \cdot a = a_1^2 + a_2^2 + a_3^2$, se sigue que [7]

$$||a|| = (a \cdot a)^{\frac{1}{2}} \tag{2.2}$$

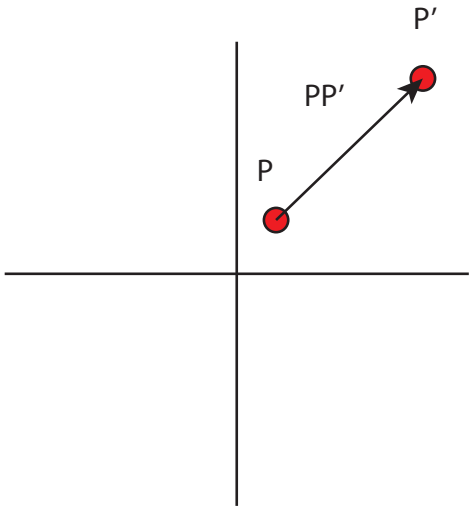


Figura 2.1: Muestra como se crea un vector a partir de dos puntos.

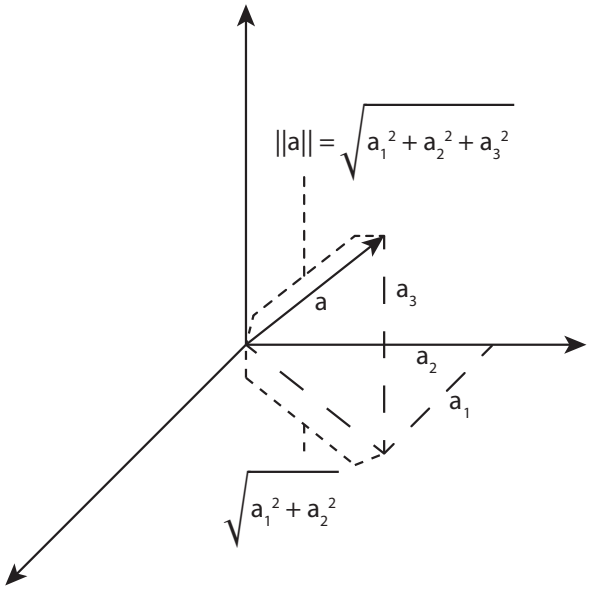


Figura 2.2: Muestra un vector descompuesto por variables en el eje de coordenadas XYZ.

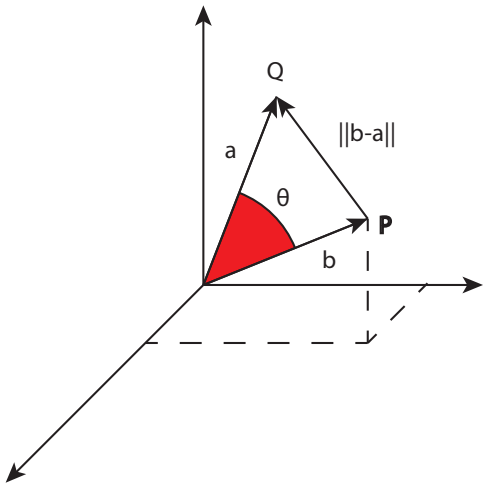


Figura 2.3: Muestra un par de vectores en el espacio que forman un ángulo θ y están a una distancia de $||b-a||$.

2.2.3. Vectores unitarios

Los vectores que tienen norma 1 se llaman vectores unitarios. Nótese que para cualquier vector a, $a/||a||$ es un vector unitario; cuando dividimos a entre $||a||$, decimos que hemos normalizado a. [7]

2.2.4. Distancia entre vectores

Si a y b son vectores, la distancia entre ambos extremos es precisamente $||b-a||$ (Ver figura 2.3). [7]

2.2.5. Ángulo entre vectores

Se deduce de la entidad $a \cdot b = ||a||||b|| \cos\theta$ que si a y b son distintos de cero, podemos expresar el ángulo que forman como (Ver figura 2.3): [7]

$$\theta = \arccos \frac{a \cdot b}{||a||||b||} \tag{2.3}$$

2.2.6. Producto escalar

Sean $a = a_1 i + a_2 j + a_3 k$ y $b = b_1 i + b_2 j + b_3 k$. Definimos el producto escalar de a y b, y los escribimos $a \cdot b$. como el número real [7]

$$a \cdot b = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{2.4}$$

Algunas propiedades se obtienen directamente de la definición. Si a y b son vectores en $\mathbb{R}^3$, y α y β números reales, entonces:

1. $a \cdot a > 0$; $a \cdot a = 0$ si y solo si $a = 0$.

2. $\alpha a \cdot b = \alpha(a \cdot b)$ y $a \cdot \beta b = \beta(a \cdot b)$.

3. $a \cdot b = b \cdot a$.

2.2.7. Producto vectorial

Dados dos vectores cualesquiera A y B, el producto vectorial $A \times B$ se define como un tercer vector C, cuya magnitud es $AB \sin\theta$, donde θ es el ángulo entre A y B, es decir, si C está dado por [6]

$$C = A \times B \tag{2.5}$$

entonces su magnitud es

$$C = |C| = |AB \sin\theta| \tag{2.6}$$

La dirección de $A \times B$ es perpendicular al plano formado por A y B y su sentido queda determinado por el avance de un tornillo de cuerda derecha, al hacerlo girar de A hacia B describiendo el ángulo menor de θ. Una regla más conveniente para ser utilizada respecto a la dirección $A \times B$ es la de la mano derecha o sacacorchos (Ver apartado 2.2.8).

Algunas de las propiedades del producto vectorial que se deducen a partir de su definición son las siguientes:

1. A diferencia del producto escalar el orden en que se multipliquen los dos vectores en un producto vectorial tiene importancia, es decir,

$$A \times B = -B \times A \tag{2.7}$$

   Por tanto,si se cambia el orden del producto vectorial debe cambiarse el signo.

2. Si A es paralelo a B, entonces $A \times B = 0$;por lo tanto, se concluye que $A \times A = 0$.

3. Si A es perpendicular a B, $|A \times B| = AB$.

2.2.8. Regla del sacacorchos o mano derecha

La regla de la mano derecha o sacacorchos se utiliza para determinar el sentido del vector resultante del producto vectorial $C = A \times B$. Se hace que los cuatro dedos de la mano derecha apunten a lo largo de A y, a continuación, se curvan hacia B, a través del ángulo menor formado, θ. La dirección en la que señale el pulgar derecho es la de C. La regla del sacacorchos tiene la misma fundamentación pero utilizando el giro del sacacorchos para determinar la dirección (Ver figura 2.4). [6]

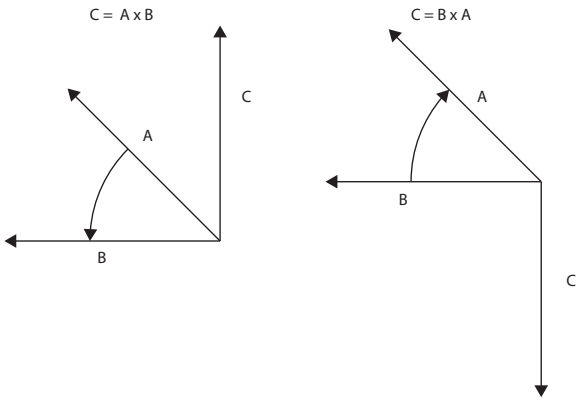


Figura 2.4: Muestra el funcionamiento de la regla de la mano derecha o sacacorchos.

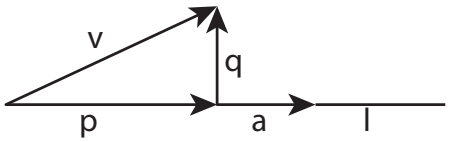


Figura 2.5: Muestra una proyección ortogonal donde p es la proyección ortogonal de v sobre a.

2.2.9. Proyección ortogonal

Si v es un vector, y l es la recta que pasa por el origen y tiene la dirección del vector a, la proyección ortogonal de v sobre a es el vector p cuyo extremo se obtiene al trazar una recta perpendicular a l desde el extremo de v. Mirando la figura 2.5, vemos que p es un múltiplo de a, y que v es la suma de p y un vector q perpendicular a a. Por tanto,

$$v = ca + q \tag{2.8}$$

, donde $p = ca$ y $a \cdot q = 0$. Si multiplicamos por a escalarmente en ambos lados de la igualdad $v = ca + q$, tenemos que $a \cdot v = ca \cdot a$, de modo que $c = (a \cdot v)/(a \cdot a)$, y entonces

$$p = \frac{a \cdot v}{||a||^2} a \tag{2.9}$$

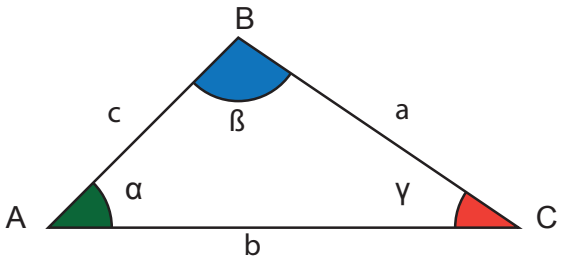


Figura 2.6: Muestra el triángulo a solucionar en el plano XY.

2.2.10. Teorema del coseno

Dado un triángulo ABC, siendo α, β, γ, los ángulos, y a, b, c, los lados respectivamente opuestos a estos ángulos, entonces (Ver figura 2.6):

$$c^2 = a^2 + b^2 - 2ab \cdot \cos\alpha \tag{2.10}$$

2.2.11. Resolución de triángulos

Para resolver un triángulo ABC, siendo α, β, γ, los ángulos, y a, b, c, los lados respectivamente opuestos, entonces (Ver figura 2.6):

En primer lugar, y para el problema planteado, se tienen los siguientes datos:

Distancia del segmento AB, la cual es fija.

Distancia del segmento BC, la cual es fija.

Distancia del segmento AC, la cual es variable.

Al conocer el valor de los lados del triángulo se puede obtener el valor de los ángulos aplicando el teorema del coseno (Ver apartado 2.2.10). Por consiguiente obtendremos los valores de α y β aplicando las siguientes ecuaciones.

$$\alpha = \arccos\left((a^2 + b^2 - c^2)/2ab\right) \tag{2.11}$$

$$\beta = \arccos\left((b^2 + c^2 - a^2)/2bc\right) \tag{2.12}$$

2.3. Modelado 3D

Actualmente existen diferentes herramientas para crear elementos 3D y distintas formas de creación. Entre ellos se diferencian dos tipos de técnicas de modelado, modelado procedural y mediante polígonos.

La dificultad de la creación de elementos 3D depende de la correcta elección de herramientas *software* que faciliten su creación. Para el avatar propuesto se han utilizado varios programas con interacción entre ellos. Estos programas han sido *3Ds Max* y *Solidworks* permitiendo *3Ds Max* diseño mediante polígonos y *Solidworks* paramétrico, permitiendo el uso de planos para crear objetos 3D.

Además se utilizan técnicas de animación accesibles gracias a estas aplicaciones, como es por ejemplo el uso de *bones*, modificación de mallas y jerarquías de objetos. [1]

2.4. Cinemática

Como se expuso en el apartado *Necesidades*, existen dos grandes grupos relacionados con la cinemática que a su vez se dividen en diversas técnicas (Ver apartado 1.2): [2] [3]

- Cinemática inversa
  - Métodos algebraicos: Solo pueden ser utilizados en casos concretos donde es posible expresar el valor de los ángulos de las articulaciones mediante ecuaciones.
  - Métodos iterativos: La solución del problema de la cinemática inversa se soluciona mediante pasos que incrementalmente mejoran la solución de los ángulos de las articulaciones.
- Cinemática directa.

En este caso y debido al objetivo del TFG, el proyecto se centrará en la cinemática inversa. Además, dentro de los subgrupos de cinemática existen multitud de métodos pero este TFG se centrará en un método algebraico explicado en el capítulo *Desarrollo* (Ver apartado 3).

2.4.1. Problemática de la cinemática inversa

En primer lugar debe ser definido el que se conoce como "el problema de la cinemática inversa". La definición de la problemática según algunos autores es básicamente, determinar los valores de las juntas, las cuales se calculan su posición a partir de un objetivo. [2][9]

Aunque se piensa que todo en el campo de la cinemática está estudiado, en realidad es un campo en constante evolución y con infinidad de aplicaciones. Algunas aplicaciones son: [4]

- Animaciones virtuales.
- Robótica.
- Medicina.

Además la IK está dando lugar a nuevas tecnologías, como por ejemplo nuevos robots. IK da la posibilidad de escalar los robots tanto a nivel micro como macro.

2.5. Técnicas de animación

En la actualidad existen diversas técnicas que permiten animar modelos 3D. Algunas de estas técnicas son: [1]

- Estructura jerárquica articulada de objetos 3D: Se define una estructura jerárquica del cuerpo como si fuesen objetos separados. Los objetos son almacenados en la jerarquía y se unen entre sí mediante sus *pivots*.

- Mezcla de mallas entre personajes: Esta técnica se basa en usar más de un modelo que representen al mismo objeto, mostrándolos en poses diferentes. Todos los objetos deben estar formados por la misma malla ya que la animación se realizará a partir de la interpolación de los vértices con una posición inicial, tomada de uno de los modelos y como final la de otro de los modelos con pose diferente.

- Animación del esqueleto y mallado de la piel: Se basa en la creación de un endoesqueleto que es una estructura jerárquica de articulaciones, que además tendrá un mallado de vértices representando la silueta del objeto. El movimiento del endoesqueleto modificará la malla del objeto original afectando a los vértices.

- Cinemática inversa en tiempo real: Se basa en el cálculo automático de la posición que deben tomar las diferentes articulaciones del objeto (Ver apartado 2.4).

- Captura de movimientos: Consiste en la extracción de movimientos a partir de actores reales. La captura de movimientos se suele usar junto con la cinemática inversa, la cual permite obtener los ángulos formados por la pose tomada por un actor real.

Capítulo 3

Desarrollo

3.1. Modelado

3.1.1. Planos diseñados

Antes de modelar el Robot_Ayuda se realizó una fase de diseño del mismo. El diseño fue costoso debido a que se quería conseguir dotar de ciertas características al componente a crear. Los principales requisitos iniciales fueron: [8]

- El modelo debería ser geométricamente exacto.
- Debería ser fácil aplicar pequeñas modificaciones al modelo.
- El cuerpo del modelo debería ser similar a una esfera y dar sensación de arácnido.
- El modelo debería tener la habilidad de plegarse y desplegarse.
- Una vez plegado, debería formar una esfera perfecta.
- Las distintas partes que forman el robot deberían encajar de forma perfecta.
- Minimizar costes si se decidiese realizar una impresión 3D.

A partir de los requisitos expuestos, se realizaron diversos diseños, los cuales no cumplían dichos requisitos. Una vez conseguido el diseño de un modelo que cumpliese los requisitos se comenzó la creación del modelo. Dado que se habían creado croquis con las partes que formarían el Robot_Ayuda, se pensó que el programa ideal para llevar a cabo el diseño sería *Solidworks*.

Para construir el componente fueron necesarios cuatro planos.

1. Seccion-1_Y: La pieza está construida de forma que su conexión con la Esfera_Robot sea una sección de esfera y pueda alojar la Seccion-1_Y correspondiente (Ver figura 1.6).
2. Seccion-2_Y: La pieza está construida de forma que su conexión con la Seccion-1_Y correspondiente sea una sección de cilindro, y la Seccion-2_Y cuando esté plegada formará la sección de superficie correspondiente de la esfera (Ver figura 1.7).

3. Proyector: La parte superior del proyector es como la superficie de la Esfera_Robot, su zona intermedia es fina evitando el rozamiento o choque con el hueco donde está alojado, y en la parte inferior hay peana que funcionara como tope en el movimiento del proyector. Además, se diseñó con el objetivo de proyectar, alumbrar o futuras posibilidades a diseñar (Ver figura 1.4).

4. Esfera_Robot: La Esfera_Robot tiene huecos donde irán encajadas las Seccion-1_Y y el Proyector. Cuando todos los elementos del Robot_Ayuda estén cerrados el Robot_Ayuda forma una esfera perfecta[1] (Ver figura 1.3).

3.1.2. Modelado en Solidworks

Solidworks es una herramienta *software* desarrollada por *Solidworks Corp.*, una subsidiaria de *Dassault Systèmes*. Permite modelar de manera paramétrica. *Solidworks* modela como *AutoCAD* a partir de planos 2D. Pero se diferencia en la fácil y rápida consecución de resultados de cuerpos sólidos respecto a *AutoCAD*, siendo muy útil para la creación de piezas, herramientas o conjuntos de piezas, como por ejemplo un robot. Es un programa muy potente que permite infinidad de oportunidades a la hora de crear elementos fieles a la realidad, debido a que se generan a partir de planos. [8]

Solidworks da la posibilidad de generar objetos 3D a partir de los planos de las piezas. La creación de las partes del Robot_Ayuda se crearon con su uso, pero los planos que estaban diseñados no eran válidos a la hora de crear los objetos. El problema surgía de particularidades de uso de *Solidworks* y por tanto, se tuvo que crear planos auxiliares o elementos de corte para construir todas las piezas diseñadas al principio.

Además, otra de las herramientas que proporciona *Solidworks* es el ensamblaje de piezas que al unirse forman un objeto final, y las piezas, siguen manteniendo su identidad, es decir, las agrupa y no las fusiona. Como consecuencia de la exactitud geométrica de los planos y la posibilidad que proporciona *Solidworks*, el ensamblaje de piezas pudo hacerse con dicho programa.

En otro orden, *Solidworks* aparte de permitir modelar, permite obtener los planos incorporarlos para utilizarlos en documentación, como se puede observar en este documento (Ver figuras 1.2,1.3, 1.6, 1.7, 1.4 y 1.5). Pero la potencia de *Solidworks* no acaba solo en la generación de planos y modelos planos. Este programa permite realizar cambios en los planos y que esto afecte al modelo que se crea a partir de él, es decir, podemos crear cambios en los objetos de forma rápida, sencilla y exacta.

El problema que apareció en el uso de Solidworks fue de compatibilidad y de situación de los *pivot* de los componentes. La situación de los *pivots* no era la ideal para conseguir el sistema de cinemática inversa y la compatibilidad evitaba su exportación directa a la plataforma *Unity*. Para solventar el problema se utilizó una herramienta *software* intermedia denominada 3Ds Max, cuya función se explicará en el siguiente apartado (Ver apartado 3.1.3).

Por último mencionar que la licencia utilizada es del tipo "estudiante".

[1]Salvo errores de visualización

3.1.3. Modelado en *3ds Max*

3Ds Max es un programa de *Autodesk* cuya función es modelar, animar y renderizar en 3D. Esta herramienta *software* proporciona facilidades a la hora de modelar abstrayendo algoritmos y herramientas para facilitar su uso. [8]

En un principio se creyó que el modelado en *3Ds Max* iba a ser mínimo y se centraría en hacer compatibles las extensiones soportadas en *Unity* con las extraídas de *Solidworks*. La realidad por el contrario ha demostrado que se ha dado diversos usos a la herramienta ya mencionada. Sus usos han sido:

- Transformar del formato *.STL* procedente del programa *Solidworks* al formato *.FBX* soportado por la plataforma *Unity3D*. La realización de este punto es simple y sencilla ya que simplemente hay que introducir el modelo *.STL* en *3Ds Max* y exportarlo con el nuevo formato deseado teniendo en cuenta la escala.

- Recolocación y adaptación de los ejes de coordenadas de los elementos 3D que forman el Robot_Ayuda. Para colocar de nuevo los *pivots* se utilizó opciones que proporciona *3Ds Max* de alineamiento de los mismos. Además, se diseño un *script* en Max *Script*, que está basado en *Visual Basic Script*, que realizase una adaptación del eje de coordenadas de mano derecha, usado por *3Ds Max*, a coordenadas de mano izquierda, utilizada por *Unity*. La funcionalidad del *script* rota el *pivot* $90\,^{\circ}$ sobre el eje X que da lugar a que el eje Z funcione de forma inversa (Ver figura 3.1). También se tuvo que reposicionar los centros de los elementos teniendo en cuenta sus futuros movimientos.

- Creación de elementos 3D para diferentes escenarios. Fue necesario crear diversos objetos 3D para realizar pruebas, para ello se utilizó *3Ds Max*, por ejemplo se crearon vallas, rocas, puertas, etc.

Por último mencionar que la licencia utilizada es del tipo “estudiante”.

3.2. Cinemática inversa

3.2.1. Definición del método

El método seleccionado o elegido ha sido un método geométrico o analítico, dependiendo de la clasificación usada. El método fue seleccionado por las razones que se exponen a continuación:

1. Los métodos iterativos fueron descartados ya que no se ajustan bien a menos de seis grados de libertad y dado que el Robot_Ayuda solo tiene tres grados de libertad por cada Union_Y, no es aconsejable utilizar estos métodos.

2. Los métodos más complejos, como redes neuronales, son más lentos, y su compleja implementación ha conducido a seleccionar un método geométrico para la creación del sistema de cinemática inversa.

3. Por el contrario, el método geométrico es más rápido y sencillo de implementar para pocos grados de libertad.

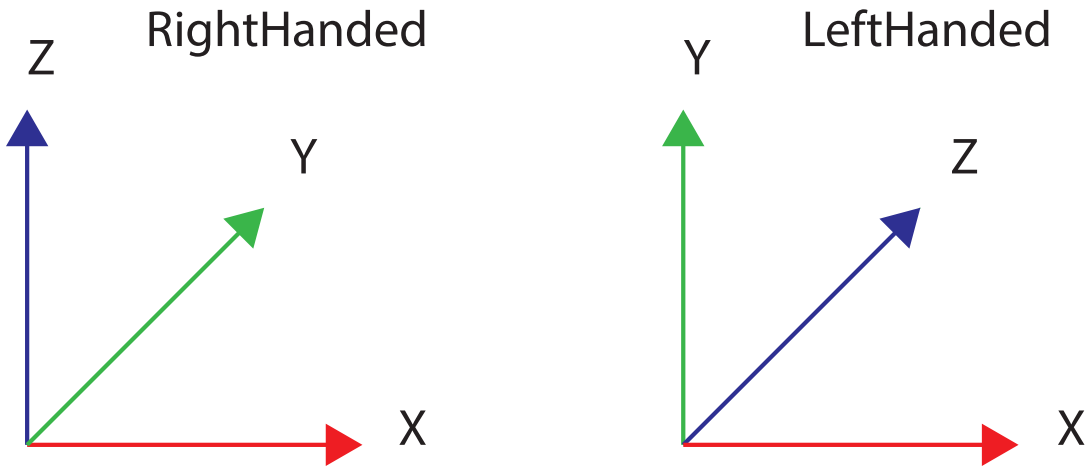


Figura 3.1: Muestra el cambio realizado del eje de coordenadas utilizado por *3Ds Max* a *Unity3D* aplicando una rotación sobre el eje X de 90 °.

4. Las extremidades se mueven formando siempre un triángulo entre las secciones 1 y 2.

5. Gran parte de los conceptos matemáticos ya están implementados y proporcionados por las bibliotecas de *Unity*.

La solución de los ángulos implicados en los movimientos de las extremidades se basa en la resolución del triángulo que se forma entre las secciones Seccion-1_Y y Seccion-2_Y y el punto objetivo (Ver figura 3.4). De este modo se consigue solucionar dos de los tres grados de libertad. El grado de libertad restante, se soluciona mediante el ángulo formado entre dos vectores y aplicando localmente la solución relativa al ángulo actual (Ver figura 3.6). También hay que considerar el caso de los puntos que no entran dentro del rango de alcance del triángulo. En estos casos se usa la prolongación del vector dirección origen-destino y se busca la solución más alejada en la recta producida a partir del mencionado vector dirección.

3.2.2. Implementación

Esqueleto lógico

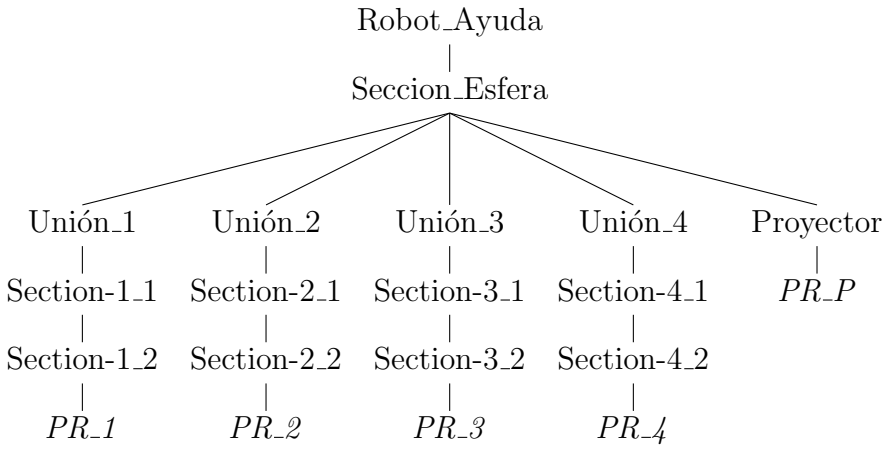
En un principio se pensó en utilizar un endoesqueleto para controlar al Robot_Ayuda y más adelante implementar la cinemática inversa. Sin embargo, durante el planteamiento del problema, se observó que utilizar una jerarquía de objetos 3D se ajustaba mejor a las necesidades. Las razones que llevaron a elegir esta técnica fueron:

1. El robot está fabricado a partir de distintos submodelos o piezas que encajan. Una vez encajadas todas las piezas se obtiene el robot final.

2. El modelo es sólido, y por tanto, el uso de un endoesqueleto no sería la mejor opción ya que surgen problemas de visualización cuando se utiliza en objetos 3D sólidos.

3. Debido al punto anterior, no habrá repartición de pesos para controlar los polígonos, es decir, funcionarán como bloques y tampoco debe sufrir alteraciones en la malla.

4. La propia construcción del modelo tiene una jerarquía de objetos 3D. La jerarquía formada es:



A partir de los puntos anteriores se tomaron decisiones de diseño expuestas a continuación:

- Se creará un esqueleto lógico que afecta a las patas y toman como centro la esfera.

- El esqueleto lógico no afectará al proyector, que funcionará como un elemento independiente.

- Las secciones que componen el esqueleto lógico surgen a partir de la relación entre los *pivots* de los objetos 3D que forman la Seccion_Esfera. En otras palabras, las secciones serán cada una de las ramas del árbol de jerarquía, y para formarlas, se utilizarán los *pivots* de los nodos que están conectados por la rama.

- A partir del punto anterior, surge la necesidad de crear objetos de referencia que actuarán como el punto final del segmento. Estos objetos auxiliares serán totalmente invisibles al usuario final.

El esqueleto lógico será necesario por la utilización de un método geométrico que será expuesto más adelante.

Restricciones en cinemática inversa de Union-X

Como se ha mencionado con anterioridad, la cinemática inversa de las patas o miembros del Robot_Ayuda se basa en un método geométrico o analítico dependiendo de las distintas clasificaciones. Sin embargo, el primer tema a tratar es el uso de las restricciones.

En el sistema de cinemática se deben usar distintas restricciones, dependiendo del número de soluciones posibles y del modelo desarrollado (posibilidades de movimiento del modelo por ejemplo). En el caso que concierne a este TFG, el número de soluciones posibles son dos por cada cálculo de posición en el plano XY del espacio XYZ (Ver figura 3.3). Obtenida la solución se restringe mediante la aplicación de los ángulos α y β del triángulo que se crea entre las secciones Seccion-1_Y y Seccion-2_Y y el vector $\vec{OD}$ (Ver figuras 2.6 y 3.4). En otras palabras, los ángulos que se calculan, α y β, no son la solución final a aplicar a cada Seccion-X_Y, sino que es una solución relativa, ya que hay que aplicar los ángulos obtenidos a los actuales e introducirlos en el eje de coordenadas local correspondiente (Ver figuras 5.8 y 3.2). Entonces, dependiendo de cómo se apliquen, se obtendrá una solución u otra. Para este caso en concreto los ángulos finales aplicados son (Ver figura 3.2):

- Referente al ángulo α: El ángulo final a aplicar se obtendrá de la ecuación:

$$\alpha_f = \alpha_r - \gamma + \theta^{**} \tag{3.1}$$

- Referente al ángulo β: El ángulo final a aplicar se obtendrá de la ecuación:

$$\beta_f = \beta_r - \theta - \omega^{***} \tag{3.2}$$

Por otro lado, en el caso de la restricción dependiente del modelo, se realizó una aproximación de los ángulos de movimiento permitidos. Como los ángulos permitidos tienen dependencias entre sí (β y δ son dependientes de α, este a su vez es dependiente del modelo), se ha aproximado mediante curvas. Las curvas utilizadas son curvas diseñadas para crear animaciones pero que inherentemente pueden ser utilizadas para otra finalidad, interpolación, aproximación, etc. De este modo, se puede definir restricciones en ángulos a partir de relaciones de ángulos.

De todos modos las restricciones dependientes del modelo se han implementado pero no se han introducido. La decisión de no introducirlas se debe a que el modelo diseñado en un inicio es excesivamente restrictivo, dando lugar a movimientos muy poco dinámicos y poco naturales. Se pensó en modificar el modelo del Robot_Ayuda, pero existen diversas razones que han conducido a evitar introducirlas. Las razones son:

- La licencia utilizada para el desarrollo del modelo ha sido del tipo "estudiante" con duración de un mes. Al expirar la licencia y dado el excesivo precio de la renovación de la misma se decidió mantener el modelo actual.

- Se dio prioridad a la libertad de movimientos sobre los posibles problemas existentes al no restringir los movimientos.

[**] Ángulo de compensación de inclinación. Ver anexo 5.2.

[***] Ángulo de compensación de inclinación. Ver anexo 5.2.

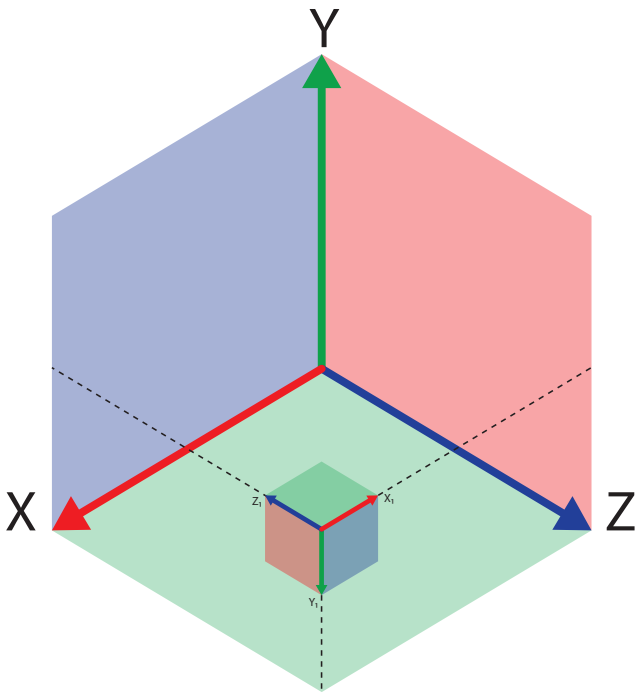


Figura 3.2: Ejemplo de un espacio local contenido dentro del espacio global.

- El problema se detectó en un momento avanzado del desarrollo del TFG.

Por consiguiente, y basándose en las razones expuestas, se decidió que en un futuro se solventaría el problema de las restricciones por modelo.

Cinemática inversa de Union-X

Como ya se ha mencionado, el método utilizado es un método geométrico. El método se basa en la resolución de triángulos y división del espacio 3D en planos 2D, por un lado el plano XY y por el otro lado XZ, dando lugar a que las rotaciones se aplicarán sobre los ejes Z e Y respectivamente. El sistema de cinemática afecta solo a los ángulos de los ejes Z e Y en coordenadas locales y sin modificar las posiciones locales de los *pivot*. Trabajar en coordenadas locales ha sido un punto importante en el desarrollo del sistema, ya que gracias a ello, el movimiento del Robot_Ayuda dentro del entorno virtual no afectará a los cálculos que se realizan (por ejemplo, rotar o mover el Robot_Ayuda durante la ejecución).

En cuanto al desarrollo de la cinemática, se pueden diferenciar tres grandes fases obviando las restricciones por el modelo (Ver apartado 3.2.2). Por lo tanto las fases serán:

1. Cálculo de los ángulos sobre los ejes locales Z: Los ángulos sobre los ejes locales Z se solucionan proyectando el esqueleto lógico del Robot_Ayuda y el punto objetivo sobre el plano XY. En el plano XZ existen dos grados de libertad que son las rotaciones sobre los ejes Z de los objetos 3D Seccion-1_Y y Seccion-2_Y, cuyos

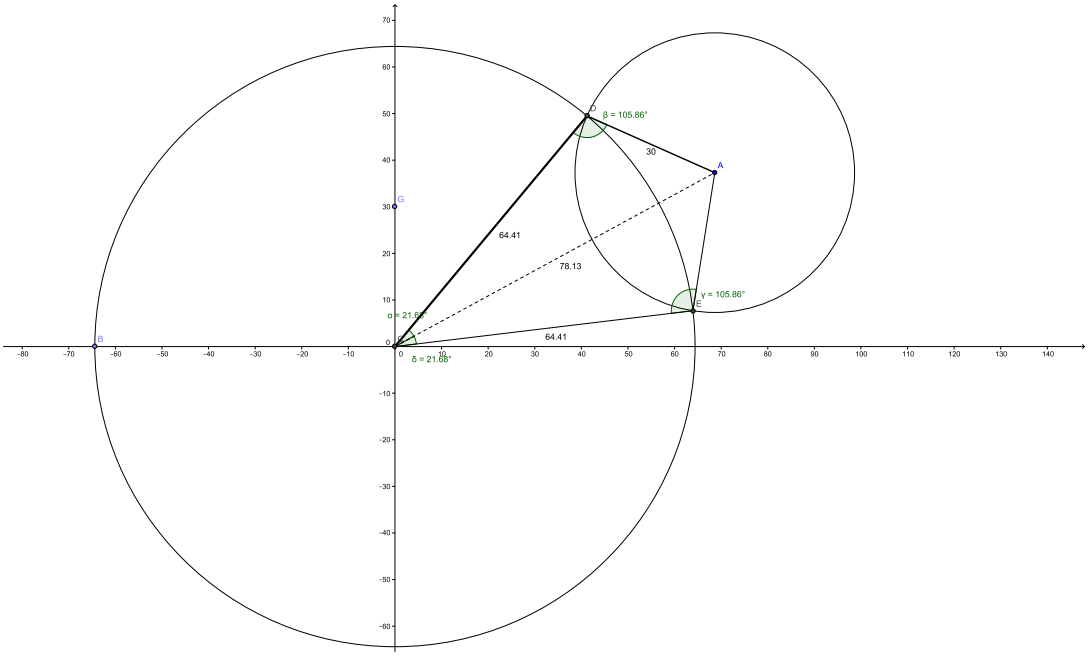


Figura 3.3: Muestra las soluciones posibles para cada punto final.

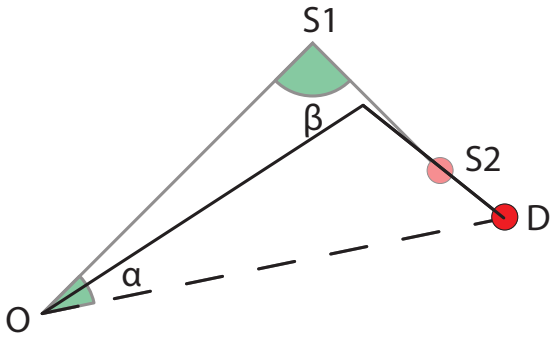


Figura 3.4: Muestra cómo se forma el triángulo a solucionar entre la Seccion-1_Y ($\vec{OS1}$), Seccion-2_Y ($\vec{S1S2}$) y el vector $\vec{OD}$ en el plano XY.

respectivos ángulos se han denominado α y β respectivamente. El triángulo formado entre la Seccion-1_Y, Seccion-2_Y y el vector variable $\vec{OD}$, puede ser resuelto mediante geometría y así solventar dos de los tres grados de libertad (Ver figuras 2.6, 3.4 y 3.5).

A partir del triángulo formado se obtienen los siguientes datos:

- Posición actual del TG-Y.
- Vectores $\vec{OS}1$, $\vec{S1S}2$ y $\vec{OD}$.
- Longitud de los vectores $\vec{OS}1$, $\vec{S1S}2$ y $\vec{OD}$.

Con los datos expuestos es posible solucionar cualquier triángulo aplicando el teorema del coseno y relaciones geométricas (Ver apartados 2.2.10 y 2.2.11). Sin embargo, existe un grave problema que se produce comúnmente: el caso de que la lejanía del TG-Y provoque que no exista el triángulo planteado. Para dar solución al problema se busca el punto alcanzable más lejano a partir de la siguiente fórmula:

$$P(X,Y,Z) = Pivot_{Seccion-1\_Y} + \vec{OD} * MaxDist \tag{3.3}$$

Donde $Pivot_{Seccion-1\_Y}$ será la posición del *pivot* de la Seccion-1_Y, $\vec{OD}$ el vector formado entre los puntos origen-destino normalizado y $MaxDist$ que será la distancia máxima alcanzable por la Union_X (Ver apartado 2.2.3).

Aplicando la fórmula anterior se obtiene un punto sobre la recta que permite formar un triángulo y por tanto solucionarlo mediante el método geométrico expuesto anteriormente.

2. Cálculo del ángulo sobre el eje Y: El ángulo a obtener se formará al proyectar el esqueleto del Robot_Ayuda del espacio 3D sobre el plano ZX. En este caso, únicamente existirá un grado de libertad, el cual pertenece a los objetos del tipo Seccion-1_Y, ya que los objetos Seccion-2_Y no dispondrán del grado de libertad de rotar sobre su eje local Y. Por consiguiente, el problema planteado será solucionar el ángulo formado entre dos vectores del plano XZ. La solución del sistema se lleva a cabo utilizando operaciones vectoriales, por un lado el ángulo entre vectores, por otro el producto vectorial y por último aplicando la regla de la mano izquierda o del sacacorchos (Ver apartado 2.2.5 y 2.2.7, y figura 3.6). Sin embargo, el ángulo que se calcula es relativo a la rotación actual de la Seccion-1_Y. Por tanto, una vez obtenido dicho ángulo y el producto vectorial entre ambos vectores se aplicará la regla de la mano izquierda o del sacacorchos para conocer el signo del ángulo calculado, dando lugar a dos ecuaciones que permiten aplicar el ángulo final δ sobre el eje Y. Las ecuaciones serán:

   Sea $\vec{ProdVectorial} = (x,y,z)$ donde $\vec{ProdVectorial} = \vec{OS}1 \times \vec{OD}$. Si $z >= 0$, entonces

$$\delta_r = \delta_r \tag{3.4}$$

   en caso contrario

$$\delta_r = -\delta_r \tag{3.5}$$

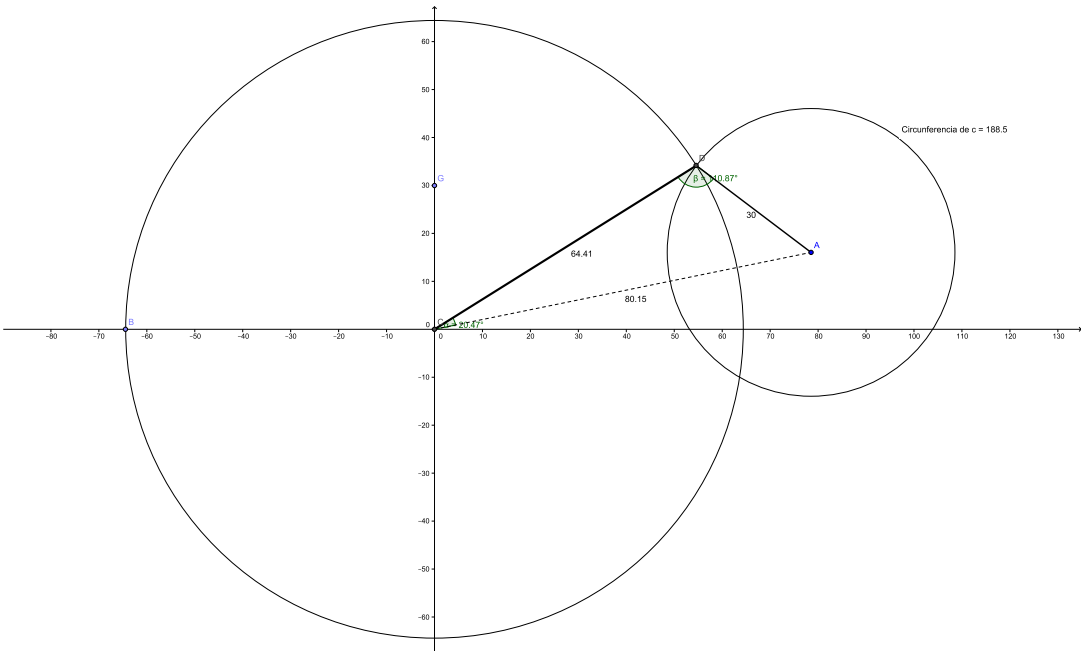


Figura 3.5: Triángulo a solucionar que se forma a partir del esqueleto lógico del Robot_Ayuda y el vector dinámico $\vec{OD}$.

Y por tanto el ángulo final será:

$$\delta_t = \delta_t + \delta_r \tag{3.6}$$

3. Aplicación de ángulos sobre los ejes Z e Y: La última fase es la aplicación de los ángulos obtenidos, es decir, después de obtener los ángulos α y β de la primera fase y el ángulo δ de la segunda fase. Simplemente habrá que aplicar los ángulos de la siguiente forma:

   - α será aplicado sobre el eje de rotación Z del objeto Seccion-1_Y.
   - β será aplicado sobre el eje de rotación Z del objeto Seccion-2_Y.
   - δ será aplicado sobre el eje de rotación Y del objeto Seccion-1_Y.

Además existen varias condiciones que hay que cumplir para el correcto funcionamiento del sistema de cinemática inversa.

1. Cualquier cálculo se debe hacer en coordenadas locales.
2. Cualquier aplicación de rotaciones se hará sobre coordenadas locales.
3. Se debe mantener la lógica entre el paso de coordenadas globales y locales.
4. En última instancia se debe ejecutar cualquier operación relacionada con la cinemática inversa.

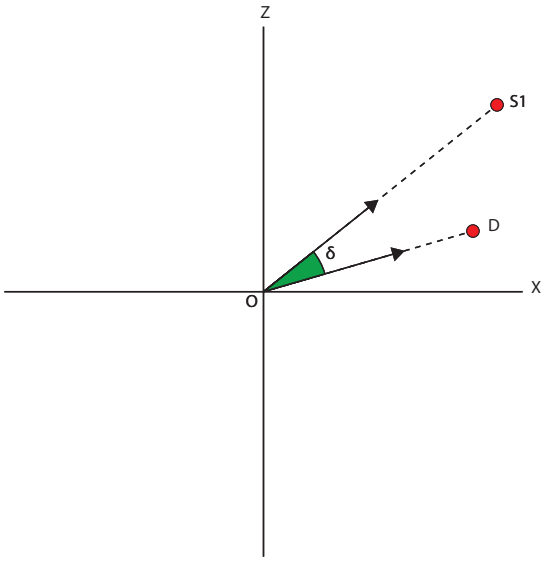


Figura 3.6: Se muestra el ángulo que forman dos vectores, el vector $\vec{OD}$ y el vector de dirección (Puede ser del elemento Proyector o Seccion-1_Y).

En el diagrama de flujo de la ejecución del sistema de cinemática inversa de Union-Y se puede observar que en el primer instante de ejecución se iniciarán ciertas variables globales que se mantendrán estáticas durante el tiempo que dure la ejecución del entorno virtual (Ver figura 3.7) . Una vez inicializadas, se evaluará el modo de ejecución en el que nos encontramos, si el modo es igual a *false* entonces no se realizará ningún cálculo ni movimiento de las diferentes secciones que componen la jerarquía de Union-Y. Por el contrario, se continuará actualizando los vectores que forman el esqueleto lógico ya que dependen de la posición actual de las distintas secciones. Obtenidos los vectores del esqueleto lógico habrá que calcular las longitudes de los distintos vectores para obtener los datos necesarios para realizar la solución geométrica, como se ha explicado anteriormente. En este punto se realiza una bifurcación del hilo de ejecución, por un lado si el punto es alcanzable no se harán operaciones añadidas, o por el contrario el punto es inalcanzable habrá que realizar una búsqueda de un punto alcanzable como se ha explicado anteriormente. Terminada la operación de búsqueda o en el caso de no ser necesaria, ambas posibilidades volverán a unirse para realizar la resolución geométrica del problema planteado. En este punto las fases mencionadas anteriormente se ejecutan. En primer lugar la primera fase (Cálculo de los ángulos sobre los ejes locales Z), seguida de la segunda fase (Cálculo del ángulo sobre el eje Y) y por último, la tercera fase (Aplicación de ángulos sobre los ejes Z y el eje Y). Una vez ejecutadas las tres fases, se volverá al momento de ejecución del modo en el que nos encontramos, volviendo a realizar toda la realización del flujo de ejecución.

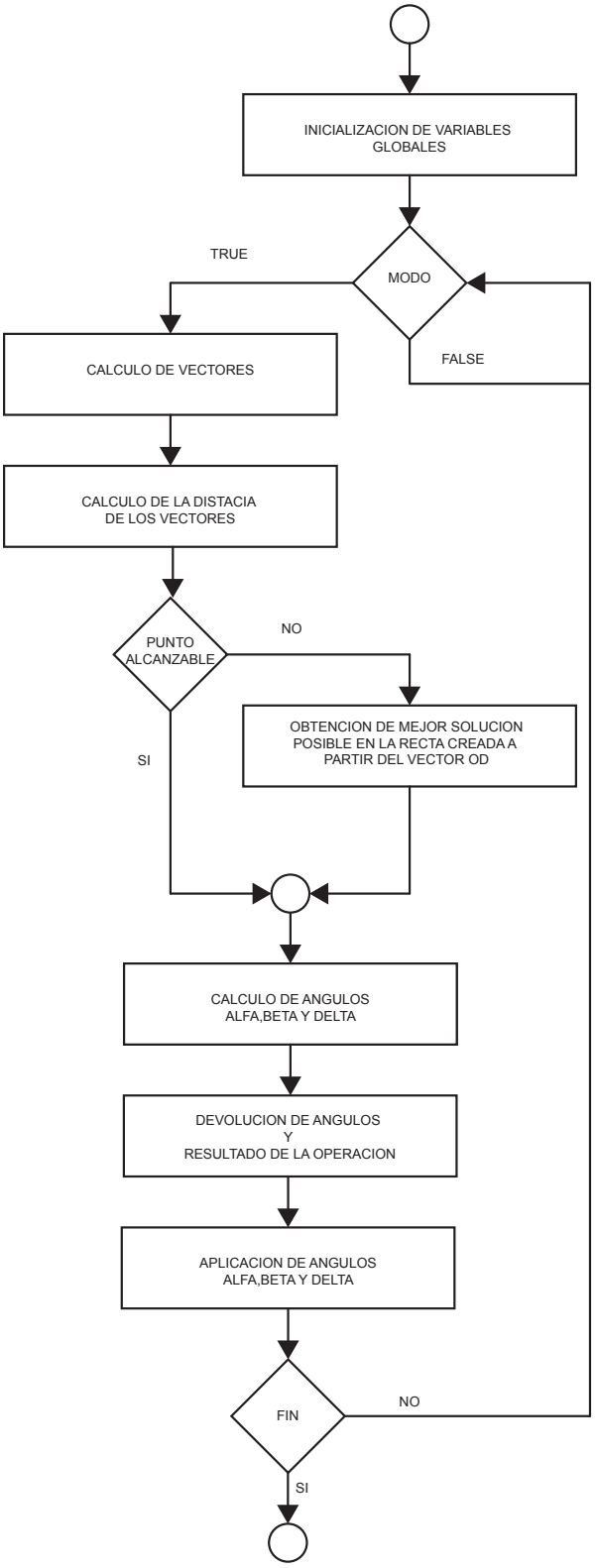


Figura 3.7: Diagrama de flujo de ejecución del sistema de cinemática inversa de los componentes Union-Y.

Restricciones en la cinemática inversa del proyector

El objeto proyector que forma parte del componente Robot_Ayuda tiene dos tipos de restricciones de movimientos. En primer lugar y al igual que en caso de la cinemática inversa de las Union-Y, tiene una restricción que se debe al modelo y prácticamente todas las restricciones surgen del cuerpo del Robot_Ayuda. Sin embargo, para dar mayor flexibilidad al proyector se han diseñado tres tipos de restricciones que dependen de la lógica y se apoyan en las restricciones del modelo. Las tres restricciones inferidas son:

1. Totalmente abierto: En este caso el proyector se mantendría siempre en su máxima altura. Es decir, el proyector deberá mantener su máxima altura local y para que se mantenga visible el proyector.

2. Totalmente cerrado: Al contrario que en el caso anterior, en este caso se mantendría la apertura mínima permitida al proyector, dando lugar a una adaptación de la zona superior del proyector a la esfera que formaría la Esfera_Robot si no hubiese sido cortada.

3. Altura variable: En este caso será cuando el sistema de cinemática inversa calcule la altura del proyector. La cinemática se encargará de buscar la altura óptima para un objetivo dado y se intentará mantener mientras las restricciones por modelo no la limite.

Las restricciones dependientes del modelo producen que la rotación sobre los ejes X y Z del espacio XYZ estén bloqueadas, permitiendo solamente la rotación sobre el eje Y. Además, y a diferencia de la cinemática de los miembros, la cinemática del proyector podrá modificar la posición del proyector sobre su eje Y, ya sea aumentándola o reduciéndola.

Cinemática inversa del Proyector

En lo referente a la cinemática inversa del componente Proyector se ha utilizado un medio procedente de la realización de la cinemática inversa de las Union-Y y otro perteneciente exclusivamente a la cinemática del proyector. Además, y al igual que en apartado de cinemática de los objetos Union-Y, existen tres grandes fases en la resolución del problema de cinemática inversa del proyector. Por tanto, las tres fases existentes son:

1. Cálculo de la posición en el eje Y: En todo el sistema de cinemática desarrollado en el TFG , es la única vez donde el cálculo no se realiza en coordenadas locales, aunque el movimiento se aplique localmente Tampoco se trabaja con ángulos y en cambio se trabaja con el posicionamiento. Para calcular la altura deseada, se utiliza el PR_P como baliza de señalización de la altura a la que se encuentra el proyector. Por tanto para calcular el movimiento relativo necesario para obtener la mejor posición se ha realizado resolviendo la siguiente ecuación:

$$Movimiento(x, y, z) = TG\_P_{pos} - PR\_P_{pos} \tag{3.7}$$

   Donde $yPosition$ será la $y \in Movimiento$, TG_P_{pos} será la posición objetivo y PR_P_{pos} el punto de referencia del proyector.

El cálculo en coordenadas globales se debe a que ambas deben estar en el sistema de coordenadas y éste debe ser estático. Como consecuencia se ha utilizado coordenadas globales y una vez obtenido el valor se asigna localmente para evitar errores de coherencia.

2. Cálculo del ángulo de rotación sobre el eje Y: El ángulo de rotación sobre el eje Y será el equivalente a buscar la dirección a la que apuntará el proyector. Para ello y al igual que en la segunda fase de la cinemática inversa de las Union-Y el cálculo del ángulo a aplicar se basa en proyectar los vectores $\vec{OD}$ y $\vec{OPR\_P}$ sobre el plano XZ. A continuación, se utilizarán los conceptos matemáticos del cálculo de ángulo entre vectores, producto vectorial y regla de la mano izquierda o sacacorchos (Ver apartados 2.2.5, 2.2.7 y 2.2.8). Por tanto, el ángulo a aplicar, δ, se solucionará según el siguiente método:

   Sea $\vec{ProdVectorial} = (x, y, z)$ donde $\vec{ProdVectorial} = \vec{OPR\_P} \times \vec{OD}$. Si $z >= 0$, entonces

   $$\delta_r = \delta_r \tag{3.8}$$

   en caso contrario

   $$\delta_r = -\delta_r \tag{3.9}$$

   Y por tanto el ángulo final será:

   $$\delta_t = \delta_t + \delta_r \tag{3.10}$$

3. Aplicación de la posición en el eje Y y ángulo en el eje Y: Una vez finalizadas las fases *Cálculo de la posición en el eje Y* y *Cálculo del ángulo de rotación sobre el eje Y* se pasaría a aplicar los resultados obtenidos siendo:

   - *yPosition* será aplicado sobre el eje de posición Y del proyector.
   - δ será aplicado sobre el eje de rotación Y del objeto Proyector.

Además existen varias condiciones que hay que cumplir para el correcto funcionamiento del sistema de cinemática inversa.

- La aplicación de rotaciones o movimientos debe ser local.
- La altura debe ser obtenida en el eje de coordenadas global.
- El cálculo del ángulo δ debe realizarse en el eje de coordenadas local.
- La rotación del objeto Proyector debe estar siempre activa.
- Pueden existir distintos modos de apertura.

El diagrama de flujo de la ejecución del sistema de cinemática inversa del proyector, comenzará iniciando las variables globales (Ver figura 3.8). Una vez iniciadas, se actualizan los vectores necesarios para realizar los cálculos expuestos anteriormente y se realiza el cálculo de la posición a aplicar sobre el eje local Y. A partir de este punto se continúa con la evaluación del modo en el que se encuentra el proyector, existiendo tres posibilidades.

1. Modo = 0 : En este caso, el Proyector es obligado a mantener su altura al mínimo, es decir, cerrado y, por tanto, formando una esfera con la Esfera_Robot.

2. Modo = 1: En este caso, el proyector toma la altura calculada.

3. Modo = 2: En este caso, el Proyector es obligado a mantener su altura máxima.

Más adelante y ya evaluado el modo y ejecutada la rama correspondiente, todas las ramas volverán a unirse en el mismo hilo para realizar el cálculo del ángulo δ, en otras palabras, la realización de la segunda fase. Finalmente, se aplicará tanto la altura obtenida del proceso como el ángulo y una vez aplicado se volverá a la fase de actualización de vectores. El sistema de cinemática terminará cuando concluya la ejecución del entorno virtual.

3.2.3. Problemas encontrados

A lo largo del desarrollo de sendos sistemas de cinemática, han aparecido diversos errores, los cuales han provocado su detección, estudio, planteamiento de solución, aplicación de la solución y volver a chequear el error para comprobar que ha sido solucionado. En caso negativo fue repetido el proceso de resolución de errores y en caso positivo, se ha continuado con las siguientes fases del proyecto. Además cabe destacar que gran parte de los errores han provocado una instrucción en campos teóricos no pensados al comienzo del TFG.

Los principales problemas encontrados en relación con ambos sistemas de cinemática inversa han sido:

- Elección del método entre los estudiados: El primer problema afrontado fue la elección de un método, ya que como se ha expuesto en apartados anteriores existen multitud y diversidad de métodos (Ver apartado 2.4).

  Los medios utilizados para solventar el problema de selección fue comparar las necesidades, requisitos y recomendaciones propuestas en las distintas documentaciones existentes y seleccionar el método que más se adapta o ajusta a las necesidades. Se tuvo en gran consideración la dificultad de los métodos y principalmente su funcionalidad.

- Errores de planteamiento: En ocasiones han sido planteadas soluciones más complejas de lo necesario o con ciertos errores de realización. Por ejemplo, usar un método iterativo en contra del geométrico o un mal uso de las bibliotecas proporcionadas por *Unity*[4].

  La solución aplicada consistió en la reconsideración del método tras estudiar la cinemática de las Union-Y. En cuanto al uso de bibliotecas o conceptos matemáticos, la resolución consistió en una mayor la formación en los campos teóricos necesarios.

- Uso de coordenadas locales contra globales: Aunque este error se podría incluir en el punto anterior, en realidad ha sido planteado como un nuevo punto debido a que el error planteado produjo muchos problemas a la hora de conseguir finalizar la cinemática inversa de ambos módulos. El problema, aunque en un principio es

[4]Ver `http://docs.unity3d.com/Documentation/ScriptReference/`

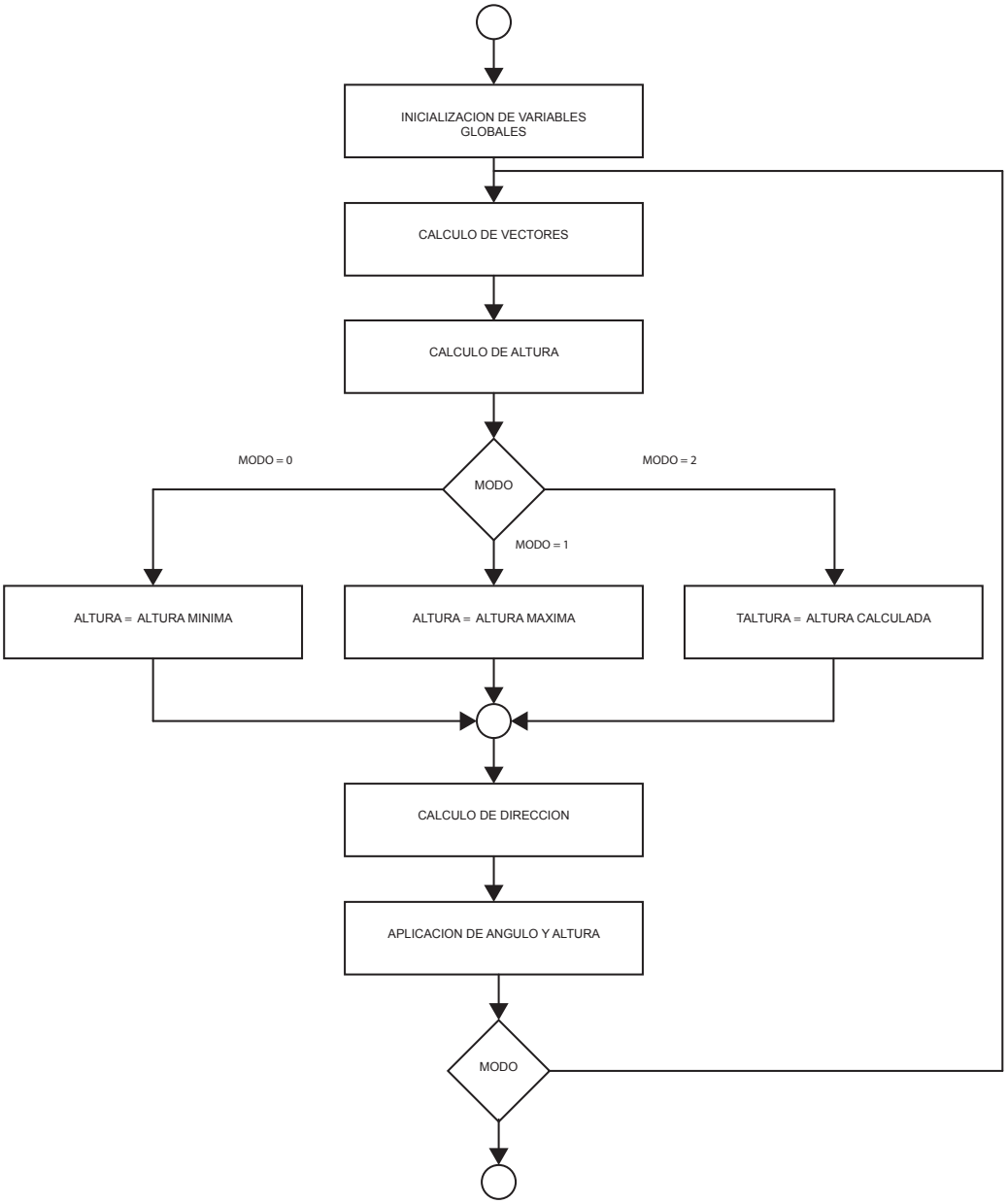


Figura 3.8: Diagrama de flujo de ejecución del sistema de cinemática inversa del componente Projector.

evidente a la hora de implementación, provoca errores graves que retrasaron la finalización del TFG.

El problema fue solventado a través de utilizar maquetas y herramientas *software* para emular el objetivo de la implementación y diferenciar en qué casos concretos se usaban los distintos tipos de coordenadas.

- Errores de precisión: Una vez finalizado el sistema de cinemática inversa se captó un error de precisión en el movimiento que tenía dos orígenes. En primer lugar, el orden de ejecución afectaba a la precisión de los movimientos de las distintas partes con cinemática, debido a que por cada *frame* de ejecución, la cinemática obtenía los resultados correspondientes al *frame* anterior dando sensación real de imprecisión en el movimiento. Por otro lado, se producía imprecisión provocada por la ligera desviación entre el eje de coordenadas local utilizado y el eje de coordenadas real (Ver figura 5.2).

  Para solventar ambos problemas, en primer lugar, se estipuló mediante las posibilidades de la plataforma *Unity* que los sistemas de cinemática fuesen los últimos en ejecutarse por cada *frame*. En segundo lugar, para solventar la desviación, se utilizaron dos ángulos de compensación calculados al inicio de la ejecución. Estos ángulos son ω y θ, que aplicados a los ángulos α y β solventan la desviación existente. (Ver apartado 5.2).

- Adaptación a la plataforma Unity: uno de los grandes problemas se produjo al utilizar *Unity* ya que, al igual que al empezar a utilizar cualquier plataforma, fue necesario un tiempo de aprendizaje que se ha mantenido a lo largo de la realización del TFG. Este problema no se centra solo en el uso de las bibliotecas sino también en particularidades del sistema.

  La solución fue el uso de la documentación [5], bastante extensa y bien estructurada, y el uso continuado de la plataforma.

3.3. Comportamientos

El capítulo *Comportamientos* tratará las animaciones que se han diseñado y cómo interactúan entre ellas. También se explicarán las posibilidades y uso dado a las capacidades proporcionadas por *Unity* para la consecución de los objetivos del trabajo de fin de grado.

3.3.1. Definición de comportamientos

El primer paso una vez conseguido el sistema de cinemática inversa, es el establecimiento y diseño de las animaciones o comportamientos que se crearán para un modelo en concreto, en este caso, el robot de ayuda.

A la hora de diseñar los comportamientos se han considerado futuras necesidades dentro de un entorno virtual, como por ejemplo, andar, señalar, iluminar, etc.

[5] Ver `www.Unity3D.com`

A continuación, hay que pasar a diseñar el modo de llevar a cabo las animaciones elegidas a incorporar en el componente 3D.

El TFG ha dado lugar a la creación de nueve animaciones en total, y en algunos casos, con más de un modo de animación. Por ejemplo, la animación de andar con dos tipos distintos de caminar y con posibilidad de añadir nuevos modos de andar.

Animación abrir

La necesidad de la "animación abrir", surge de las posibilidades del modelo 3D, y consiste en desplegar las patas del robot dando lugar a una mayor interacción con el entorno virtual y a mayores posibilidades de uso del componente.

Animación cerrar

"Animación cerrar" del robot, está estrechamente relacionada con la "animación abrir" debido a que la existencia de una "animación abrir++ exige su animación antagonista, es decir, la animación de "cerrar". De este modo se da la posibilidad al futuro usuario de volver al estado inicial con el robot cerrado y pasar al estado que permite el resto de animaciones, el estado abierto.

Animación andar

Una de las principales capacidades de la que dotar un componente 3D para un entorno virtual es su movilidad por un escenario. Para este cometido se pensaron varios comportamientos posibles, como volar, rodar y andar. Al diseñar e implementar las animaciones se eligió el movimiento mediante pasos, es decir, andar. La elección de dicho método fue para posibilitar el uso del sistema de cinemática inversa diseñado anteriormente y por una mayor visualización del movimiento. Además la utilización del sistema de cinemática inversa daría la posibilidad de establecer distintos tipos de andar y adaptación al escenario.

Animación señalar

Se extrajo la necesidad, dentro de un entorno virtual, de dotar de capacidad de señalar tanto objetivos como posiciones. Es importante el comportamiento de señalar para que el robot sea capaz de proporcionar indicaciones y, por tanto, información añadida o un medio de expresión.

La "animación señalar" utiliza el sistema de cinemática inversa. Se decidió que únicamente fuese posible realizar esta animación cuando las cuatros patas estén sobre el suelo, y solo una pata sería capaz de realizar la animación al mismo tiempo.

Animación escanear

Desde un principio el comportamiento de realizar un escaneo del escenario virtual no se identificó como un comportamiento esencial y solo se incluiría si el tiempo de desarrollo del TFG lo permitía. Como resultado de la planificación, ha sido posible incluirlo. Para la realización del comportamiento "escanear" se usa tecnología de cinemática inversa, *shaders* y sistema de partículas proporcionado por el programa *Unity*.

Animación encender/apagar luz

Al diseñarse este TFG enfocado a integrase con un módulo agente, se exigió que incluyera la capacidad de poder encender luces o un medio de cambiar la luminosidad de la estancia. Para ello se diseñó un método de alumbrado mediante linterna, es decir, aprovechando el modelo 3D del proyector se le ha proporcionado la funcionalidad de uso de una linterna, pudiendo activarse o desactivarse según deseo del usuario.

Animación hablar

Realmente, el "comportamiento hablar" no se basa en la emisión de palabras articuladas sino en el código Morse mediante señales lumínicas. Dentro de la animación de hablar hay un comportamiento particular de saludar pero englobado en el comportamiento de hablar, es decir, se ha proporcionado al robot la habilidad de decir "*Hello*" en código Morse. Dado que codificar palabras en Morse es fácil mediante curvas de animación, se da la posibilidad de crear una animación que codifique frases en código Morse.

Animación cargar

Al igual que el "comportamiento encender/apagar luces" surgió la necesidad de una animación que mostrase como si el modelo 3D estuviese realizando una carga de batería. Fue necesario porque el módulo agente tiene un comportamiento planificador de carga/descarga del componente 3D. También se utilizan las posibilidades de la cinemática inversa del proyector y el propio proyector como linterna con posibilidad de cambiar el color de luz y su intensidad.

Animación proyectar

Como el propio nombre del elemento 3D que forma el robot, el proyector debería ser capaz de proyectar vídeos en paredes. Por tanto, se diseño el comportamiento de proyectar como uno de los principales a implementar. Además, se pensó que sería necesario poder detener la proyección del vídeo pero se evitó la posibilidad de rebobinado o aceleración del mismo e incluso pausa. Siendo, el vídeo a proyectar reproducido desde inicio a fin o en su defecto de inicio hasta parada por deseo específico del usuario.

Animacion idle

"Animación *idle*" surgió para animar el Robot_Ayuda cuando llevase un tiempo en el estado *OpenIdle* sin recibir actividad alguna.

3.3.2. Definición del grafo de animación

Al existir distintas animaciones que interaccionan entre sí, surge la necesidad de definir estados de animación y transiciones entre los mismos. Además cabe resaltar la existencia de varias animaciones en ejecución al mismo tiempo. Por tanto, aparecen diversas capas de estados que mantendrán un estado local por cada capa en el que se encuentre el modelo 3D.

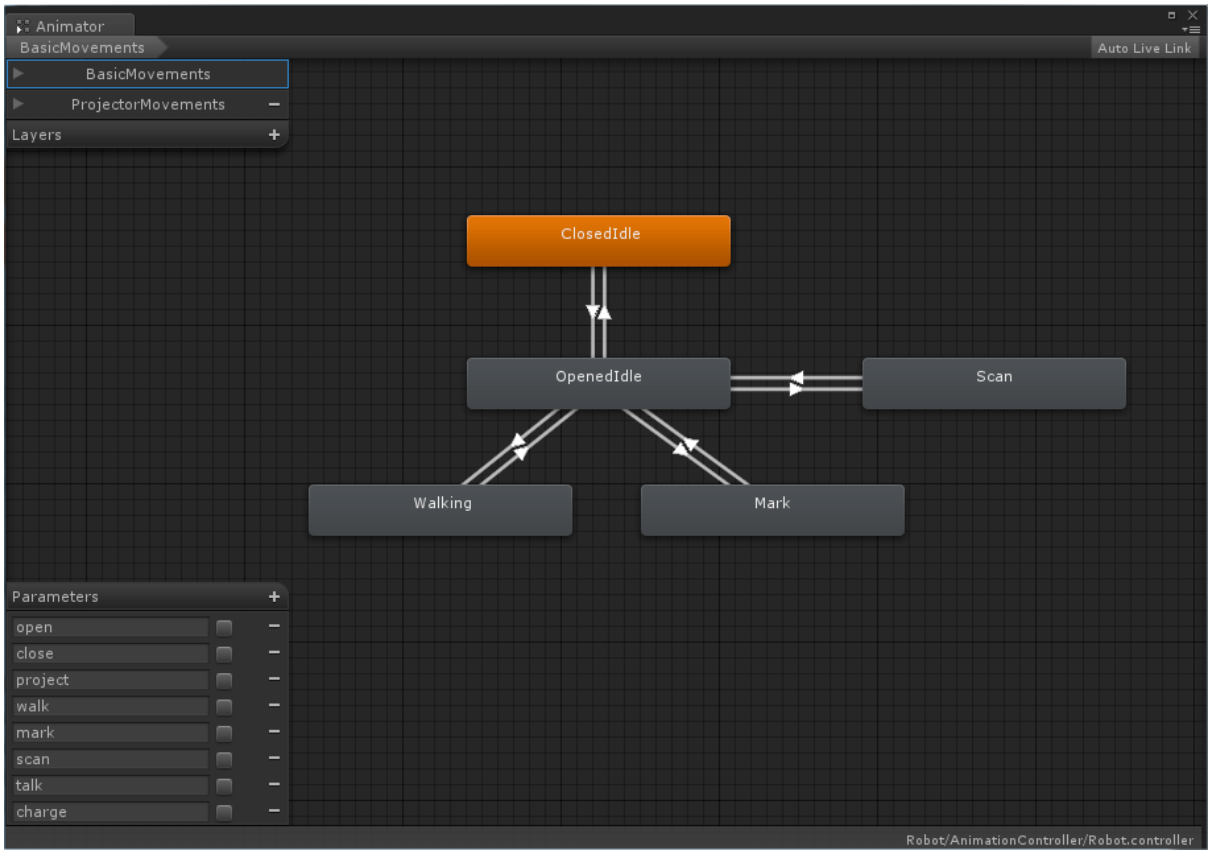


Figura 3.9: Grafo de estados de la animación que dirige los movimientos del cuerpo.

Todas las capas interactúan al mismo tiempo pero independientemente, es decir, que la permanencia en un estado en una capa A no afecta al estado actual de una capa B. Esta capacidad del grafo de estados o animación da lugar a mezcla de animaciones, dependiendo de las partes del modelo que animan o del peso que se quiera dar a cada capa.

Para este proyecto se han utilizado dos capas de animación con un total de nueve estados de animación, cinco en una de las capas y cuatro en otra. La creación de dos capas se planteó debido a las zonas del modelo 3D que controlaban las animaciones que seguidamente se asignarían a estados de ambas capas. Estas capas son:

- *BasicMovements*: Esta capa controla todas las animaciones relacionadas con el cuerpo del robot exceptuando el proyector.
- *ProjectorMovements*: Al contrario que la anterior, esta capa controla todos los estados de animación que interactúan con el proyector del robot.

Sendas capas evitan cualquier conflicto entre ellas debido a que sus zonas de control están totalmente diferenciadas y todas las animaciones manejan las partes del cuerpo asignadas a cada capa, evitando provocar problemas de coherencia entre ambas. Únicamente existe una excepción que aparece en el estado *Scan* de la capa *BasicMovements*, el cual toma el control total del robot durante un tiempo. Para solventar dicho problema

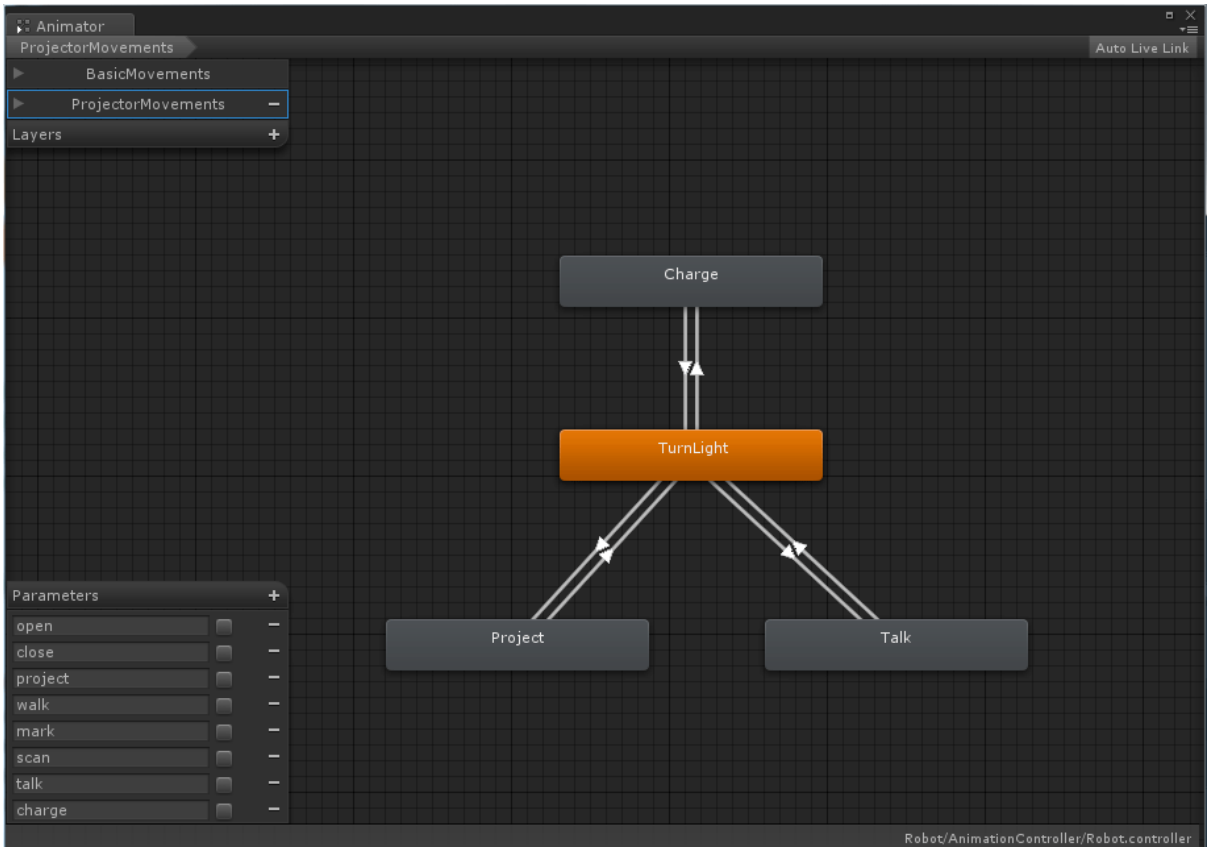


Figura 3.10: Grafo de estados de la animación que dirige los movimientos del proyector.

se le ha asignado mayor peso a la capa *BasicMovements* que *ProjectorMovements*, pero una vez terminado el estado *Scan* el control volverá a ser equitativo y diferenciado por zonas del modelo 3D.

En cuanto al grafo entre estados se ha utilizado la tecnología proporcionada por *Unity*. Esta tecnología permite crear grafos de animación, establecer animaciones, definir transiciones, crear diferentes capas de animación, mezcla de animaciones y mantenimiento de coherencia entre estados y capas. Dicha tecnología, además, proporciona al usuario facilidades en la depuración del grafo de animación dando una interfaz visual en la cual se puede comprobar el estado actual del mismo y el cambio de estados. Sin embargo, esta tecnología fue creada y enfocada para animaciones mediante el uso del nuevo sistema de animación *Mecanim*. El uso que se le ha dado desde el punto de vista práctico ha sido únicamente lógico, ya que los estados de animación no tienen asignados animaciones. Esto se debe a que todas las animaciones del componente 3D son mediante *scripting* y utilizan las posibilidades del sistema de cinemática inversa.

La capa *BasicMovements* contiene los siguientes estados:

- *CloseIdle*: El acceso a este estado provoca el cierre del robot. Mientras el estado de la capa *BasicMovements* sea *CloseIdle* el robot se mantendrá cerrado.

- *OpenIdle*: La permanencia del grafo de animación en este estado, indica que el robot está abierto. Además, pasado un tiempo, se podrá activar una animación en caso de falta de actividad del componente 3D.

- *Walk*: El estado *Walk* indica que el modelo 3D se encuentra andando y por consiguiente, moviéndose por el escenario.

- *MarkObjective*: Este estado muestra que el componente 3D se encuentra realizando la animación de señalar a un punto del escenario 3D.

- *Scan*: Este estado toma el control total del modelo 3D realizando una animación de escaneo del escenario.

Por otro lado, el grafo de animación tiene las siguientes transiciones entre estados:

- *CloseIDle-OpenIdle*: Esta transición activa la animación Open que abrirá el modelo 3D y cambiará el estado del grafo de animación al estado *OpenIDle*. Solo se activará esta transición cuando el módulo que controle el componente 3D mande la orden de abrir.

- *OpenIdle-CloseIdle*: La transición *OpenIdle-CloseIdle*, al igual que la transición *CloseIdle-OpenIDle*, se activará cuando el módulo controlador dé la orden, en este caso dé cerrar, al componente 3D. Cuando esta transición se active, se ejecutará la animación de cerrar y el estado cambiará a *CloseIdle*.

- OpenIdle-Walk: Esta transición se llevará a cabo cuando el módulo controlador dé la orden de caminar al componente 3D y se cambiará el estado actual del grafo de animación al estado *Walk*. La transición *OpenIdle-Walk* comenzará la animación de caminar creada para este modelo 3D.

- *Walk-OpenIdle*: Esta transición puede ser activada mediante dos medios: el primero, mediante el fin de la animación de andar, y el segundo mediante la parada ordenada por el módulo que controle el componente 3D. Una vez ejecutada la transición el estado del grafo será *OpenIdle*.

- *OpenIdle-MarkObjective*: Esta transición se ejecutará cuando el módulo controlador dé la orden de señalar a un objeto del entorno virtual. El paso por esta transición conlleva el paso del grafo de animación al estado *MarkObjective* y el comienzo de la animación diseñada para el Robot_Ayuda.

- *MarkObjective-OpenIdle*: Esta transición solo se activará cuando la animación de señalar esté totalmente concluida, pasando al estado *OpenIdle* del grafo de animación.

- *OpenIdle-Scan*: Esta transición solo se efectuará al recibir la orden del módulo controlador y cambiando el estado actual del grafo al estado *Scan*. Además, realizará el comienzo de la animación de escaneo diseñada para el componente 3D.

- *Scan-OpenIdle*: Esta transición únicamente se ejecutará cuando la animación de escaneo esté totalmente acabada y seguidamente cambiará el estado del grafo de animación a OpenIdle.

En cambio, la capa *ProjectorMovements* agrupa los siguientes estados de animación:

- *TurnLight*: El estado *TurnLight* es el que se mantendrá por defecto o al que se volverá una vez que se termine cualquier acción iniciada del proyector. Cuando se pasa a este estado la luz vuelve al estado anterior pero mantiene la intensidad y el último ángulo utilizado.

- *Talk*: En este estado el Proyector del robot se encuentra proyectando señales luminosas que darán lugar a un mensaje en código Morse. Además, este estado también agrupa la animación particular de mandar la secuencia de señales Morse que darían lugar a la palabra *"Hello"* que sería un caso particular del estado *Talk*.

- *Project*: Estar en el estado *Project* significará que el Robot_Ayuda se encuentra proyectando una textura animada, es decir, un vídeo.

- *Charge*: El estado *Charge* indica que el Robot_Ayuda se encuentra realizando una carga y ejecutando una animación que muestra el estado de la carga.

Y las transiciones que agrupa son:

- *TurnLight-Talk*: La transición se llevará a cabo cuando el módulo controlador envíe la orden *Talk* o *SayHello*, pasando a ejecutar la animación deseada por el usuario.

- *Talk-TurnLight*: Una vez finalizada la animación de hablar o saludar, se llevará a cabo la transición *Talk-TurnLight*. El único momento en el que se producirá la transición será cuando cualquiera de sendas animaciones hayan finalizado totalmente.

- *TurnLight-Project*: Al igual que la transición *TurnLight-Talk*, se llevará a cabo si el módulo inteligente manda ejecutar la animación de proyectar.

- *Project-TurnLight*: También, como en el caso de la transición *Talk-TurnLight*, se llevará a cabo cuando la animación esté totalmente terminada. Sin embargo, esta transición tiene un modo más de activarse, es el caso en que el usuario solicita parar la proyección dando lugar a la ejecución de la transición.

- *TurnLight-Charge*: El paso por la transición se producirá al recibir la orden procedente del módulo controlador del componente 3D que ordene reproducir la animación de cargar.

- *Charge-TurnLight*: En este caso, solamente se saldrá del estado *Charge* al estado *TurnLight* cuando la animación de *Charge* termine en su completud. A diferencia de la transición *Project-TurnLight* no existirá otro modo de pasar por la transición *Charge-TurnLight*.

En lo referente a los cambios de estado por terminación de las respectivas animaciones, se realizan mediante la activación/desactivación por *script* de atributos booleanos. *Mecanim* permite el cambio de un estado a otro mediante un tiempo de duración perteneciente a una animación pero, en este caso y como se ha mencionado anteriormente, el sistema *Mecanim* solo se utiliza desde el punto de vista lógico y, por tanto, ha sido necesario adaptarse a este sistema.

3.3.3. Implementación

Estructuras auxiliares

En algunas de las animaciones se han incluido objetos vacíos[6] que se han utilizado para crear animaciones. La mayoría de los elementos auxiliares están contenidos, dentro de la jerarquía, en el objeto Robot_Ayuda. Por tanto, los elementos auxiliares utilizados han sido:

- *ClosePosition*: Su finalidad es servir como posición de referencia a la hora de apoyar el Robot_Ayuda sobre una superficie. Por consiguiente, *ClosePosition* será utilizado en la implementación de las animaciones "Cerrar", "Abrir" y "Escanear"[7]. El objeto auxiliar está situado en la parte inferior de la esfera que forma el cuerpo del robot.

- *ParticleCamera*: *ParticleCamera* es una cámara auxiliar que muestra los puntos detectados después de realizar una animación de escaneo. La cámara únicamente visualiza partículas. Las partículas se crean dinámicamente durante la ejecución de escaneo. Es necesario incluir un sistema auxiliar que genere las partículas.

    - *ParticleSystem*: Está contenido dentro del objeto auxiliar *ParticleCamera*. *ParticleSystem* es el sistema de partículas que proporciona *Unity* que mediante un *script* permite el manejo y uso del mismo.

- *ScanPlane*: La animación de escaneo usa un plano que tiene aplicado un *shader* que da la sensación visual de un plano formado por láseres. Este plano es únicamente visual y es activado/desactivado según indique la animación de escanear.

[6] *GameObject Empty* en la plataforma *Unity3D*

[7] En este caso se utiliza por el uso de los métodos de cerrar y abrir

- *WalkPositions*: *WalkPositions*, al igual que *ClosePosition*, es un objeto vacío pero que a diferencia de *ClosePosition* contiene otros objetos vacíos. Su uso es diverso ya que se utiliza en distintos comportamientos, andar, abrir y escanear. *WalkPositions* tiene asignado un *script* que relaciona el objeto robot con los objetos contenidos. Su finalidad es dar soporte a las diferentes posiciones de los puntos de apoyo de las Union-Y. Para ello se utilizan cuatro objetos vacíos que son:

  - *WalkPosition_1*: Marca la posición de apoyo de la Union-1.
  - *WalkPosition_2*: Marca la posición de apoyo de la Union-2.
  - *WalkPosition_3*: Marca la posición de apoyo de la Union-3.
  - *WalkPosition_4*: Marca la posición de apoyo de la Union-4.

  Implícitamente, cada componente de *WalkPositions* tiene asignado un *script* que se encarga de lanzar *RayCast* hacia el suelo y devolviendo la información del punto de contacto.

Además, existe una estructura auxiliar más, pero está contenida en otro subnivel de la jerarquía de objetos del componente. En este caso es el componente *Projector*[8], el cuál está contenido dentro del objeto 3D Proyector. Su finalidad es servir como soporte a la hora de proyectar vídeos durante el estado de animación de proyectar. La funcionalidad del componente *"Proyector"* de *Unity* permite reproducir texturas animadas en otros objetos 3D del escenario virtual.

Animación abrir

La animación abrir se ha divido en dos subestados para logar la apertura del componente de forma dinámica y adaptable. En primer lugar, el Robot_Ayuda debe estar en el estado de animación *CloseIdle* y activar el atributo booleano *open*.

Una vez realizado el primer paso, se entra en el primer subestado de apertura de las patas y a través de la estructura auxiliar *WalkPositions* se lanzan *RayCast* para detectar las posiciones donde las patas del robot deberán apoyar y se crean nuevos objetivos TG_Y, situándolos en la posición actual de los PR_Y de cada Union-Y. El tiempo de apertura es parametrizable, siendo la duración del subestado la mitad de la duración total. Durante el subestado, la posición de los objetivos se interpolará desde la posición actual de los PR_Y hasta la posición detectada mediante *RayCast*.

Una vez terminado el primer subestado, se comienza el segundo subestado de elevación del cuerpo. Idénticamente al primer subestado, el tiempo de duración es la mitad del tiempo total y, durante ese tiempo, la posición de la Esfera_Robot se irá elevando, hasta alcanzar la altura deseada.

[8] `http://docs.unity3d.com/Documentation/Components/class-Projector.html`

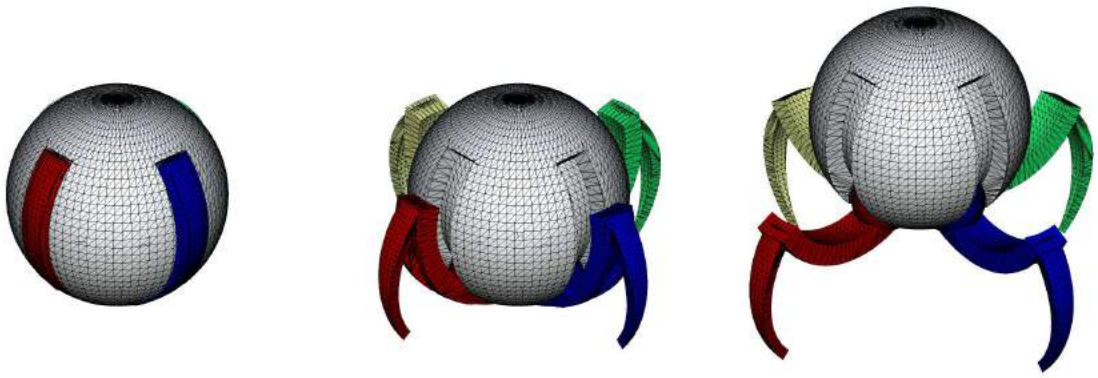


Figura 3.11: Fotogramas obtenidos a partir de la grabación de “animación abrir”

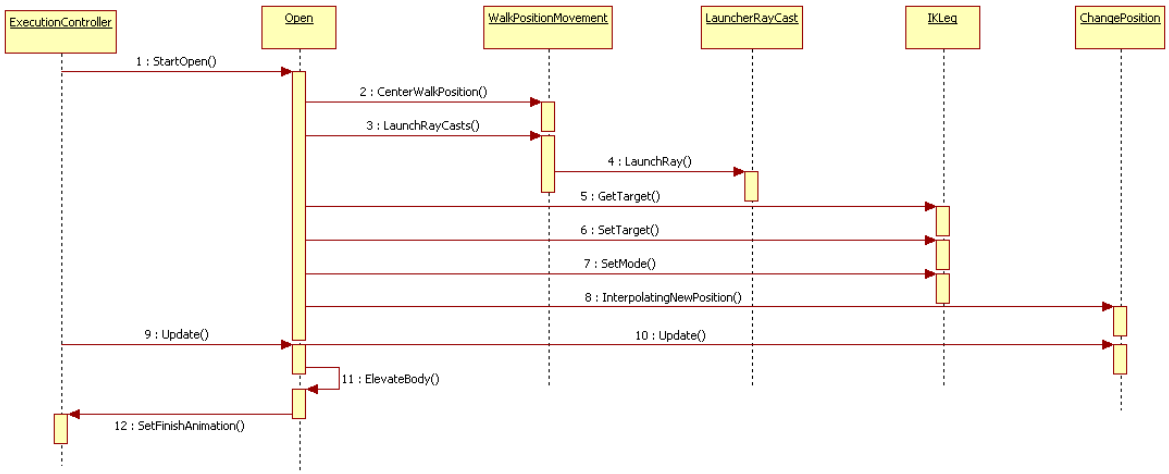


Figura 3.12: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación cerrar

La “animación cerrar” está dividida en dos subestados tanto para mantener la coherencia de la animación como para convertirla en una animación adaptable. Inicialmente, el estado desde el cual se puede mandar cerrar el modelo 3D es *OpenIdle*, en el cual el Robot_Ayuda estará abierto o en proceso de apertura. Cuando se reciba la orden de cerrar el componente 3D, se activará el atributo booleano *close* dando lugar al paso por la transición *OpenIdle-CloseIdle*.

Una vez producido los cambios iniciales, la ejecución de la animación usará dos subestados al igual que en el caso de “animación abrir”. Los dos subestados se dividen en; descenso de la Esfera_Robot, y apertura de las Union-Y. Al producirse la transición se pasaría al primer subestado donde se buscaría mediante *Raycast* la posición donde se encuentra el suelo u objeto sólido donde apoyar. El *RayCast* se lanza desde la estructura

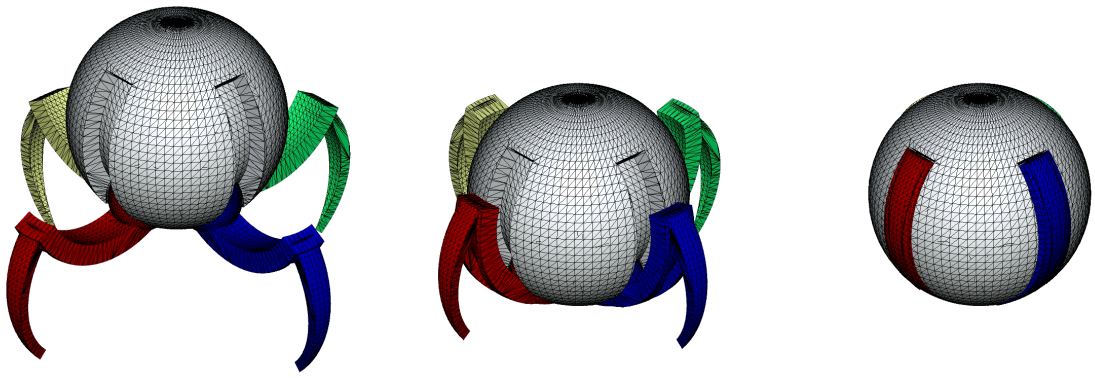


Figura 3.13: Fotogramas obtenidos a partir de la grabación de “animación cerrar”.

auxiliar *ClosePosition* para evitar la detección del propio cuerpo del Robot_Ayuda. Cabe reseñar que el punto de apoyo es la estructura auxiliar *ClosePosition* ya que, si se utilizase el *pivot* del Robot_Ayuda, se situaría atravesando el objeto de apoyo y por tanto, daría lugar a errores de coherencia, visualización y animación. Además, la ejecución de la animación solicita un tiempo de duración total, siendo la mitad para el primer subestado y la otra mitad para el segundo subestado.

Una vez terminado el primer subestado se pasaría al cierre de las Union-Y, para ello se utiliza la posición de los PR_Y en el momento inicial de arranque del sistema para poder interpolar desde su posición actual hasta la posición mencionada. El encargado del movimiento de las diferentes Union-Y es el sistema de cinemática, ya que el único trabajo del *script* de animación es mover los TG_Y correspondientes.

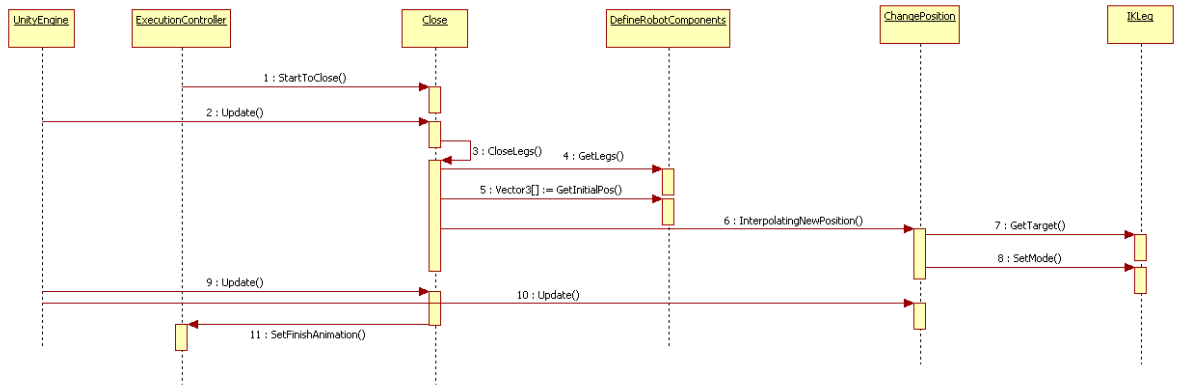


Figura 3.14: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación andar

Esta animación ha sido la más compleja de lograr debido a la cantidad de conceptos a evaluar y desarrollar para conseguir un movimiento fluido y coherente. En primer lugar,

la "animación andar" puede ser llamada desde los estados *OpenIdle* y el propio estado *Walk*. En el caso de estar situado en el estado *OpenIdle*, la transición empleada para llegar al estado *Walk* será *OpenIdle-Walk*; en caso contrario, se mantendrá en el estado *Walk*. Una de las primeras decisiones que hubo que realizar fue el diseño de la cinemática de la animación. Para tomar esta decisión hubo que plantearse las siguientes cuestiones:

1. Si la Esfera_Robot sería afectada por el movimiento de las patas.
2. Número de Union-Y que estarían apoyadas al mismo tiempo en el suelo durante el paso.
3. Si el Robot_Ayuda tendría una cara o dirección fija con la que andar hacia adelante.
4. La velocidad.
5. La longitud del paso.

Estas preguntas ocasionaron las siguientes respuestas:

1. El único efecto que sufriría la Esfera_Robot sería el provocado por la altura de la ruta o escenario.
2. Se diseñaron dos modos, el primero en el que se mantendrían apoyadas siempre dos de las cuatro Union-Y y otro que mantendría apoyadas tres de las cuatro sobre el suelo.
3. El Robot_Ayuda al poder rotar el proyector se decidió que no tendría cara definida.
4. La velocidad dependerá del usuario que controle el Robot_Ayuda, ya que tanto la ruta como el tiempo del recorrido de la misma, dependerá del módulo controlador.
5. La longitud de un paso, sin incluir la posibilidad de correr, se realizó de forma que la escala no interviniese en la cinemática de la animación. Por lo que se deduce que la longitud del paso es variable pero no modificable por el usuario controlador.

Solventadas las preguntas, se decidió dividir la animación de andar en dos interpolaciones de posición, por un lado la de la Esfera_Robot y por el otro la de las patas. Para conseguir la interpolación del cuerpo se utilizaron curvas de animación si bien, no eran utilizadas como una animación. Las curvas, una por cada variable del espacio 3D, se crean dinámicamente guardando puntos claves en cada momento del tiempo, acotado entre 0 y 1, donde cambia la dirección de andar. La posición en el tiempo dentro de la curva se efectúa de manera que se establece una proporción entre la longitud del trayecto total y la porción del trayecto. La relación de proporción da como resultado un valor entre 0 y 1 que es el lugar temporal dentro de la curva donde se coloca el punto de referencia. Una vez preparadas las curvas de animación a través de otro *script* se da la opción de evaluar las curvas en el momento actual. Este *script* se basa en guardar tanto el tiempo de comienzo como el tiempo de finalización de la animación y proporcionar un método que evalúa cuál es la posición del objeto en ese momento. Finalmente el *script* encargado de actualizar la posición de los objetos se encarga de ordenar la inicialización de los tiempos necesarios como las curvas con las futuras posiciones.

Por otro lado, está la actualización de las posiciones de las Union-Y. Para lograr este hito, se crearon animaciones auxiliares con medios procedentes de las animaciones como eventos programados y reproducción en bucle.

El primer paso a construir era la inicialización de los pasos de forma cíclica, surgiendo como solución la posibilidad de ejecutar una animación en bucle y programar eventos que invocarían una función auxiliar que actualizaría tanto el punto inicial como el final. Para cada instante de lanzamiento del evento, la funcionalidad indica que el punto final actual pasa a ser el inicial del siguiente ciclo y el final se calcula gracias a la estructura auxiliar *WalkPositions*.

Esta estructura auxiliar envía hacia el suelo un *RayCast* para cada Union-Y y devuelve los futuros puntos de apoyo de las mismas. El segundo paso a construir era la interpolación del movimiento de cada Union-Y de forma que se pudiese garantizar la coherencia en la animación. Entonces se volvió a utilizar la posibilidad de trabajar con animaciones que duran un segundo aprovechando la animación creada en el primer paso del movimiento de las Union-Y. En este caso se utilizaron cuatro variables que modificarían la posición entre los puntos inicial y final de cada una y una variable más que modificaría la altura relativa de los pasos. Estas variables se repitirían por cada ciclo de paso. La utilización de animaciones auxiliares dio lugar a un fácil diseño de cinemáticas de andar, pudiendo ser probadas con solo modificar las curvas de la animación auxiliar.

El tercer problema a resolver era la dirección en la que andar, como adaptar el cambio de dirección y cuándo parar. En el primer caso, se comprendió que los puntos donde estaban definidos los cambios de dirección estaban guardados en las curvas que interpolan la posición del cuerpo. Esto dio como resultado que cuando se creaban las curvas, también se creaban eventos por cada punto clave, excepto el inicial y final, que ordenaban el cambio de la dirección en la que andar. Una vez logrado el primer caso, al probar la solución se observó un desfase en los pasos que no debería suceder. El desfase en la animación se debía a que el momento de cambio de dirección no tenía por qué coincidir con el fin de ciclo de un paso. La solución propuesta fue que en cada momento de cambio de dirección se volviesen a actualizar los puntos inicial y final donde apoyar pero sin reiniciar la animación auxiliar. El último problema era cuándo terminar la animación, planteándose dos posibilidades. La primera que fuese parada por el usuario, lo que ocasionaría que se detuviese bruscamente quedando el Robot_Ayuda congelado. La segunda a la finalización del recorrido. Para detener el Robot_Ayuda una vez recorrida la ruta, al igual que en el primer caso, se insertó un evento en las curvas de interpolación de la Esfera_Robot en el último punto clave que finalizaría tanto la animación auxiliar como la interpolación de las distintas partes.

También hay que mencionar que aunque hay diseñados tres modos de andar, solo se utilizan dos, ya que en el caso de mantener tres patas apoyadas, la velocidad de la animación no permite realizarla correctamente.

Una vez acabada la ruta o terminada por el usuario se pasa del estado *Walk* al estado *OpenIdle* por medio de la transición *Walk-OpenIdle*.

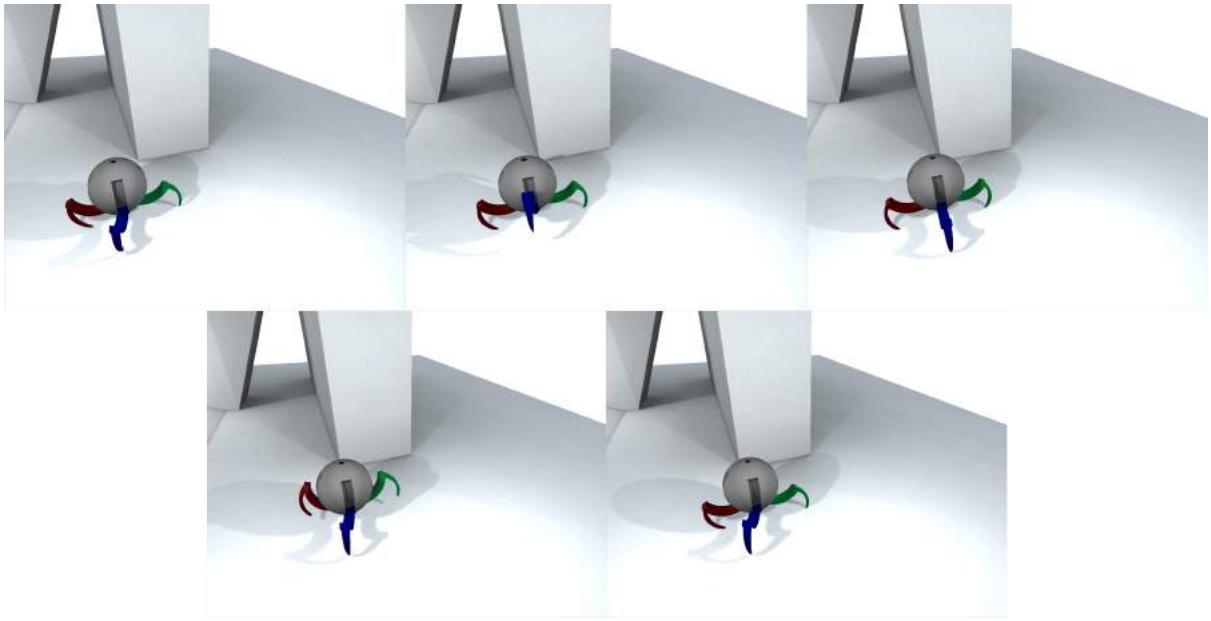


Figura 3.15: Fotogramas obtenidos a partir de la grabación de “animación andar”.

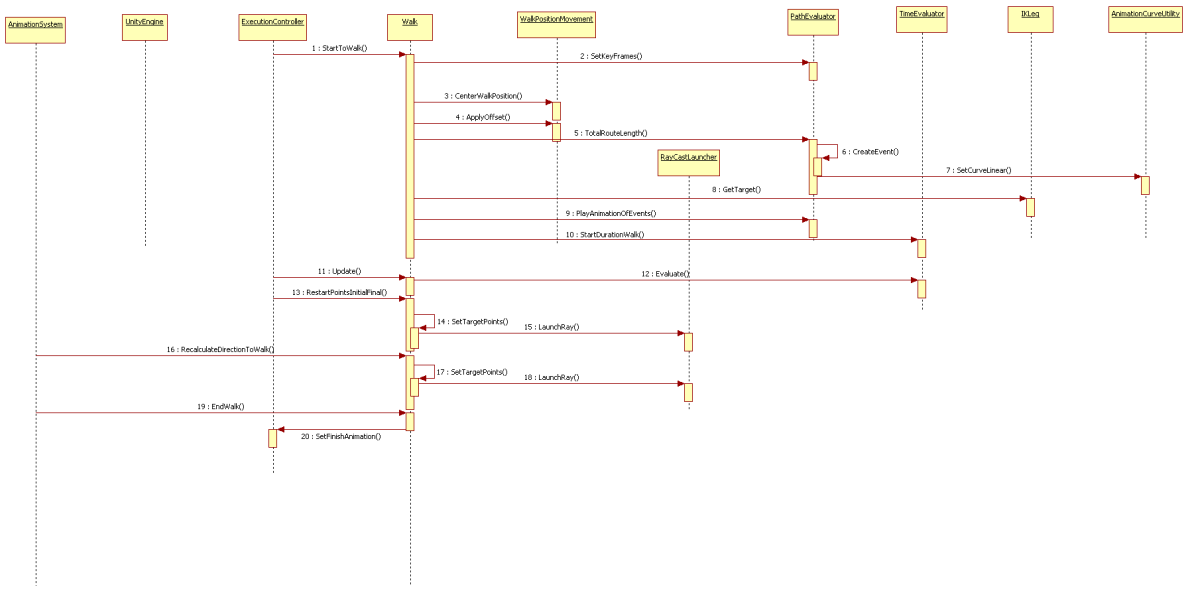


Figura 3.16: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación señalar

La “animación señalar” solo se podrá ejecutar desde el estado *OpenIdle*, desde el cual se invocará la orden y a través de la transición *OpenIdle-MarkObjevtive* se pasará al estado *MarkObjevtive*. Para realizar está animación se ha utilizado tanto la tecnología de

cinemática inversa desarrollada, como las curvas de animación y bibliotecas relacionadas que son proporcionadas por *Unity*.

Cuando se produce la transición al estado *MarkObjective*, se solicitan dos valores; uno el tiempo de duración, otro el objeto del entorno virtual al que hay que señalar. Para llevar a cabo la animación se utiliza una curva de animación que actualiza el valor de la variable utilizada en el *script*. Esta variable se encarga de medir el momento de la interpolación, siendo la duración de la animación de un segundo. Es importante que el tiempo de duración sea de un segundo ya que de este modo se puede controlar el tiempo total de ejecución mediante la ralentización o adelantamiento de la reproducción. Por ejemplo, si se quiere que la animación dure tres segundos, la velocidad de reproducción será un tercio de la inicial, y dado que inicialmente dura un segundo por defecto, implicará que la duración final será de tres segundos.

Por consiguiente, la acción que se realiza al reproducir la animación se basa en los siguientes pasos:

1. Búsqueda de la Union-Y más cercana al objetivo.

2. Adaptación de la velocidad de reproducción de la animación de apoyo.

3. Lanzamiento de la animación de apoyo y actualización de la posición del TG_Y que se corresponde con la Union-Y.

El movimiento de la Union-Y elegida será tarea del sistema de cinemática inversa.

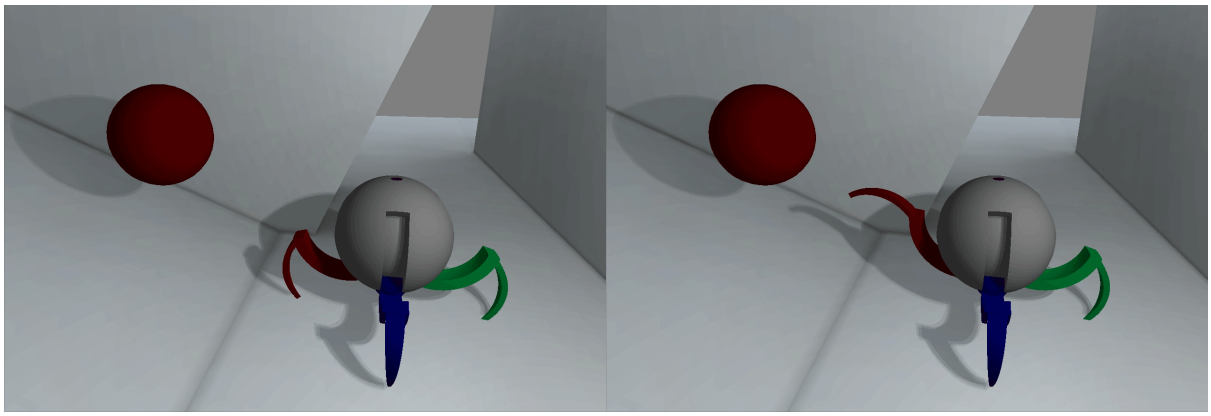


Figura 3.17: Fotogramas obtenidos a partir de la grabación de “animación señalar”.

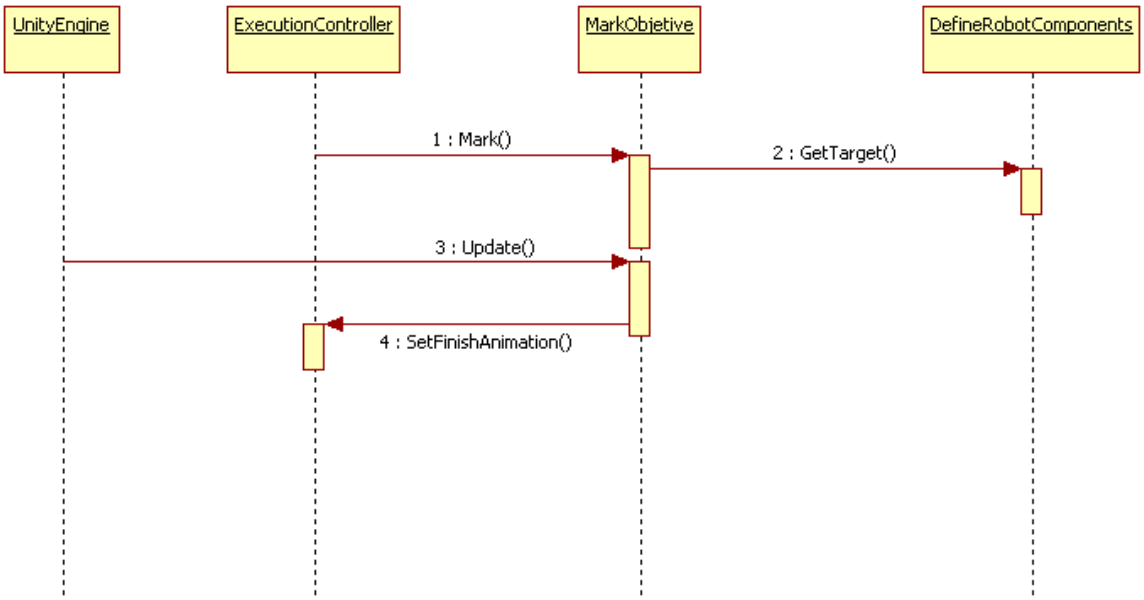


Figura 3.18: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación escanear

La "animación escanear" solo es accesible desde el estado *OpenIdle* y a través de la transición *OpenIdle-Scan*. Esta animación usa diversos medios: por un lado animaciones ya creadas, en otras palabras, las animaciones "cerrar" y "abrir", uso de sistema de partículas proporcionado por *Unity*, uso de *shaders* y curvas de animación.

En este caso, la animación se compone de tres subestados: primero el proceso de cerrar, segundo el proceso de escaneo y tercero el proceso de apertura. Para acceder a la reproducción de esta animación se solicita un tiempo de duración total, el cual se repartirá entre los tres subestados según la siguiente regla:

1. El tiempo total se dividirá en tres porciones de tiempo de distinta duración.

2. El primer subestado, proceso de apertura, durará 1/5 de su tiempo total.

3. El segundo subestado, proceso de escaneo, durará 3/5 del tiempo total.

4. El tercer subestado, proceso de cerrado, durará 1/5 del tiempo total.

La decisión de dividir en cinco partes el tiempo total se debió a que el proceso de escaneo tiene tres fases; elevación, escaneo y descenso; se diseñó que cada fase durase una porción del tiempo a repartir. Por otro lado, se decidió que tanto la "animación cerrar" como la "animación abrir" deberían durar a su vez una porción del tiempo total cada una. Concluyendo en que el tiempo total debía ser dividido en cinco partes con la repartición mostrada anteriormente.

A la hora de realizar la animación, el primer subestado reutilizaba la "animación cerrar" invoncándola con el tiempo deseado, 1/5 del tiempo total. A pesar de utilizar la "animación cerrar" el estado de animación se mantiene y además se programa un evento que daría paso al segundo subestado de la animación, el subestado de escaneo.

Cuando se termina el primer subestado y se llega al segundo, se realizan las siguientes acciones dependiendo de la fase en que se encuentre. Por tanto las acciones son:

1. Fase 1: Al igual que en la "animación señalar", se modifica la velocidad de reproducción para que dure el tiempo deseado, ya que también en este caso la animación auxiliar tiene una duración de un segundo. Más adelante, se lanza un *RayCast* hacia arriba para detectar si existe algún obstáculo a la hora de elevarse. Si no se encuentra obstáculo alguno, el Robot_Ayuda se elevará hasta una altura por defecto, y en caso contrario, a una altura que evite el choque con el obstáculo interpuesto. Además, hay que mencionar que el *Collider*[9] del Robot_Ayuda se desactiva al inicio de esta fase para evitar errores de detección de sí mismo y se reactivará al terminar la última fase.

2. Fase 2: En este momento el Robot_Ayuda se encontrará en su altura máxima. En primer lugar se activará el objeto auxiliar *ScanPlane* que utiliza la tecnología de *shaders*. Una vez activada la estructura auxiliar, se realiza una interpolación de la rotación que se aplica en dos de sus ejes, el X y el Z. A partir de este momento, por cada *frame* se lanzaran *RayCasts* en diferentes direcciones detectando estructuras que tengan *Collider* asignados. Además, por cada *frame*, se crearán partículas por cada choque detectado, es decir, por cada *RayCast*. Cuando la animación auxiliar acaba, se lanza un evento que pasa la ejecución a la última fase, también se desactivará el *ScanPlane*.

3. Fase 3: Esta fase se encarga de descender el Robot_Ayuda desde la altura alcanzada en la primera fase, hasta la posición inicial de la fase 1 (apoyado en el suelo). La tarea que se realiza en esta fase es la interpolación de la posición del componente 3D a través de animación de apoyo. Una vez apoyado el modelo en el suelo, la animación auxiliar manda un evento que invoca la ejecución del tercer subestado.

Además, hay que reseñar que aunque se decidió que para cada fase se asignaría una porción de tiempo, en realidad se redujo el tiempo de las fases 1 y 3, con objeto de incrementar la fase 2 para darla prioridad sobre las otras, debido a que es el elemento principal de la animación, es decir, escanear.

Por último el último subestado reutiliza la animación de abrir y al igual que en el primer caso, el grafo se mantiene en el estado de animación *Scan*. Al empezar este subestado, se programa un evento que ejecutará la transición *Scan-OpenIdle* activando las variables necesarias.

[9] `http://docs.unity3d.com/Documentation/Components/class-BoxCollider.html`

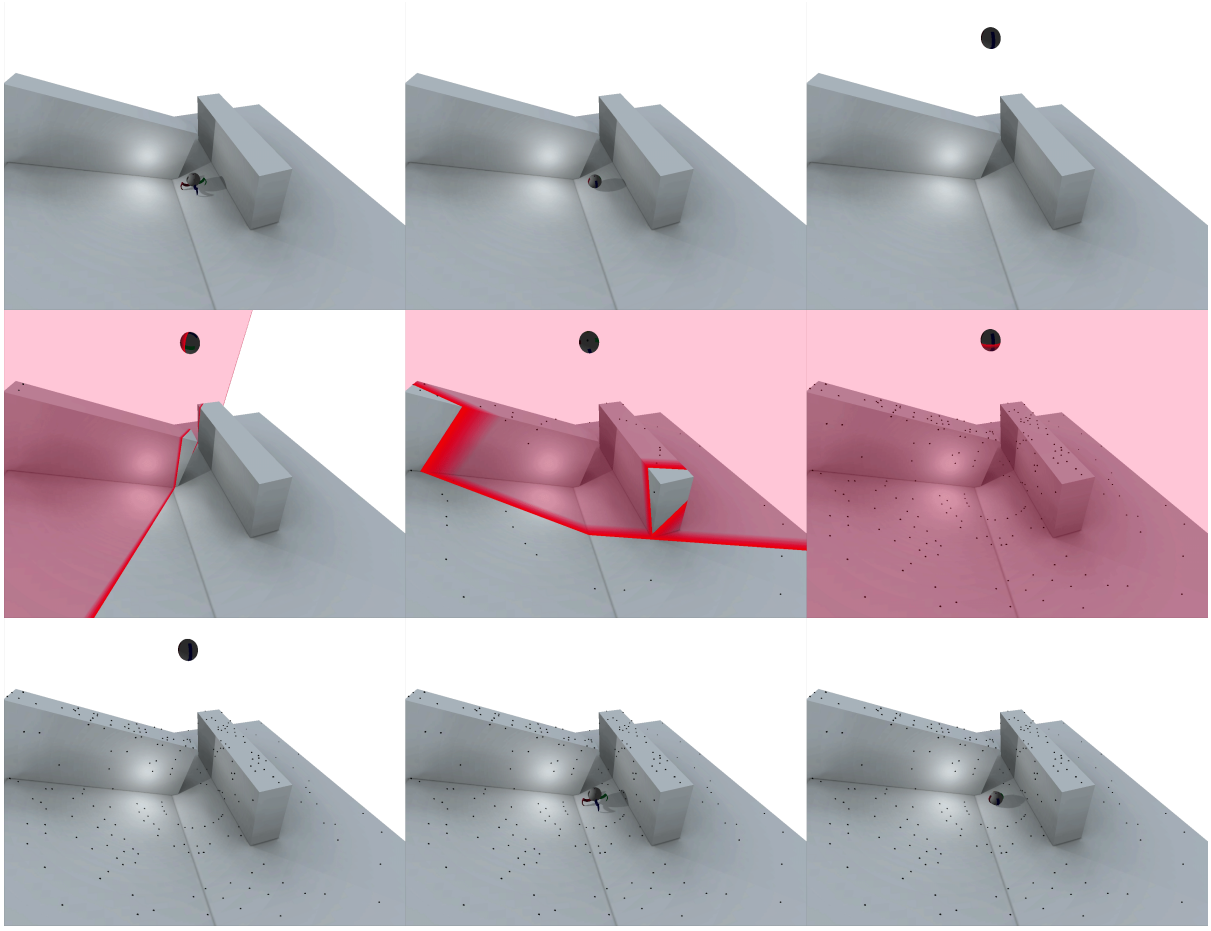


Figura 3.19: Fotogramas obtenidos a partir de la grabación de “animación escanear”.

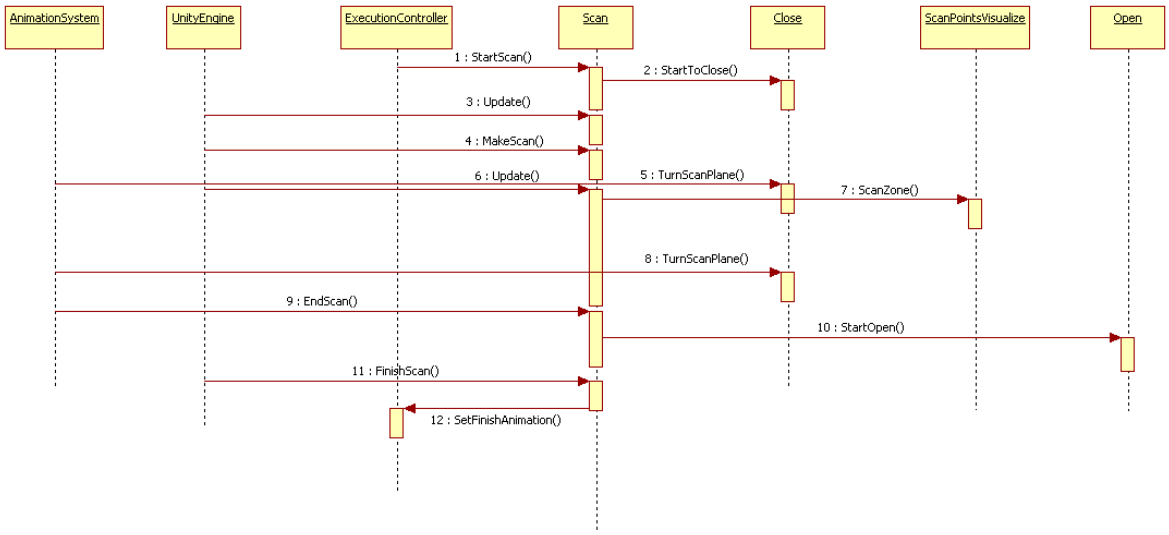


Figura 3.20: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación encender/apagar luz

La "animación encender/apagar luz" solicita una posición, un color y una duración en el caso de "encender luz", y solo se podrá llevar a cabo cuando el grafo de animación de la capa *ProjectorMovements* esté en el estado *TurnLight*. El color pasado como parámetro será el de la luz, la posición el lugar, donde apuntarla, y la duración referencia el tiempo que tarda el proyector en cambiar desde la posición inicial a la pasada como parámetro.

En el caso de apagado se pide una posición y una duración que será el tiempo que tardará el proyector de cambiar desde la posición actual a la posición pasada, y al igual que en el caso de encendido, solo se ejecutará si se encuentra en el estado *TurnLight*.

Para llevar a cabo ambas acciones al comienzo de la ejecución, se crea y configura un *PointLight* dinámicamente. Después, en lo concerniente a la luz, se consiguen los efectos deseados modificando a través de *script* los atributos de la luz definidos por Unity[10].

Inicialmente, la intensidad y el ángulo de iluminación están asignados por defecto pero pueden ser variados por funciones auxiliares. También hay que mencionar que la ejecución de ambas acciones no provocará un cambio en el estado del grafo de animación.

[10] `http://docs.unity3d.com/Documentation/ScriptReference/Light.html`

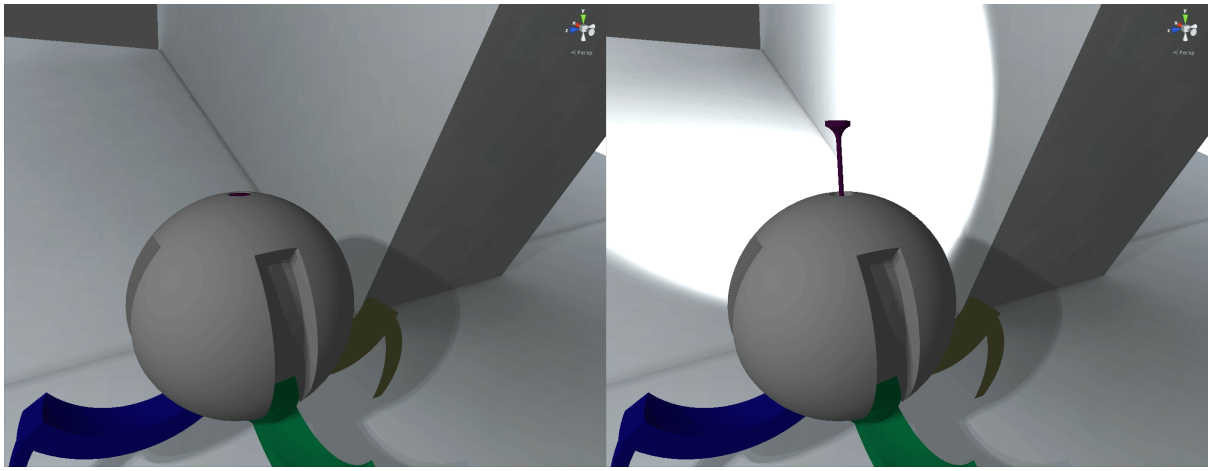


Figura 3.21: Fotogramas obtenidos a partir de la grabación de "animación encender/apagar luz".

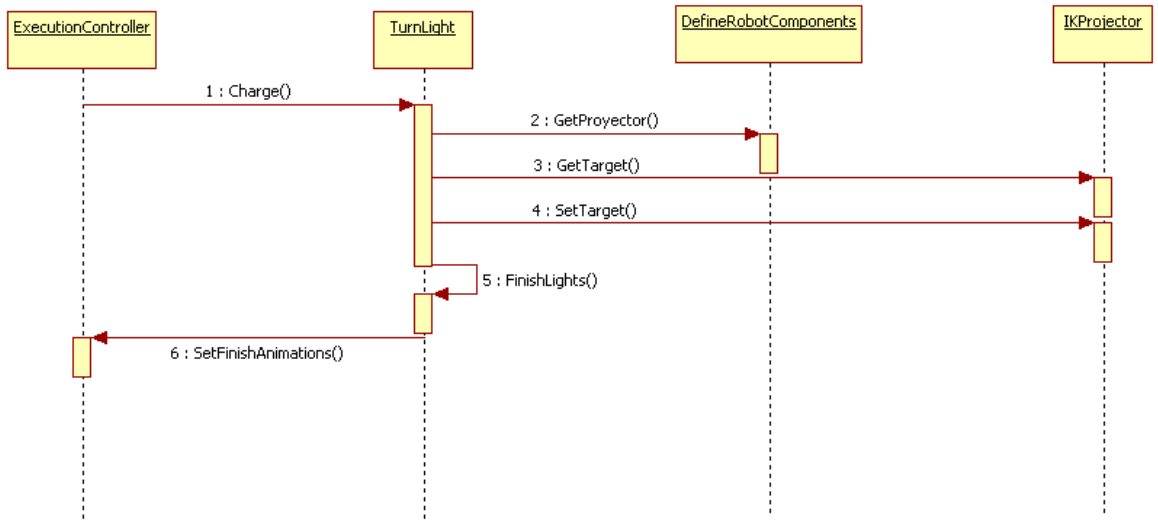


Figura 3.22: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación hablar

La "animación hablar" solo será accesible cuando el estado del grafo de animación sea *TurnLight*, y una vez lanzada la animación se pasaría a *Talk* por la transición *TurnLight-Talk*. La acción que se realizará, independientemente del mensaje, será la emisión de señales luminosas en una dirección Para ello, se utiliza una animación de apoyo para el caso de saludar o bien se puede pasar como parámetro cualquier animación que codifique el mensaje a transmitir. Por tanto, cuando el método es llamado, se lanza la animación correspondiente y se modifica la intensidad y el color de la luz a unos valores por defecto.

Durante la ejecución la acción a realizar será activar y desactivar la luz dependiendo del valor de un atributo que es modificado por la animación auxiliar que se reproduce. Terminada la animación, se vuelve al estado *TurnLight* a través de la transición *Talk-TurnLight*.

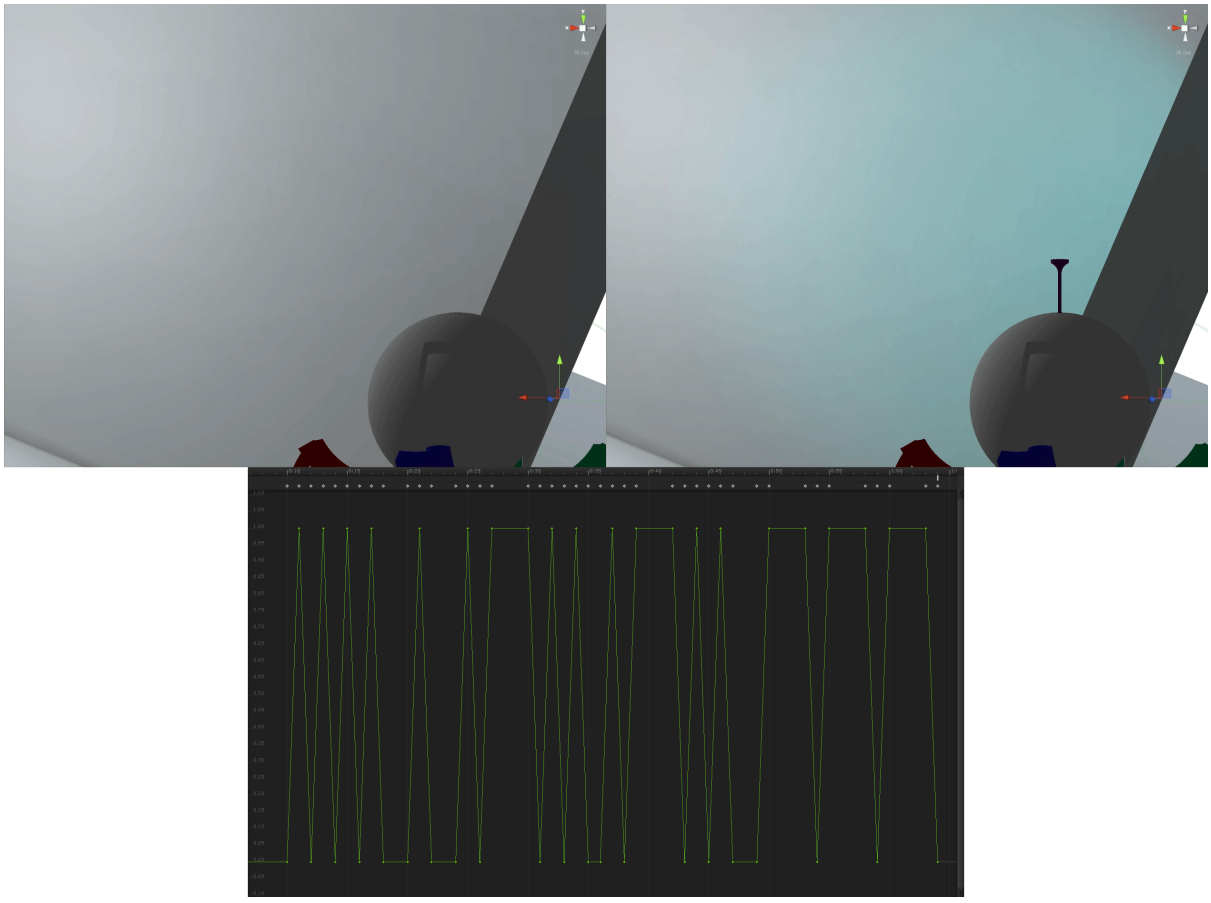


Figura 3.23: Fotogramas obtenidos a partir de la grabación de “animación hablar”.

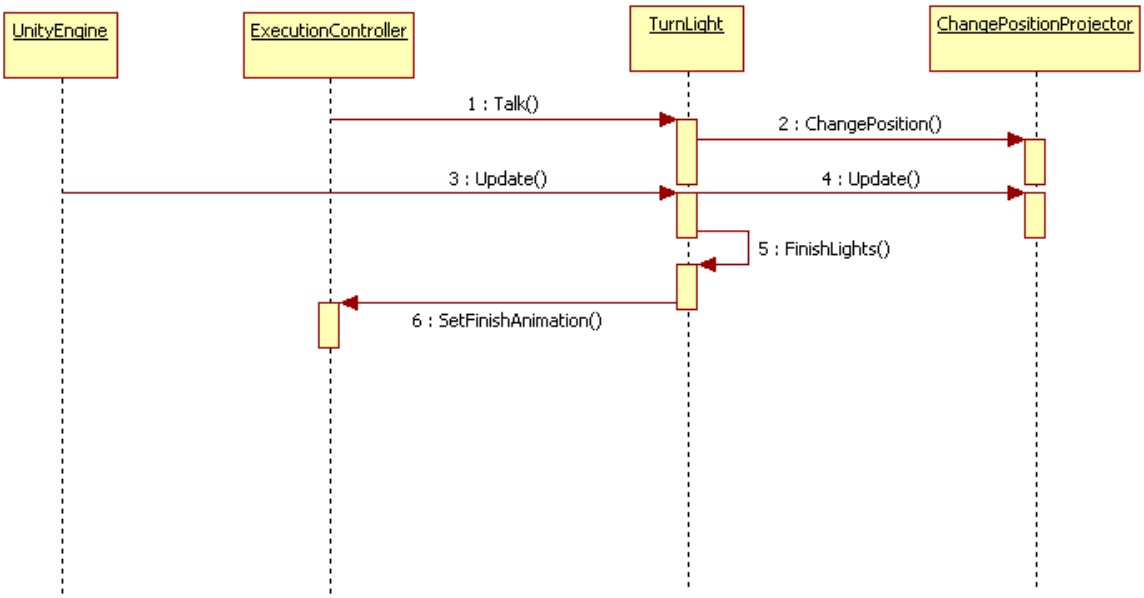


Figura 3.24: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación cargar

En cuanto a la "animación cargar", se accede desde el estado de animación *TurnLight* mediante la transición *TurnLight-Charge*. La animación se basará en el tiempo que tardará la animación en finalizar. Para llevar a cabo la animación, la luz pasará por distintos colores, rojo, amarillo y finalmente verde, que indicarán el estado de la carga, siendo el color rojo el inicio de la carga y el verde el final. Aprovechando las posibilidades de las curvas de animación, se ha utilizado una animación auxiliar para realizar pequeñas modificaciones dinámicas tanto en el color como en la intensidad. Además, también se utiliza la interpolación del movimiento del proyector para dar mayor sensación de actividad. Durante la realización de la animación se ha utilizado tanto la cinemática inversa, como el uso de luces. Terminada la animación, se volverá de nuevo al estado desde el cual puede ser referenciada esta animación que será posible mediante la transición *Charge-TurnLight*. Esta transición se producirá cuando termine la animación.

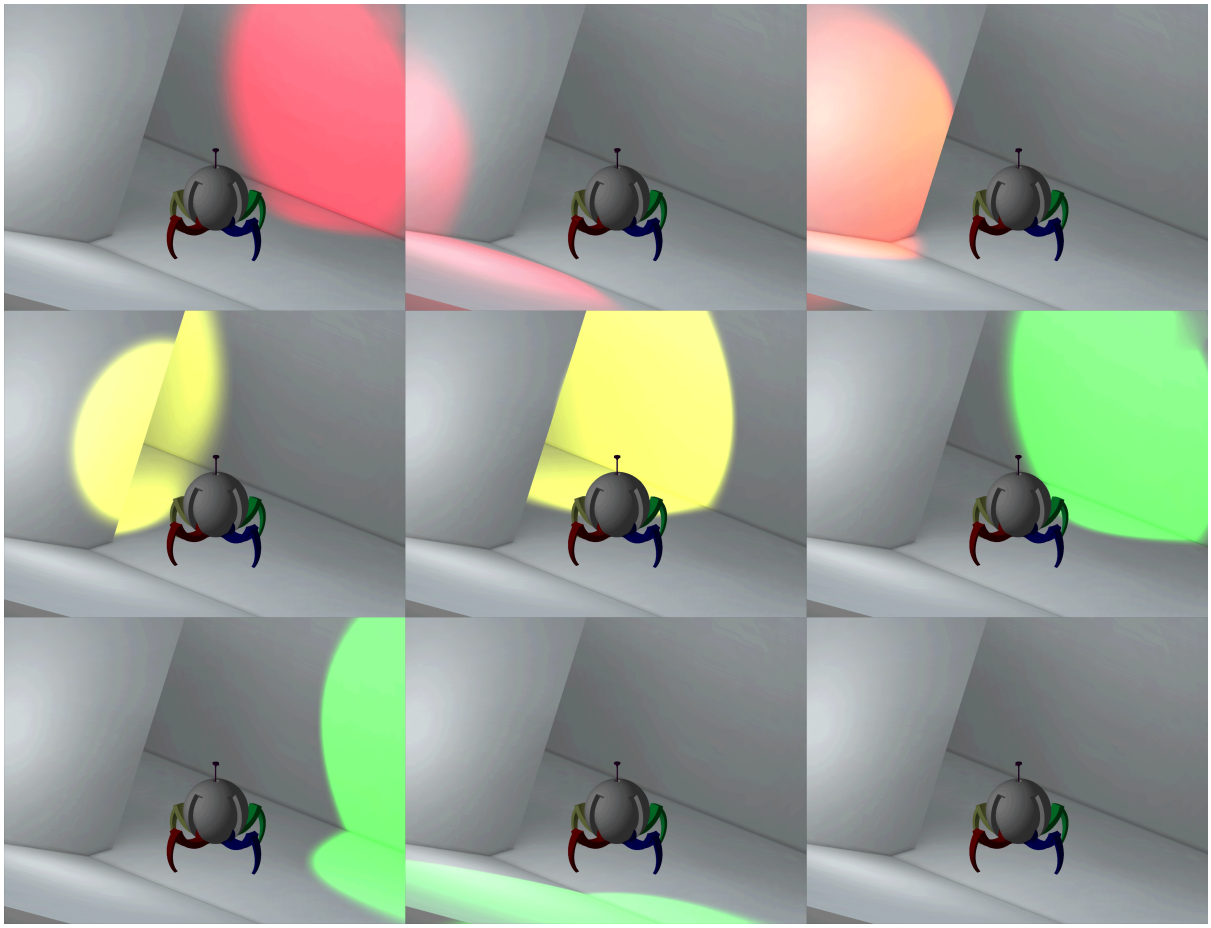


Figura 3.25: Fotogramas obtenidos a partir de la grabación de “animación cargar”.

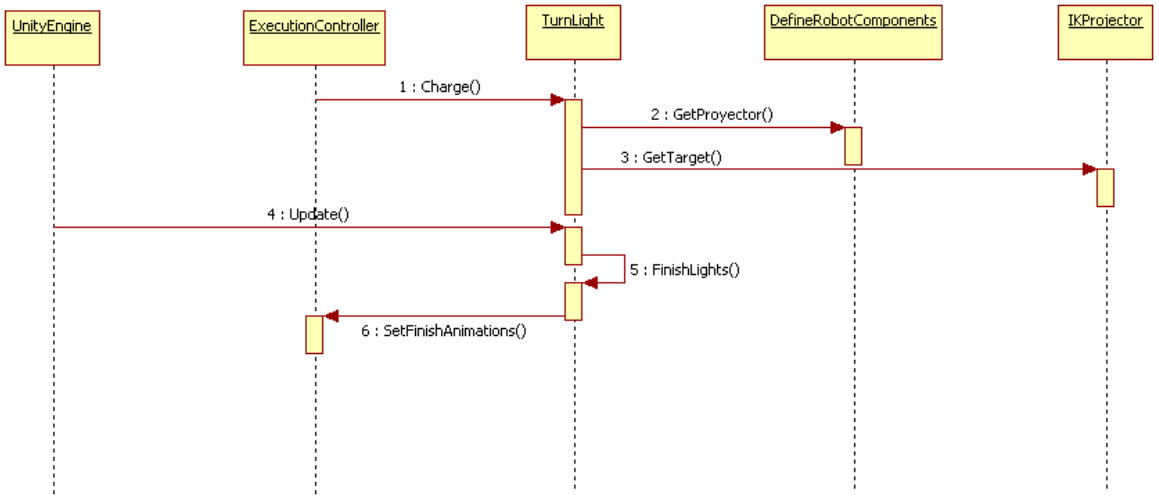


Figura 3.26: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación proyectar

Esta animación se reproduce cuando es invocada por el módulo agente y el grafo de animación se encuentra en *TurnLight*. Desde este estado se pasa a través de la transición *TurnLight-Project* al estado *Project*. Para crear esta animación ha sido necesario utilizar *shaders* y *projectors* proporcionados por la plataforma *Unity*. Además, para reproducir vídeos se utilizan texturas animadas que funcionan como vídeos pero que pueden ser utilizadas sobre materiales. En cuanto a la realización de la animación , utilizando la tecnología mencionada, se solicita al usuario una textura del tipo *MovieTexture* (una textura animada) que se carga en el proyector y se reproduce. También se solicita un tiempo y una posición, que será el tiempo que tarde en apuntar a la zona donde se quiere proyectar. Esta zona es conocida mediante el paso de parámetros al método. Debido a que el vídeo puede ser de larga duración, se consideró insertar un método para detener la reproducción del vídeo. Por último, una vez terminada la animación o parada mediante el método proporcionado, se retornará, pasando por la transición *Project-TurnLight*, al estado *TurnLight*.

Figura 3.27: Fotogramas obtenidos a partir de la grabación de “animación proyectar’.’

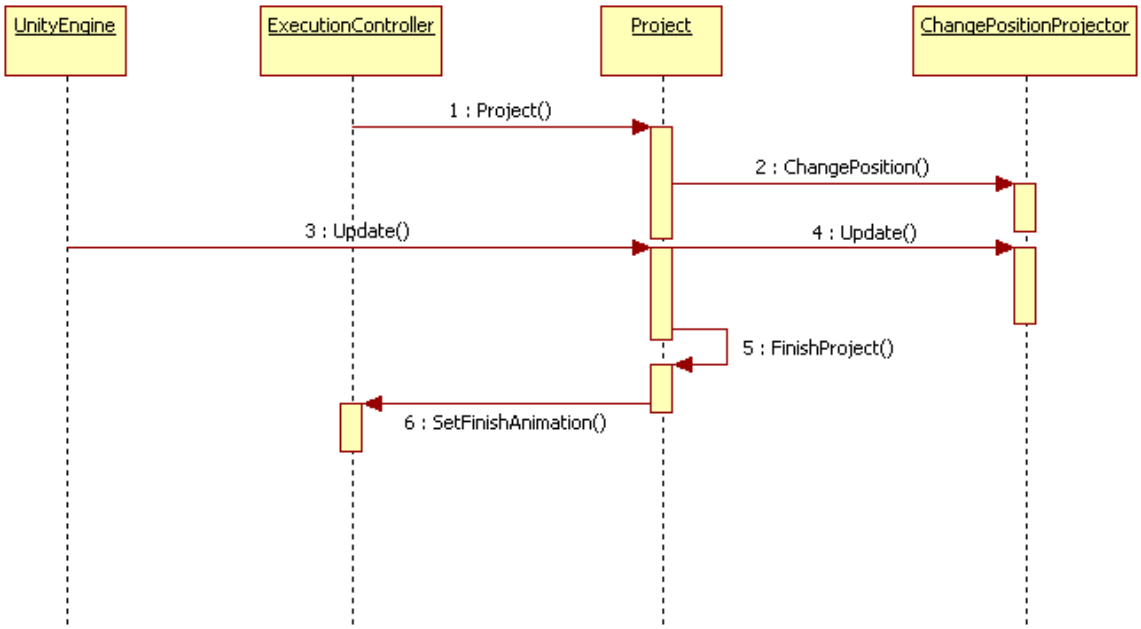


Figura 3.28: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

Animación *idle*

La "animación *idle*" se reproduce cuando es invocada por el módulo agente y se detiene cada vez que pasa del estado *Open* a otro. Esta animación está diseñada para ser activada pasado un tiempo sin realizar ninguna acción. La animación consiste en mover ligeramente el cuerpo, encargándose el sistema de cinemática inversa de adaptar las diferentes partes del cuerpo. Para realizar esta animación se interpola la posición de la Esfera_Robot a partir del tiempo de ejecución.

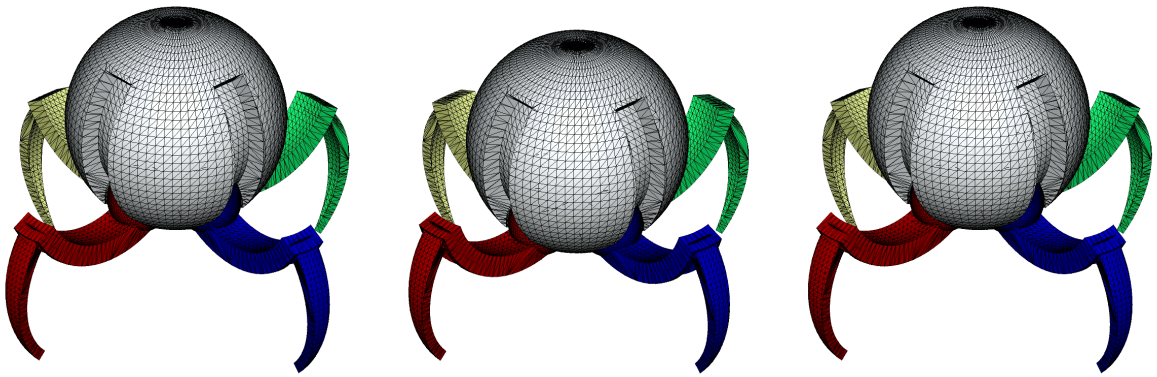


Figura 3.29: Fotogramas obtenidos a partir de la grabación de "animación *idle*".

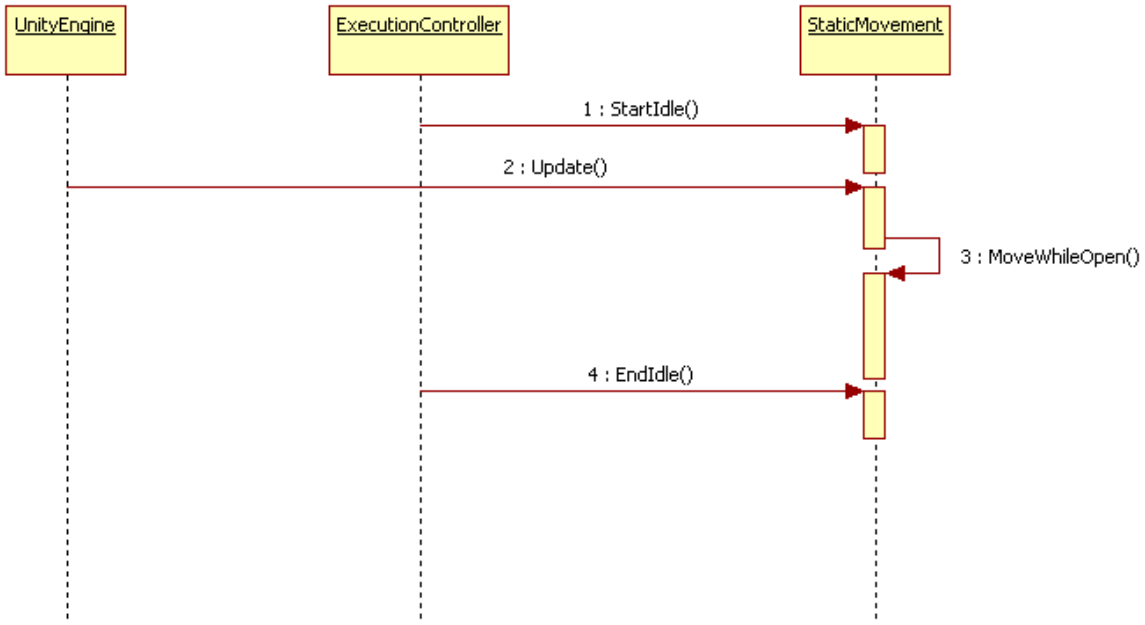


Figura 3.30: Diagrama que muestra la secuencia de interacciones entre los distintos actores que actúan en la animación.

3.4. Interfaz de acceso

La estructura de funcionamiento se basa en el sistema de cinemática inversa, el cual recibe modificaciones procedentes del módulo de animación o comportamientos. El módulo de comportamientos recibe acciones a realizar procedentes de un módulo inteligente (Ver figura 3.31). Para facilitar el uso del componente 3D, se decidió crear una interfaz de uso que no tuviera en cuenta la dificultad de utilizar todos los *script* que componen el módulo de animación.

Además se concluyó que el control del grafo de animación fuese obra de la interfaz ya que sería el elemento que relacionaría todas las animaciones existentes. En caso de no ser una función suya activar alguna de las transiciones, se proporcionarían los métodos necesarios. Por tanto los métodos existentes en la interfaz son:

Método	Uso	Transición a activar	Valores devueltos	Visibilidad
Open	Ejecuta la "animación abrir"	*CloseIdle-OpenIdle*	*True* si se ejecuta, *false* en caso contrario	Pública
Close	Ejecuta la "animación close"	*OpenIdle-CloseIdle*	*True* si se ejecuta, *false* en caso contrario	Pública

Método	Uso	Transición a activar	Valores devueltos	Visibilidad
Walk	Para la "animación andar"	*Walk-OpenIdle*	*True* si se ejecuta, *false* en caso contrario	Pública
CancelWalk	Ejecuta la "animación andar"	*OpenIdle-Walk*	*True* si se ejecuta, *false* en caso contrario	Pública
Mark	Ejecuta la "animación señalar"	*OpenIdle-MarkObjetive*	*True* si se ejecuta, *false* en caso contrario	Pública
Scan	Ejecuta la "animación escanear"	*OpenIdle-Scan*	*True* si se ejecuta, *false* en caso contrario	Pública
TurnLightOn	Ejecuta la "animación encender luz"	No procede	*True* si se ejecuta, *false* en caso contrario	Pública
ModifyLights	Cambia los propiedades de la luz	No procede	*True* si se ejecuta, *false* en caso contrario	Pública
TurnLightOff	Ejecuta la "animación apagar luz"	No procede	*True* si se ejecuta, *false* en caso contrario	Pública
Talk	Ejecuta la "animación hablar"	*TurnLight-Talk*	*True* si se ejecuta, *false* en caso contrario	Pública
SayHello	Ejecuta la "animación hablar" (caso particular)	*TurnLight-Talk*	*True* si se ejecuta, *false* en caso contrario	Pública
Charge	Ejecuta la "animación cargar")	*TurnLight-Charge*	*True* si se ejecuta, *false* en caso contrario	Pública
Project	Ejecuta la "animación proyectar"	*TurnLight-Project*	*True* si se ejecuta, *false* en caso contrario	Pública
StopProject	Para la "animación proyectar"	*Project-TurnLight*	*True* si se ejecuta, *false* en caso contrario	Pública
StaticIdle	Ejecuta la "animación *idle*"	Mantiene el estado *OpenIdle*	Vacío	Pública

Método	Uso	Transición a activar	Valores devueltos	Visibilidad
GetCurrent-State	Compara el estado actual con el nombre pasado por parámetro	No procede	*True* si existe el estado comparado, *False* en caso contrario	Pública
SetFinish-Animation	Cambia el valor del atributo pasado por parámetro	No procede	Vacío	Pública

Tabla 3.1: Tabla explicativa de los métodos que componen la interfaz

3.5. Pruebas

3.5.1. Diseño de pruebas

A la hora de diseñar las pruebas se planteó inicialmente qué se quería probar y cómo obtener los resultados. Las pruebas han sido diseñadas teniendo en cuenta la posibilidades en tiempo de ejecución de *Unity*.

En un principio se decidieron las partes a probar del proyecto, las cuales eran:

1. Funcionamiento de la cinemática inversa.
2. Funcionamiento de las animaciones por separado.
3. Funcionamiento de las animaciones agrupadas.
4. Pruebas de funcionamiento en escenarios de pruebas (Ver apartado 4.1).
5. Evaluación de errores surgidos en integración.

Como se puede observar, las pruebas han sido realizadas progresivamente al mismo tiempo que se realizaba el TFG, empezando desde el primer paso hasta el último. El problema surge a la hora de obtener los datos resultantes de la realización de las pruebas, es decir, no existen resultados numéricos que demuestren empíricamente que los resultados obtenidos de las diversas funcionalidades sean los deseados. Por contra, sí existe la posibilidad de observar y evaluar los movimientos y aceptar o declinar el resultado de las operaciones. Por ejemplo, si una de las animaciones muestra saltos en la animación o realiza una animación incoherente. Cabe reseñar que este tipo de evaluación es dependiente del encargado de evaluar las pruebas. Por tanto a la hora de evaluar las pruebas se han tomado varias hipótesis por las cuales se rigió la evaluación. Además se han realizado pruebas mediante las posibilidades de ejecución de *Unity*, estas son:

- Toda variable perteneciente a un *script* y declarada como pública, *Unity* la muestra en su interfaz y permite modificarla de forma visual y observar su valor actual.

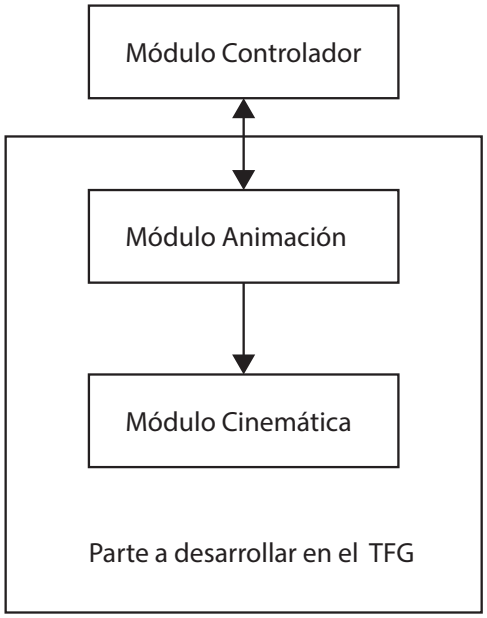


Figura 3.31: Muestra los distintos módulos que componen el proyecto y la comunicación entre ellos.

- *Unity* permite asignar variables durante el tiempo de ejecución.
- *Unity* permite ver la evolución del grafo de animación siendo visible el estado de animación o transición actual y el valor de las variables que se activan o desactivan dependientes de los *script* utilizados.
- Existe la posibilidad de enviar mensajes de depuración a través de la interfaz.
- Durante la ejecución es posible pausarla y reactivarla a deseo del usuario.
- Existen medios de depuración de código integrados en la plataforma.
- Se permite la creación de objetos de depuración visibles durante la ejecución.

3.5.2. Diseño de escenarios

Se han considerado dos escenarios en los que se realizarán las pruebas diseñadas (Ver apartado 3.5.1). El primero de estos escenarios diseñados es el semicomplejo de Miles que fue diseñado durante el semestre anterior a la realización del proyecto (Ver figura 3.32). Las principales características del escenario procedente del proyecto Miles son: [8]

- Es un escenario que simula una posible realidad, es decir, está compuesto por mesas, puertas, ventanas, etc.
- El entorno es un escenario cerrado, en otras palabras, el escenario simula el interior de un edificio.

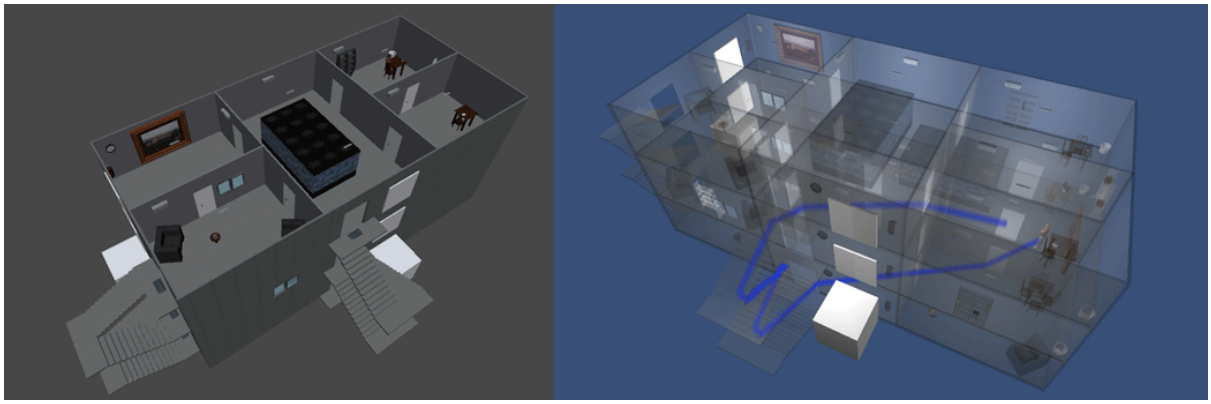


Figura 3.32: Escenario semi-complejo procedente del proyecto Miles.

- Existen cambios de altura.
- La orientación del suelo se mantiene en todo el escenario.
- Se producen cambios de altura a través de ascensores como diferentes escaleras.

También, se pensó en realizar pruebas dentro de un escenario natural, por ejemplo, un bosque o jardín, donde se podrían chequear otras habilidades del robot. El escenario natural está compuesto por otras características, algunas comunes al escenario procedente de Miles y otras nuevas que pertenecen al escenario virtual creado (Ver figura 3.33). Las características del escenario natural son:

- Simulación de una posible realidad natural como árboles, arbustos, piedras, etc.
- El entorno es abierto, es decir, el escenario solamente está acotado por las dimensiones del mismo.
- Existen cambios de altura.
- La orientación del suelo es variable.
- Se producen cambios de altura mediante pendientes, barrancos, o en su defecto obstáculos como troncos, piedras, etc.

La creación del escenario natural se llevó a cabo utilizando la plataforma *Unity* que proporciona herramientas específicas para la generación de escenarios, aunque también se insertaron objetos creados en *3Ds Max*. En la creación se hicieron objetos 3D de baja densidad de polígonos para optimizar la malla y evitar posibles problemas de ejecución debido a la capacidad de procesamiento del hardware del equipo donde se realizarán las pruebas.

Figura 3.33: Escenario que simula un entorno natural.

Capítulo 4

Resultados y conclusiones

4.1. Resultados de pruebas

Se han planteado hipótesis que permitan evaluar las pruebas mediante los medios explicados anteriormente. Como resultado de esta evaluación, se obtienen resultados válidos que demuestran que el funcionamiento se ajusta al deseado. Las hipótesis planteadas son:

- Se busca la precisión tanto en las animaciones como en la cinemática.
- Las animaciones deben ser fluidas y sin errores de visualización (movimientos incoherentes, saltos en la animación, etc.).
- La interacción entre animaciones debe ser la adecuada. Los estados de animación deben estar estrechamente relacionados con la animación correspondiente.
- Cualquier funcionalidad soportada por el componente, debe ser realizada satisfactoriamente.
- Comprobar que no existen errores durante la ejecución.

Entonces a partir de las hipótesis mostradas, se han ido realizando las pruebas expuestas antes y permite obtener los siguientes resultados:

- La cinemática inversa es precisa.
- El funcionamiento de la cinemática inversa es correcto y continuo, siempre y cuando se desee.
- Las animaciones son fluidas, coherentes y sin errores de visualización. El grafo de animación funciona correctamente.
- Las animaciones funcionan tanto por separado como una vez agrupadas, sin entorpecerse.
- Las capacidades proporcionadas al Robot_Ayuda funcionan correctamente y además se le han asignado habilidades que actualmente no se utilizan pero que en futuro podrían ser usadas.

- Se han detectado errores de ejecución debidos a un mal uso del componente por lo que podemos afirmar que corrigiendo dicho mal uso, la ejecución será la correcta.

Particularmente, al realizar la integración con el módulo agente virtual se detectaron nuevos errores que no fueron contemplados inicialmente. Alguno de estos errores son la falta de funciones de manejo (por ejemplo, parar la animación de andar) o usos no contemplados. La resolución de los problemas de integración se han ido realizando gradualmente mientras aparecían. En la actualidad, el prototipo se encuentra en fase de integración con el módulo agente desarrollado por terceros.

Por otro lado, en las pruebas realizadas en escenarios complejos se ha comprobado la ejecución de animaciones y cómo son afectadas por el entorno que les rodea. A estas pruebas se debe la afirmación de que un buen sistema de cinemática inversa permite que el modelo 3D se adapte muy bien a cambios en el escenario adoptando nuevas soluciones para cada momento.

4.2. Conclusiones

La realización del TFG ha dado lugar a conclusiones relacionadas con la temática del mismo. Las conclusiones que han podido ser extraídas son:

- La creación de modelos 3D correctos permite que el resultado final sea muy superior que utilizar modelos de peor calidad.
- La construcción de objetos 3D para una finalidad y características planteadas facilitan altamente el desarrollo del componente 3D.
- La selección de herramientas adecuada, en la producción de un producto, independientemente del campo en el que se encuentra, da lugar a un ahorro de tiempo y aumento de la calidad.
- El planteamiento inicial del proyecto marca un punto importante en el desarrollo del proyecto.
- La utilización de conceptos matemáticos desde los más básicos a los más complejos ayuda altamente a solventar de forma sencilla y rápida los problemas.
- Existe la posibilidad de utilizar medios y métodos procedentes de otros campos en entornos virtuales.
- La mezcla de técnicas adaptadas y elegidas según su finalidad y uso llevan a conseguir mejores resultados.
- Referente al tiempo de planificación hay que contar con posibles imprevistos como errores o sucesos externos.
- La cinemática inversa tiene grandes aplicaciones en el campo de la animación.
- Buenos sistemas de cinemática dan lugar a movimientos realistas que podrían mejorarse añadiendo dinámica.

- Dividir en módulos las distintas partes del proyecto ayuda a una posible reutilización por otros módulos.
- En cuanto a las animaciones, es muy importante identificarlas y diseñarlas después de un estudio previo para optimizar su creación e implementación.
- Del grafo de animación dependerá mucha parte del realismo en la transición de animaciones.
- La consecución de animaciones realistas dependerá del uso de las diversas tecnologías existentes a la hora de animar.
- Al igual que en modelado o en la cinemática es importante buscar los mejores métodos para obtener los resultados deseados.

4.3. Conocimientos adquiridos

Los conocimientos adquiridos durante la elaboración del TFG son:

- Uso del programa de modelado *Solidworks*.
- Uso del programa de modelado *3Ds Max*.
- Uso de la plataforma *Unity3D*.
- Comprensión de conceptos matemáticos.
- Aplicación de conceptos matemáticos.
- Uso de técnicas de cinemática.
- Uso de técnicas de animación.
- Comprensión del estado actual de la animación y la cinemática.
- Creación de escenarios de pruebas.
- Realización de pruebas.
- Técnicas de depuración de código.
- Uso de referentes reales como maquetas para comprender mejor el problema a resolver.
- Uso de herramientas de documentación y diseño.

4.4. Futuras acciones

Aunque los objetivos del proyecto han sido logrados, el desarrollo del componente 3D sigue adelante mejorando aspectos, incluyendo nuevas habilidades y optimización de las funcionalidades existentes. Por tanto, a continuación se exponen algunas propuestas a realizar en un futuro sobre la finalidad del TFG. Las propuestas son:

- Mejorar el modelo 3D para que sea menos restrictivos y aumente su rango de movimiento.
- Creación de nuevas animaciones como correr, saltar, volar, etc.
- Realizar mezcla de animaciones al mismo tiempo, por ejemplo, andar y señalar a la vez.
- Crear diferentes modos de ejecución en las animaciones.
- Ampliar las posibilidades de animación utilizando otros métodos existentes, por ejemplo, captura de movimientos o cinemática directa.
- Optimizar funcionalidades a partir de nuevos conocimientos que pueden ser adquiridos.

Capítulo 5

Anexo

5.1. Animaciones auxiliares

5.1.1. Animación escanear

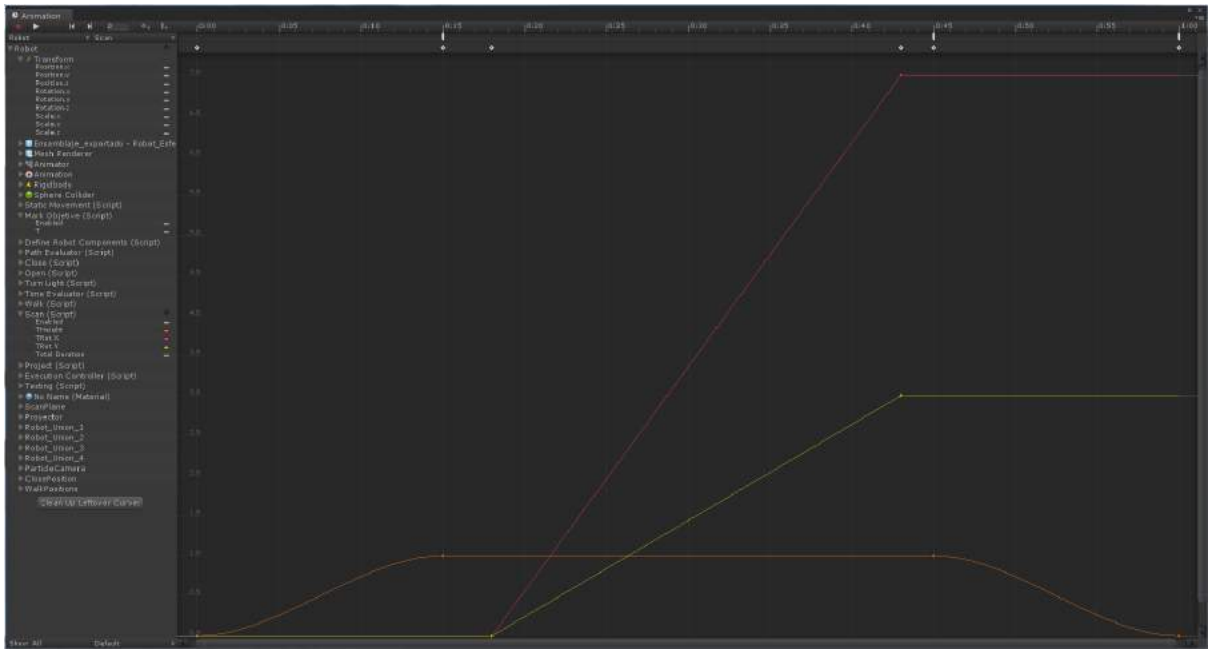


Figura 5.1: Animación auxiliar utilizada para desarrollar "animación escanear".

Animación utilizada durante el subestado escanear. Se dan valores a las tres variables usadas en el *script* que realiza la animación. Las variables son:

1. tHeight: Valor usado para interpolar la altura del Robot_Ayuda
2. tRotX: Valor usado para interpolar la rotación.
3. tRoty: Valor usado para interpolar la rotación.

5.1.2. Animación caminar

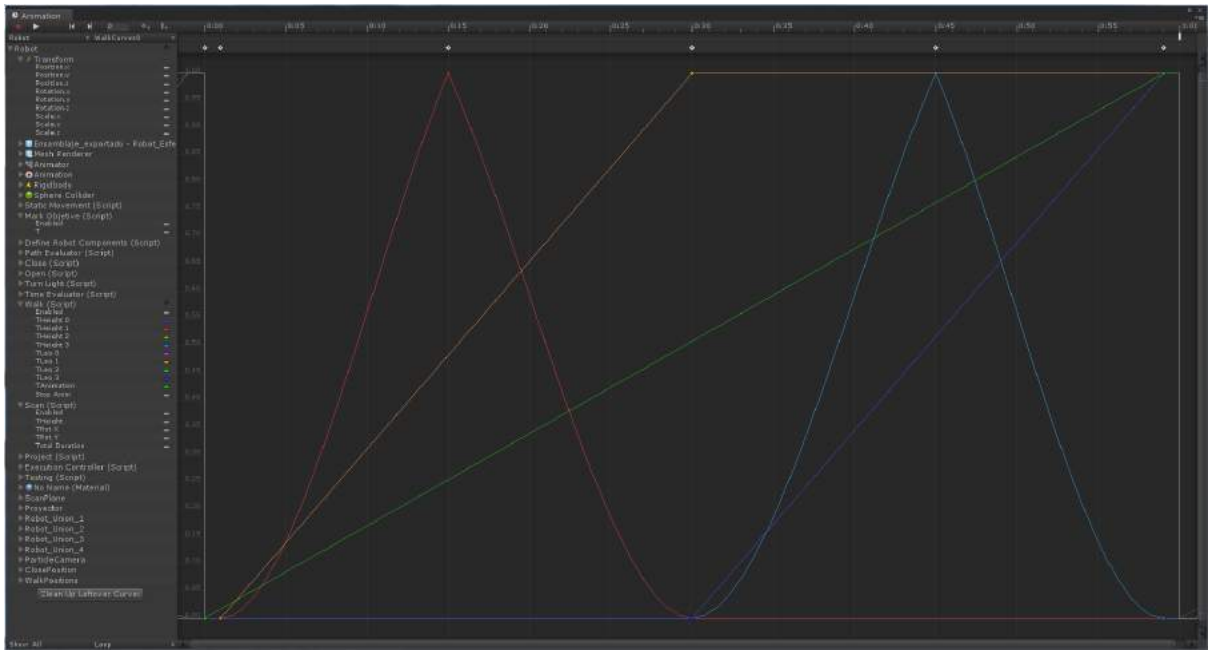


Figura 5.2: Animación auxiliar utilizada para desarrollar “animación andar”.

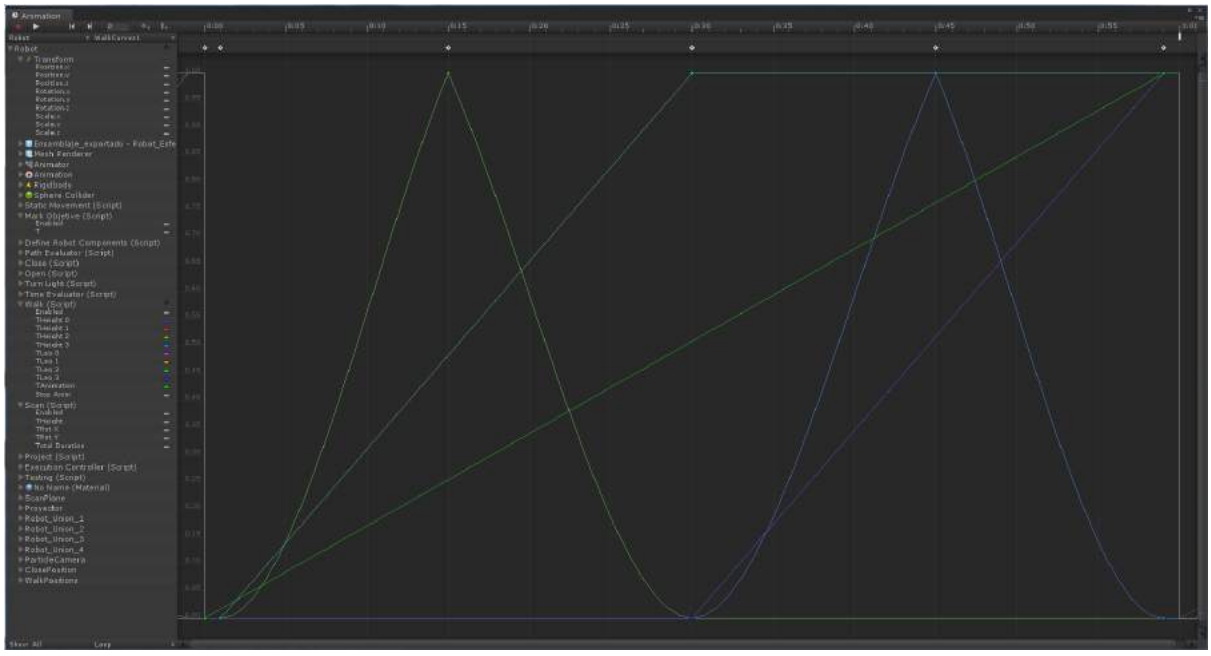


Figura 5.3: Animación auxiliar utilizada para desarrollar “animación andar”.

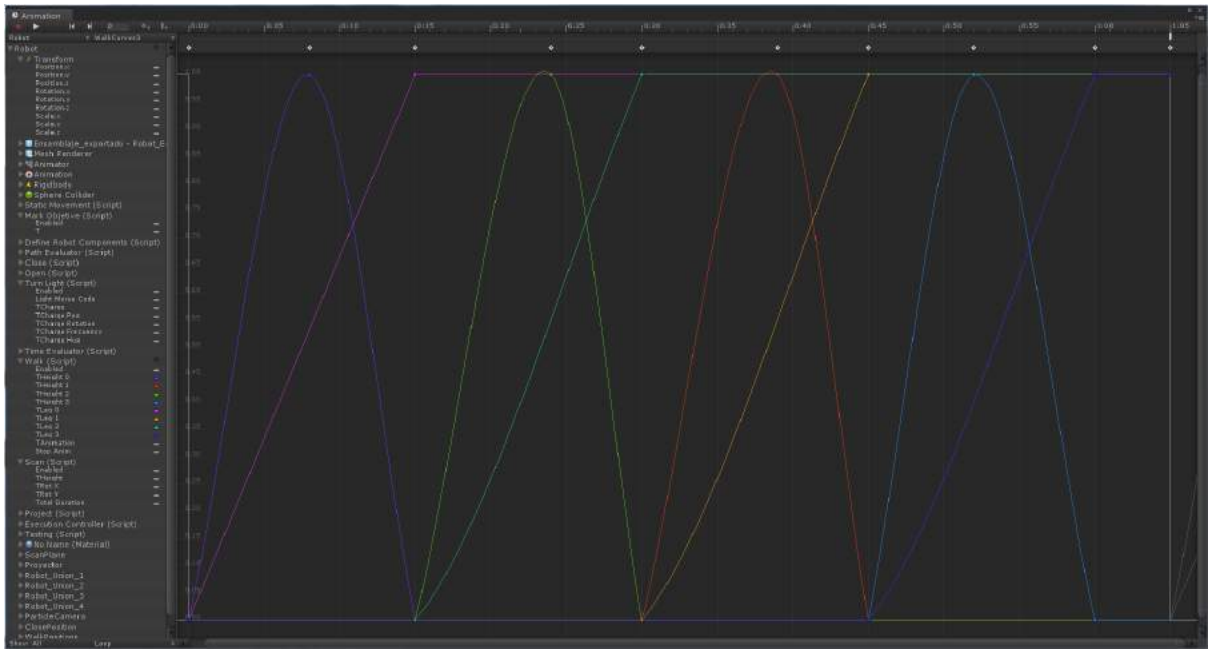


Figura 5.4: Animación auxiliar utilizada para desarrollar “animación andar”.

Animación utilizada durante “animación andar”. Se dan valores a las nueve variables usadas en el *script* que realiza la animación. Las variables son:

- tHeight0: Valor usado para interpolar la altura de Union-1.
- tHeight1: Valor usado para interpolar la altura de Union-2.
- tHeight2: Valor usado para interpolar la altura de Union-3.
- tHeight4: Valor usado para interpolar la altura de Union-4.
- tLeg0: Valor usado para interpolar la posición de Union-1.
- tLeg1: Valor usado para interpolar la posición de Union-2.
- tLeg2: Valor usado para interpolar la posición de Union-3.
- tLeg3: Valor usado para interpolar la posición de Union-4.
- tAnimation: Valor usado para calcular el tanto por ciento de animación reproducida.

5.1.3. Animación señalar

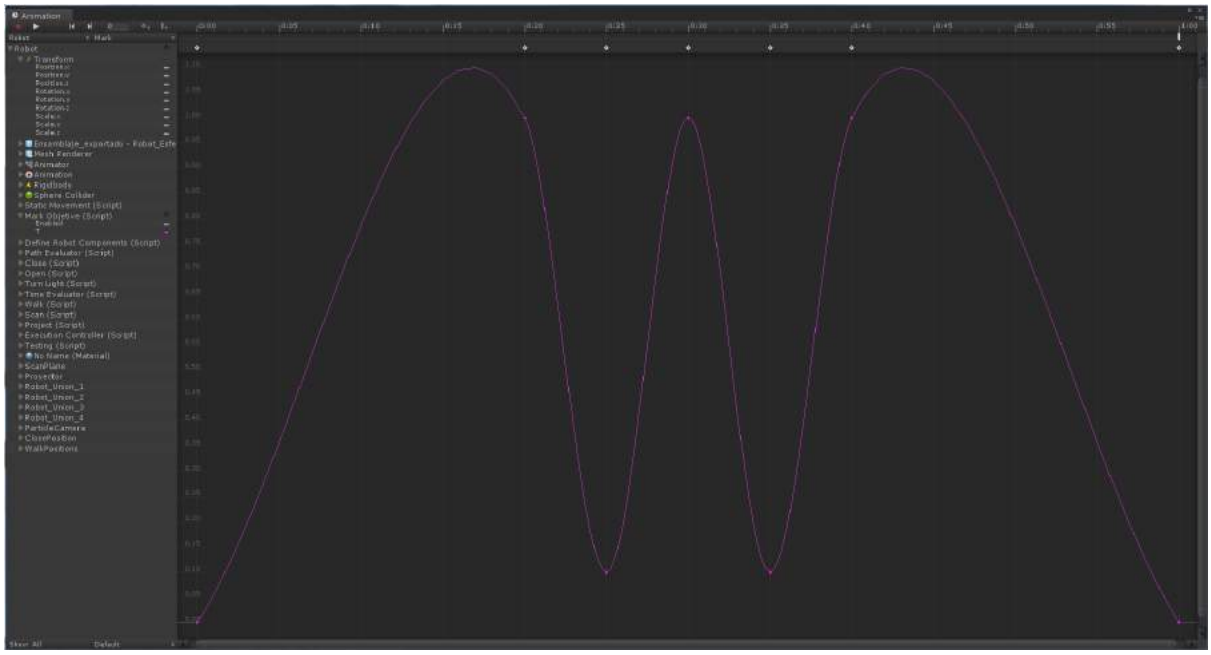


Figura 5.5: Animación auxiliar utilizada para desarrollar “animación señalar”.

Animación utilizada durante “animación señalar”. Se da valor s a la variable usada en el *script* que realiza la animación. La variable es:

- t: Valor usado para interpolar la posición de Union-Y.

5.1.4. Animación hablar

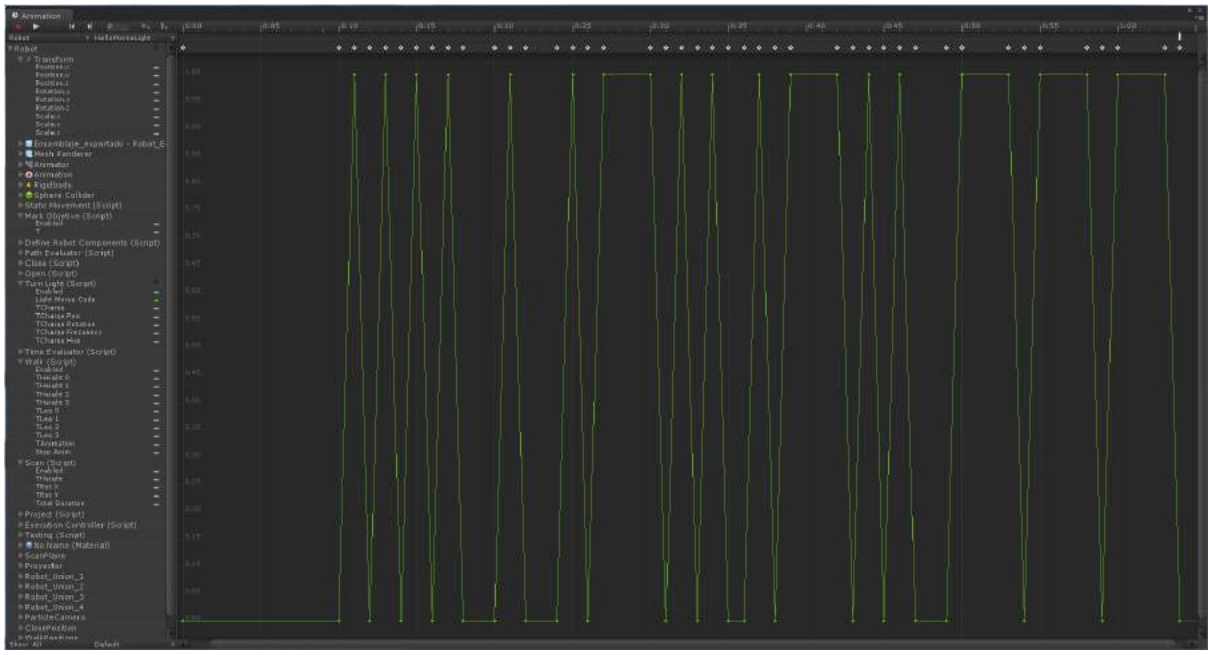


Figura 5.6: Animación auxiliar utilizada para desarrollar "animación hablar".

Animación utilizada durante "animación hablar". Se da valor s a la variable usada en el *script* que realiza la animación. La variable es:

- lightMorseCode: Valor usado para codificar una señal Morse.

5.1.5. Animación cargar

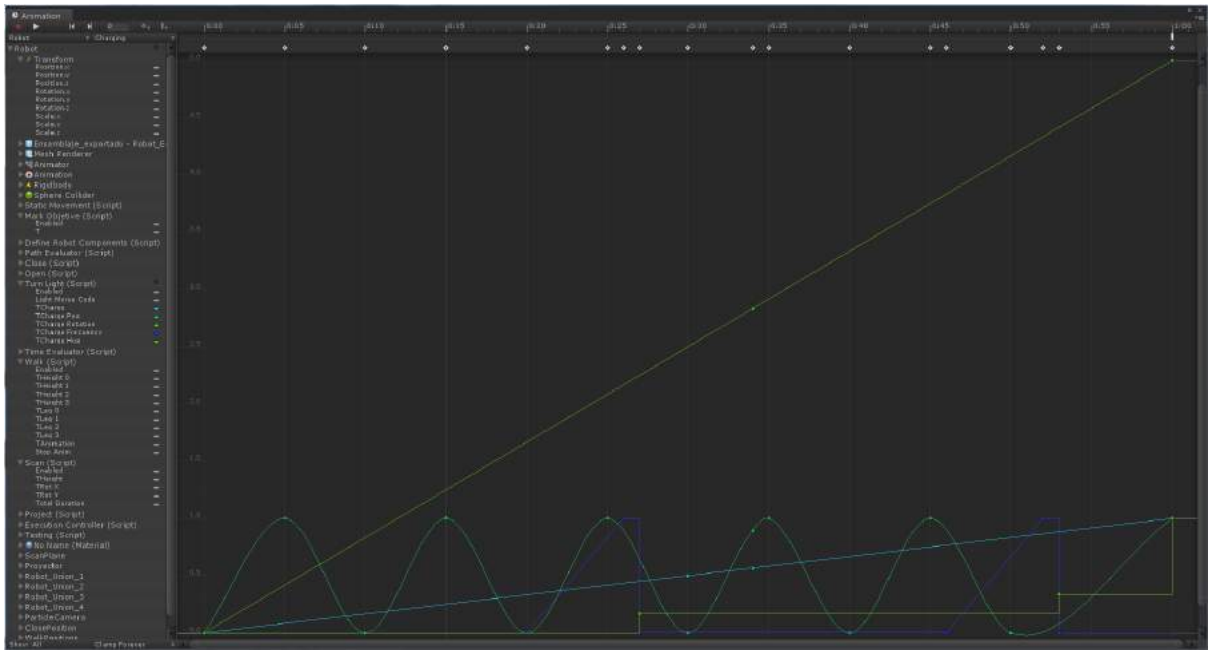


Figura 5.7: Animación auxiliar utilizada para desarrollar "animación cargar".

Animación utilizada durante "animación cargar". Se dan valores a las cinco variables usadas en el *script* que realiza la animación. Las variables son:

- tCharge: Valor usado para calcular el tiempo de carga.
- tChargePos: Valor usado para interpolar la altura del Proyector.
- tChargeRotation: Valor usado para interpolar la rotación de Proyector.
- tChargeFrecuency: Valor usado para insertar variaciones en el colore intensidad de la luz.
- tChargeHue: Valor usado para cambiar el color de la luz.

5.2. Corrección de ángulos

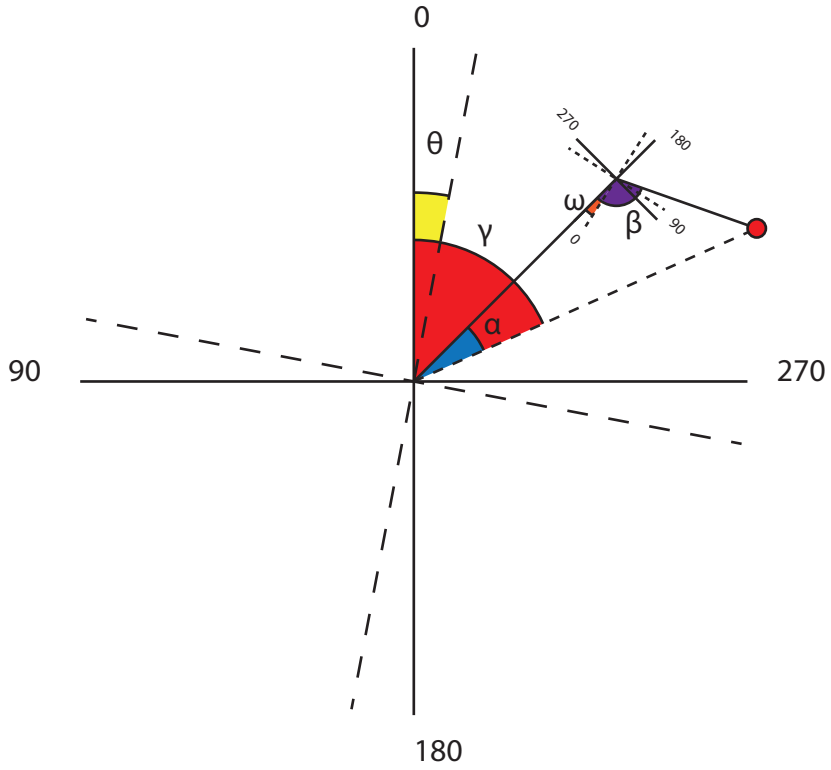


Figura 5.8: Muestra los ángulos calculados y utilizados en el desarrollo de cinemática inversa.

Los ángulos de corrección son:

1. θ: Se produce al ser necesario compensar el desfase que existe entre el eje de coordenadas de Union-Y y Seccion-1_Y. Ya que en Seccion-1_Y se aplicará la rotación.

2. ω: Se produce al ser necesario compensar el desfase que existe entre el eje de coordenadas de Seccion-1_Y y Seccion-2_Y. Ya que en Seccion-2_Y se aplicará la rotación.

En el caso del ángulo final aplicado en las correspondientes Seccion-1_Y solo interferirá θ.

Por otro lado en el ángulo final aplicado en las correspondientes Seccion-2_Y interferirán θ (desfase arrastrado) y ω.

Bibliografía

[1] Eike F Anderson. Real-time character animation for computer games. Master's thesis, National Centre for Computer Animation Bournemouth University, 2001, http://ncca.bournemouth.ac.uk/gallery/view/220/Real-Time_Character_Animation_for_Computer_Games. 2.3, 2.5

[2] Paolo Baerlocher. *Invere Kinematic Techniques for the Interactive Posture Control of Articulated Figure*. PhD thesis, Ecole Polytechnique Federale de Lausenne, 2001, http://www0.cs.ucl.ac.uk/research/equator/papers/Documents2002/Paolo%20Baerlocher_Thesis_2001.pdf. 2.4, 2.4.1

[3] Roman Berka Lukas Barinka. Inverse kinematics - basic methods. *Technische Universitaet Wien*, 2002, http://www.cg.tuwien.ac.at/hostings/cescg/CESCG-2002/LBarinka/paper.pdf. 2.4

[4] JP. Merlet. Kinematics' not dead! *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference*, 1:1–6, 2000, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=844031. 2.4.1

[5] Diego Park. 1. introduction inverse kinematics, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.133.3247&rep=rep1&type=pdf. 1.2

[6] Raymond A. Serway. *Física*. Interamericana S. A., 1985. 2.2.7, 2.2.8

[7] Jerrold E. Marsden Anthony J. Tromba. *Cálculo Vectorial*. Pearson Educación S.A., 2004. 2.2.1, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 2.2.6

[8] Carlos Castillo Venegas. Memoria practicum. Memoria sobre el Prácticum realizado en el laboratorio Decoroso Crespo de la Universidad Politécnica de Madrid. 3.1.1, 3.1.2, 3.1.3, 3.5.2

[9] Charles W. Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *Systems, Man and Cybernetics, IEEE Transactions*, Volume:16 , Issue: 1:93–101, Jan. 1986, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4075580. 2.4.1

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Feb 14 19:04:44 CET 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)