Universidad de las Ciencias Informáticas
Vertex, Entornos Interactivos 3D, Facultad 5

Módulo de captura de movimiento para el videojuego Danzo-Terapia

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Ernesto Antonio Leyva Piñeda

Tutor: Ing. Juan Carlos Ayala Alonso

Co-tutor: Ing. Rodobaldo Livan Saroza Ramírez

VERTEX
ENTORNOS INTERACTIVOS 3D

La Habana, 2015

La felicidad consiste en tener algo que hacer,
alguien a quien amar y algo que esperar.
Benjamin Franklin.

Dedicatoria

A mis padres y a mi hermano.

Agradecimientos

A mis padres, que con tantos sacrificios me han ayudado a llegar hasta aquí.

A mi hermano, por su confianza y sus tantas sugerencias.

A mi familia, que tanto me ha apoyado.

A Midiala, que siempre ha estado a mi lado, y que tanto me ayudó a terminar este trabajo.

A Erduin, con quién he compartido estos cinco años de la carrera, y en quién siempre he encontrado una mano amiga.

A Leonar y Reynier, que tantas veces dejaron su tesis para ayudarme.

A Orson, Marin y Mirnerys, en quienes he encontrado una magnífica compañía.

A todas las personas que conocí en esta universidad y que ocuparon una parte muy importante de mi vida.

A todos, muchas gracias.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Ernesto Antonio Leyva Piñeda
Autor

Ing. Juan Carlos Ayala Alonso
Tutor

Ing. Rodobaldo Livan Saroza Ramírez
Co-tutor

Resumen

El presente trabajo está dirigido a la creación de un módulo de captura de movimiento mediante el uso del sensor *Kinect* para el videojuego con fines terapéuticos Danzo-Terapia. Para que este videojuego pueda realizar el proceso de rehabilitación completo, necesita conocer las acciones realizadas por los pacientes durante la ejecución de los ejercicios que propone.

Para la realización de este trabajo se hizo un estudio de los principales conceptos relacionados con la captura de movimiento, su aplicación en terapias de rehabilitación, así como del sensor *Kinect*. Para el desarrollo del módulo se utilizó la biblioteca de interacción *"Kinect for Windows"* creada por *Microsoft* y el motor de videojuegos *Unity 3D*. Además, se utilizó la metodología ágil de desarrollo de *software* Programación Extrema para guiar todo el proceso de creación de la solución propuesta.

Como resultado de este trabajo se obtuvo un módulo que permite obtener la posición y orientación de las principales partes del cuerpo durante la realización de un ejercicio. A este módulo se le puede configurar su tasa de grabación para modificar la frecuencia con que son obtenidos los datos. Además brinda la posibilidad de guardar toda la información capturada para ser utilizada como patrón de comparación por el videojuego y así reconocer si los pacientes realizan los ejercicios correctamente.

Palabras clave: Captura de movimiento, *Kinect*, videojuego.

Índice general

Índice de figuras

Índice de tablas

Lista de códigos fuentes

Introducción

La informática es una ciencia en continua evolución que con el transcurso de los años se ha aplicado a casi todos los sectores sociales. El aumento del conocimiento acumulado sobre esta y el creciente desarrollo tecnológico de las Tecnologías de la Informática y las Comunicaciones (TIC), le ha dado un nuevo giro al mundo, proporcionando nuevas oportunidades al desarrollo científico. Uno de sus aportes, la Realidad Virtual (RV), mejoró la interacción de los usuarios con los sistemas informáticos, generando un nuevo horizonte para la aplicación de las TIC en la vida cotidiana.

La RV puede verse como la interfaz hombre-máquina, que permite al usuario sumergirse en una simulación gráfica tridimensional (3D) generada por ordenador, además de navegar e interactuar en ella en tiempo real. Su potencial está determinado por su capacidad para estimular y engañar a los sentidos del usuario (visión, audición, tacto, gusto y olfato); es por eso que la RV ideal sería aquella que hiciera una reproducción completa de ellos.

Uno de los campos de aplicación de la RV en los últimos años, gracias al creciente desarrollo tecnológico, ha sido el de la medicina. Esto ha beneficiado al hombre, especialmente al paciente, significando una mejora trascendental en su calidad de vida, permitiendo realizar un diagnóstico más certero por parte de los doctores. Su aplicación en la realización de cirugías y autopsias virtuales y más recientemente en el tratamiento de fobias, ha constituido una ayuda indiscutible en este campo. Esta tecnología emerge como una terapia adicional que promete ayudar a las personas a superar traumas físicos o cognitivos, sin enfrentar la frustración que les produce el contacto directo con la realidad.

La RV también ha tenido un amplio desarrollo en el campo de la neurociencia en lo que respecta a los procesos de rehabilitación física. Esta novedosa terapia ha sido utilizada para optimizar procesos de aprendizaje o reaprendizaje de patrones de movimiento en personas con ictus cerebral, deformación perceptiva-motora, lesión cerebral adquirida, enfermedad de Parkinson, rehabilitación ortopédica, entre otras.

Según la Organización Mundial de la Salud (OMS), el 15 % de la población mundial está afectada por alguna discapacidad física, psíquica o sensorial que dificulta su desarrollo personal y su integración social, educativa o laboral.[1] Tal porcentaje equivale a más de mil millones de personas con alguna desventaja notoria en comparación con las demás. Existe por lo tanto, una creciente preocupación mundial por eliminar, hasta donde sea posible dichas desventajas y cuando no sea posible la completa recuperación, compensarla con la rehabilitación. Es aquí donde la RV puede ser una alternativa para facilitar el proceso.

La rehabilitación con RV se basa en fundamentos científicos relacionados con el aprendizaje motor. Para

[1] OMS, Nota descriptiva 352, Diciembre 2014

que esta se pueda desarrollar de manera exitosa, es necesario tener en cuenta aspectos como la repetición de los ejercicios, la motivación del paciente y la retroalimentación (Holden, 2005). Para aprender a realizar un movimiento, este debe ser ejecutado repetidamente para su memorización. Además, debe existir un buen grado de motivación para que el proceso pueda desarrollarse de forma satisfactoria, lo que puede lograrse al aplicar la RV a las terapias, ya que la forma de presentar los ejercicios puede personalizarse para resultar agradable al usuario. Una forma de personalizar los ejercicios es a través de los videojuegos; en estos se plantean distintas metas a cumplir, las cuales dan una retroalimentación al usuario de estar realizando un movimiento de forma correcta o no. Si tales objetivos son planteados correctamente entonces representan un reto al usuario, generando una motivación para lograr el objetivo. Esto permite realizar las repeticiones necesarias para estimular la reorganización cerebral y así, memorizar el movimiento. Estos procesos se centran en el concepto de "Aprendizaje por imitación" (Suárez y Ramírez, 2012), donde con la ayuda de un profesor virtual, los sistemas le permiten al usuario volver a entrenar una amplia variedad de movimientos.

Estos tipos de videojuegos, diseñados para un propósito distinto al de la pura diversión, son conocidos como videojuegos serios. Dentro del contexto de estos, existe un tipo específico que busca fomentar la actividad física mediante diferentes roles, los *Exergames* o videojuegos de ejercicio. Estos son videojuegos interactivos que buscan hacer de la actividad física algo gratificante, generando en cada persona motivación suficiente durante el tiempo empleado para hacer ejercicios, estimulando la movilidad de todo el cuerpo mediante el uso de ambientes interactivos.

Una de las líneas productivas del Centro de Entornos Interactivos 3D (VERTEX) de la Universidad de las Ciencias Informáticas (UCI) son los videojuegos, algunos de ellos están enfocados a la rehabilitación de pacientes con enfermedades perceptivo-motoras, como es el caso de Meteorix y Danzo-Terapia. Este último surge con el propósito de ayudar en la recuperación de personas con enfermedades físico-motoras y consiste en un mundo virtual en el cual un personaje ficticio realiza diversos ejercicios, que deben ser imitados por los pacientes, cuya ejecución es diagnosticada posteriormente por un especialista. Actualmente este juego no es capaz de evaluar por sí mismo los ejercicios realizados por el enfermo, por lo que es imprescindible la presencia de un especialista que realice la evaluación. El hecho de que el videojuego por sí solo no pueda realizar el proceso completo de rehabilitación, atenta contra el objetivo por el cual fue creado. El principal motivo por el cual Danzo-Terapia no puede supervisar ni evaluar las rutinas de los pacientes, es porque no es capaz de retroalimentarse de las acciones que estos realizan al efectuar los ejercicios.

Dada la problemática planteada anteriormente, surge el siguiente **problema de investigación** a resolver, *¿Cómo obtener los datos de las acciones realizadas por los pacientes durante la ejecución de los ejercicios del videojuego Danzo-Terapia?*

Para dar solución al problema anterior se propone como **objeto de estudio** *los métodos de captura de movimiento*, y como **campo de acción** *la captura de movimientos humanos durante un proceso de rehabilitación*. Planteándose como **objetivo general** *desarrollar un módulo para el videojuego Danzo-Terapia que permita la obtención de los datos de las acciones realizadas por los pacientes*.

Para lograr dicho resultado se proponen las siguientes **tareas de investigación**:

- Elaboración del marco teórico de la investigación.
- Caracterización de los sistemas de captura de movimiento.
- Selección de un dispositivo de captura de movimiento.
- Selección de la biblioteca de integración para el dispositivo seleccionado.
- Diseño e implementación del módulo de captura de movimiento.
- Comprobación del desempeño del módulo propuesto mediante las pruebas de aceptación.

Para dar cumplimiento a estas tareas de investigación se emplearon **métodos científicos teóricos y empíricos**.

Métodos Teóricos:

- Histórico-Lógico: Este método teórico se utilizará para realizar el estudio del estado del arte. Además, para conocer los métodos de captura de movimiento.
- Analítico-Sintético: Se utilizará para estudiar cómo la captura de movimiento puede ser usada en un proceso de rehabilitación.
- Modelación: Este método se utilizará para crear un prototipo funcional del módulo.

Métodos Empíricos:

- Análisis documental: Para seleccionar la información necesaria para la construcción del marco teórico.
- Pruebas: Para comprobar el desempeño del módulo elaborado.

El presente documento está estructurado en 3 capítulos. En el Capítulo 1 se presentan los elementos teóricos que sirven de base a la investigación del problema planteado. En el Capítulo 2 se propone un método de solución basado en la captura de movimiento. Finalmente en el Capítulo 3 se muestran los resultados del módulo desarrollado, y se exponen las pruebas que se le realizaron.

CAPÍTULO 1

Fundamentación Teórica

Introducción

Para un correcto entendimiento de la solución que se propone, en el presente capítulo se documentan los conceptos fundamentales que son la base teórica de dicha propuesta. De esta forma se presentan tópicos que hacen alusión a la captura de movimiento, los principales sistemas que son empleados para este proceso y los diferentes sectores en que se puede utilizar. También se analizan los dispositivos disponibles en el Centro de Entornos Interactivos 3D (VERTEX) para capturar movimiento. Por último se realiza un desglose de las herramientas y metodologías a utilizar en el desarrollo de la solución que se presenta.

1.1. Captura de movimiento

La captura de movimiento, también conocida como *"Motion Capture"*, es una técnica de grabación de movimiento que consiste en el registro de la posición y orientación de personas u objetos. Esta se realiza durante un determinado tiempo en el que se obtienen las coordenadas de diversos puntos claves, para luego trasladar estos datos a un modelo digital. Este proceso suele realizarse mediante sistemas ópticos, inerciales, electromecánicos o electromagnéticos, con el uso de trajes especiales, cámaras, entre otros (Ortega et al., 2001).

Su utilización en la actualidad se ha ampliado a varios sectores como el de la medicina, el deporte, los videojuegos y la industria cinematográfica. En la medicina se ha utilizado en la recopilación de la información necesaria sobre locomoción, para el diseño de diferentes prótesis y la detección de errores al realizar algún ejercicio. En el deporte se ha empleado para mejorar el desempeño de los atletas, determinando posibles gestos que pudiesen disminuir sus resultados. Otra esfera en la que ha tenido un gran auge esta técnica es en la industria del entretenimiento, utilizándose en la realización de películas animadas, como *Las aventuras de Tintín (Figura 1.1)* o en las que aparecen personajes animados, como es el caso de *Gollum* en *El señor de los anillos (Figura 1.2)*. Dentro de esta industria, los videojuegos que incluyen personajes suelen aplicar la captura de movimiento, no solo para lograr que las acciones de estos sean lo más realista posible, mediante la

realización de un exhaustivo análisis de sus movimientos; sino también para brindar la posibilidad al usuario de actuar como controlador del videojuego mediante movimientos de su cuerpo.

Figura 1.1. Jamie Bell y Andy Serkis dándole vida a Tintín y Haddock.

Figura 1.2. Andy Serkis interpretando a *Gollum*.

1.1.1. Sistemas ópticos de captura de movimiento

Estos sistemas necesitan ser ubicados en amplios *sets*, en los que se utilizan dos o más cámaras sincronizadas entre sí para obtener proyecciones simultáneas de un objeto, las cuales son superpuestas para triangular la posición de este. Para facilitar el proceso, estos sistemas suelen hacer uso de marcadores pasivos o activos. Los marcadores pasivos *(Figura 1.3)* están recubiertos de materiales reflectantes y se pegan a la persona que realiza los movimientos en puntos estratégicos. Estos reflejan la luz que es generada cerca de las lentes de las cámaras, las cuales pueden configurarse con un umbral de luz pequeño, de tal forma que solo recojan la luz reflejada por los indicadores, y no la luz reflejada por la piel de la persona. El proceso de captura con estos marcadores puede ser lento, requiriendo la intervención de un operador para refinar los datos capturados y solucionar los problemas presentados durante la captura. Los marcadores activos *(Figura 1.4)* están conformados por LED, con esto se logra aumentar la distancia a la que se puede desplazar el sujeto. Para determinar la posición de los indicadores se ilumina uno en cada instante de tiempo a una frecuencia muy

alta, o varios a la vez, calculando la identidad de cada indicador a partir de su posición relativa. El uso de estos marcadores proporciona un mejor rendimiento para capturas en tiempo real.

Estos sistemas suelen necesitar más de dos cámaras para poder seguir el movimiento de todo el cuerpo. Además, la utilización de varias cámaras sirve para mitigar los errores producidos durante la identificación de los marcadores, ya que la identidad de estos puede confundirse con la de otros marcadores cercanos. Para que este proceso sea confiable y no requiera de un largo proceso de posproducción es necesario utilizar cámaras de mejores prestaciones, que posean una mayor resolución y velocidad de captura que las tradicionales *webcam* (Beltrán Álvarez et al., 2004; Vlasic et al., 2007). El montaje de un *set* con este tipo de cámaras es altamente costoso ya que el precio de cada cámara oscila entre los 600 y 6.000 dólares (OptiTrack, 2015).

Figura 1.3. Captura de movimiento facial mediante marcadores pasivos.

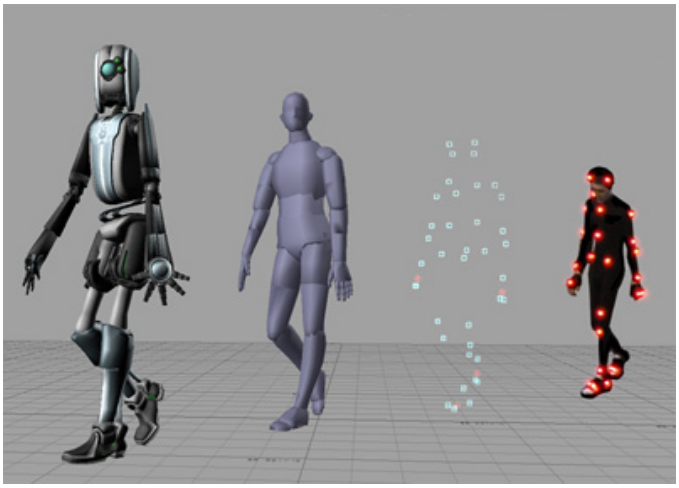

Figura 1.4. Captura de movimiento mediante LED.

Sin embargo, existen otras cámaras desarrolladas por compañías como *Sony Computer Entertainment* y *Microsoft* para sus videoconsolas *PlayStation* y *XBox* respectivamente, con el objetivo de permitir la interacción de los usuarios con las consolas. Entre estas cámaras se encuentra el *Eye Toy*, el *PlayStation Eye*

y *PlayStation Camera* de *Sony*, además de *XBox Live Vision* y *Kinect* de *Microsoft*, los cuales pueden ser empleados en una computadora, sin necesitar de la consola, para reconocer movimientos del cuerpo humano.

Eye Toy

El *Eye Toy* es una cámara digital similar a una *webcam* para *PlayStation 2* (*PS2*). Este surge con el objetivo de habilitar una interfaz de usuario vinculada a las aplicaciones de los videojuegos en tiempo real, mediante el uso de una cámara de bajas prestaciones. Utilizando el poder computacional de la *PS2*, se implementaron las tecnologías de visión computarizada y reconocimiento de gestos que utiliza este dispositivo. Este fue liberado en 2003, teniendo una amplia y rápida aceptación, ya que permitía a los jugadores interactuar con los videojuegos mediante movimientos de su cuerpo o comandos de voz. La cámara de los primeros dispositivos de este tipo fueron creadas por la compañía *Logitech* y las posteriores por *Namtai*. Cuenta además con dos LED; uno azul, que se enciende cuando la consola está encendida, indicando que está listo para usarse y uno rojo, que titila cuando no hay suficiente luz en la habitación en que se está utilizando el *Eye Toy*, siendo esta su principal limitación. Dado que el *Eye Toy* es esencialmente una cámara, este necesita poder ver al jugador, por lo que requiere ser usado en una habitación bien iluminada (Marks, 2010).

Figura 1.5. *Eye Toy*.

PlayStation Eye

PlayStation Eye es el sucesor del *Eye Toy* para *PlayStation 3* (*PS3*), lanzado en 2007. La cámara del *PlayStation Eye* es capaz de grabar video a una taza de 60 cuadros por segundo o *Frames per second* (*FPS*) a una resolución de 640 por 480 píxeles y de 120 *FPS* para 320 por 240 píxeles. Fue diseñada para ser el doble de sensible que la de su predecesor. Con la colaboración de la compañía *OmniVision Technologies* se creó un sensor más efectivo en condiciones de baja iluminación. Al *PlayStation Eye* se le incorporaron cuatro micrófonos para permitir la grabación de voz multidireccional, cancelación de eco, supresión de ruido, así como brindar la posibilidad a los usuarios de charlar con otros jugadores. A partir del 2009, con la aparición del *PlayStation Move*, un mando con sensores inerciales y una esfera lumínica en su parte posterior, cuya posición 3D es determinada por el *PlayStation Eye*, se le incluyeron nuevas características a este. Entre ellas se incluyen el reconocimiento de comandos de voz en diversos idiomas, reconocimiento facial, detectar cuando una persona sonríe, así como estimar su edad y género. Además de los usos orientados

a los videojuegos, *Sony* le incluyó la posibilidad de establecer conversaciones interactivas, con audio y video, entre los usuarios de la *PlayStation Network* (LLC, 2014; Stocker, 2007).

Figura 1.6. *PlayStation Eye*.

PlayStation Camera

PlayStation Camera es un accesorio opcional de detección de movimiento, sucesor del *PlayStation Eye* para *PlayStation 4* (*PS4*). Esta incluye dos cámaras de 1280 por 800 píxeles y abarca un campo de visión de 85°. La doble configuración de la cámara permite diferentes modos de operación, dependiendo de la aplicación de destino, y ambas cámaras pueden utilizarse para la detección de la profundidad de los objetos dentro de su campo de visión. Por otra parte, una de las cámaras se puede utilizar para la generación de la imagen de video, mientras que la otra se utiliza para la captura de movimiento (Inc., 2013).

Figura 1.7. *PlayStation Camera*.

XBox Live Vision

XBox Live Vision es una cámara de videojuego creada por *Microsoft* para su consola *XBox 360* y liberada en el 2006. Este accesorio tiene un reducido tamaño, y goza de una firme sujeción que le permite adaptarse perfectamente a cualquier superficie, lo que posibilita situarla en el sitio más conveniente para cada usuario.

Tan solo con tener la cámara conectada a la consola, el fondo de la interfaz de la *360* cambia automáticamente, mostrando la imagen que capta el ojo del periférico. La cámara graba a 30 *FPS* para una resolución de 640 por 480 píxeles y puede tomar imágenes de 1.3 megapíxeles, además puede ser utilizada para personalizar imágenes de un jugador o chatear mientras se juega (González, 2006). Esta fue descontinuada luego de la liberación de su sucesor el *Kinect* en 2010.

Figura 1.8. *XBox Live Vision*.

Kinect

Kinect es un dispositivo creado por *Microsoft*, con el objetivo de permitir la interacción de los usuarios con la videoconsola *XBox 360* en tiempo real. Dentro de sus características se puede destacar la capacidad de escaneo 3D, lo que permite reconstruir una escena a todo color. El *Kinect* es un *hardware* más barato que los sistemas complejos habituales con varias cámaras y está pensado como un sistema de captura de movimiento sin marcadores 3D, ya que puede mostrar un esqueleto simplificado sin hacer uso de vestimenta especial u otro dispositivo (Kandil, Hastak y Dunston, 2014).

Figura 1.9. *Kinect*.

Este es un periférico para la videoconsola *XBox 360*, lanzado en el 2010, siendo el resultado de las investigaciones realizadas por el departamento *Microsoft Research* y la comunidad de visión computarizada durante más de 10 años. El *hardware* de este dispositivo fue desarrollado por la compañía Israelí *PrimeSense*, la cual había creado con anterioridad cámaras de profundidad utilizando la misma técnica de rayos infrarro-

jos que utiliza el *Kinect*. Ambas compañías trabajaron en conjunto para crear una cámara de profundidad que utilizara el *software* y los algoritmos desarrollados por *Microsoft* en sus investigaciones. Luego de la liberación de este dispositivo, en noviembre del 2010, tuvo una rápida aceptación, vendiéndose 10 millones de unidades durante el primer mes para convertirse en el periférico más rápidamente vendido en la historia hasta ese momento (Borenstein, 2012).

Los componentes de *hardware* que contiene *Kinect* son dos cámaras que posee en su centro y un proyector de rayos infrarrojos. Una de las dos cámaras es un sensor específicamente diseñado para captar la luz infrarroja. Así, el *Kinect* puede ver la red de puntos infrarrojos que se proyectan en los objetos frente a él. Al lado de la cámara infrarroja se ubica la cámara de color estándar, la cual tiene un sensor digital que es similar a la de muchas cámaras *web* y cámaras digitales pequeñas, teniendo una resolución relativamente baja (640 por 480 píxeles). Por sí misma, esta cámara no es particularmente interesante, sino una cámara *web* de baja calidad, pero como se encuentra a una distancia conocida de la cámara infrarroja, es posible alterar el color de la imagen en función de su profundidad. Además de las cámaras, tiene cuatro micrófonos los que se distribuyen alrededor de este. Su propósito no es solo capturar el sonido, sino también localizarlo dentro de una habitación. Dentro de la base plástica del *Kinect* existe un pequeño motor y una serie de engranajes. Girando este motor, el *Kinect* puede inclinar sus cámaras y altavoces de arriba hacia abajo. La gama del motor de movimiento está limitado a unos 30 grados. *Microsoft* añade el motor para permitir que el *Kinect* pueda trabajar en una mayor variedad de habitaciones (ibíd.).

Mediante la cámara infrarroja se registra la distancia de los objetos, creándose una imagen de profundidad que resulta mucho más fácil de entender para un ordenador. En una imagen de profundidad, el color de cada píxel es el que indica la distancia a la que una determinada parte de la imagen está de la cámara. Debido a que el *Kinect* crea su imagen de profundidad mediante una luz infrarroja, este capturará la misma imagen de profundidad en una habitación con buena iluminación como en una a oscuras, lo que hace que las imágenes de profundidad sean más confiables. A diferencia de una cámara convencional, que capta cómo se ven las cosas, una cámara de profundidad captura dónde están las cosas (ibíd.).

La imagen de profundidad del *Kinect* se construye utilizando una luz estructurada, mediante la proyección con la luz infrarroja de un patrón de píxeles conocidos y luego observando la deformación que causan los objetos a ese patrón. Esta técnica se combina con otras dos de visión computarizada: profundidad focal, *Depth From Focus* (*DFF*) y profundidad estéreo, *Depth From Stereo* (*DFS*). La primera de estas, se basa en el principio de que los objetos más lejanos se ven más borrosos. Mientras que la segunda, utiliza el principio del paralaje, donde al observar una escena desde un ángulo diferente, los objetos cercanos cambian más que los objetos que se encuentran a una mayor distancia. Luego, de estas imágenes de profundidad se infiere la posición de cuerpos humanos y las principales partes de estos, mediante la utilización de un árbol de decisión aleatorio, técnica predictiva utilizada en el ámbito de la Inteligencia Artificial (IA) (MacCormick, 2011; Schechner y Kiryati, 2000; Shotton et al., 2013).

1.1.2. Sistemas inerciales de captura de movimiento

Los sistemas inerciales son fáciles de transportar y tienen grandes rangos de captura, utilizando pequeños sensores que pueden ser acelerómetros o giroscopios. Mediante los acelerómetros se puede medir la aceleración de los movimientos realizados y conocer su posición, mientras que con los giroscopios se obtiene la orientación de los objetos, además de la velocidad angular con la que se desplazaron. Los sensores pueden ser acoplados a trajes especiales como es el caso del *IGS-190M (Figura 1.10)*, en el que también se ubica una unidad inalámbrica para enviar los datos recogidos a una computadora para su posterior análisis. Una de las razones que impide el acceso rápido a estos trajes, es su elevado costo, ya que se encuentra por encima de los 30.000 dólares (MetaMotion, 2011b). Sin embargo, también existen otros sistemas inerciales de menores prestaciones como es el caso del mando inalámbrico de la consola de videojuegos *Wii* de *Nintendo (Figura 1.11)* (Taylor y Cubero, 2013; Vlasic et al., 2007).

Figura 1.10. Traje *IGS-190M* con sensores inerciales.

Figura 1.11. *Wiimote* de *Nintendo*.

1.1.3. Sistemas electromecánicos de captura de movimiento

Estos sistemas utilizan trajes compuestos de barras metálicas o plásticas unidas mediante potenciómetros colocados en las principales articulaciones *(Figura 1.12)*, los cuales al realizar un movimiento, detectan el grado de apertura de estas. Mediante estos sistemas es posible determinar los movimientos y la posición relativa de cada parte del cuerpo, pero su capacidad para detectar desplazamiento es limitado. Además, estos sistemas asumen que la mayoría de los huesos humanos están unidos por articulaciones, donde el único valor a medir es su grado de apertura, por lo que no tienen en cuenta rotaciones complejas que se producen frecuentemente en las articulaciones humanas, como los movimientos de hombros o los giros de los antebrazos. Estos trajes no permiten variar la posición de los sensores, son pesados y suelen restringir el movimiento del actor (Beltrán Álvarez et al., 2004; Vlasic et al., 2007). Además su costo, el cual sobrepasa los 7.000 dólares, como es el caso del traje *Gypsy 7* (MetaMotion, 2011a), dificulta su adquisición.

Figura 1.12. Traje electromecánico *Gypsy 6*.

1.1.4. Sistemas electromagnéticos de captura de movimiento

Estos sistemas utilizan sensores que son colocados en el cuerpo para medir su relación espacial con un transmisor cercano *(Figura 1.13)*. Los campos magnéticos generados por el transmisor son detectados por los sensores, los cuales transmiten esta información a una unidad electrónica de control, en la cual es filtrada y amplificada. Posteriormente, esta información es enviada a un ordenador para su procesamiento. Por cada sensor se puede calcular continuamente su posición y rotación a gran velocidad. Estos sistemas ofrecen una buena precisión y altas velocidades de actualización, sin presentar problemas de oclusión (Beltrán Álvarez et al., 2004; Vlasic et al., 2007). Sin embargo, son altamente costosos y presentan un alto consumo eléctrico, como es el caso del *MotionStar Wireless 2*, con un costo de aproximadamente 50.000 dólares (SouVR, 2008).

Figura 1.13. Componentes de captura de movimiento del *MotionStar Wireless 2*.

1.2. La captura de movimiento en la rehabilitación

El interés de algunos investigadores en utilizar videojuegos en fisioterapia, terapia ocupacional y psicoterapia, aumentó considerablemente en los últimos años. Muchas terapias físicas se basan en la repetición para lograr un rango de movimiento o control sobre un grupo muscular específico. Ese proceso se realiza sin ningún estímulo externo, por lo que hace que los pacientes pierdan la motivación durante la terapia, además de que la rehabilitación se vuelve más lenta y frustrante. Es por ello que mediante el desarrollo de un juego como herramienta de rehabilitación, se puede alcanzar la motivación deseada en los pacientes (Fernández Baena, Susin y Lligadas, 2012).

Dentro del contexto de los juegos serios, existe un tipo específico que busca fomentar la actividad física mediante diferentes roles de juego, los *Exergames*, que estimulan la movilidad del cuerpo mediante el uso de ambientes interactivos con experiencias de inmersión que simulan diferentes sensaciones de presencia. Sin embargo, existen algunos inconvenientes a la hora de implementar un sistema de captura de movimiento para estos videojuegos. Los más comunes son su elevado costo y la complejidad de su montaje. A pesar de la existencia de múltiples sensores de movimiento de bajo costo creados por las industrias de los videojuegos para mejorar las experiencias con sus consolas, cuando son utilizados en el contexto de la rehabilitación física, suelen tener inconvenientes para la captura objetiva de datos de movimiento (posición, velocidad, aceleración y ángulos) a través de herramientas computacionales (Maresh, 2014; Muñoz Cardona, Henao Gallo y López Herrera, 2013).

Dentro de los sensores creados para la industria del entretenimiento, se destacan por su capacidad de ser usados en terapias de rehabilitación, el *Wiimote*, el *PlayStation Eye* y el *Kinect*. El mando de la consola *Wii*, ha sido ampliamente utilizado para el tratamiento de personas con problemas neurológicos, de concentración y para la recuperación de la movilidad de las articulaciones. Pero su aplicación en un proceso de rehabilitación completo, en el que se incluyan varios miembros, está limitada. Por otra parte la combinación del *PlayStation Eye* con el *PlayStation Move*, un mando muy similar al *Wiimote*, ofrece esencialmente re-

sultados equivalentes a los que se pueden obtener mediante el mando desarrollado por *Nintendo*. Mientras tanto, dado que el *Kinect* puede conformar un esqueleto de la persona detectada, es el sistema más versátil y con mayor margen de experimentación. Mediante la *Software Development Kit* (*SDK*) *"Kinect for Windows"* desarrollada por *Microsoft*, que proporciona a los desarrolladores el acceso a funciones avanzadas del *Kinect*, han sido creadas numerosas aplicaciones para ayudar a los pacientes con esclerosis múltiple, para la corrección de movimientos de los miembros superiores e inferiores, entre otros. De las aplicaciones desarrolladas con el *Kinect*, también se han beneficiado los doctores, dado que esta es una herramienta que les permite analizar minuciosamente los movimientos realizados por los pacientes, logrando así realizar un mejor diagnóstico (Butler y Willett, 2010; DiCYT, 2009; Lange et al., 2010; Muijzer, 2014; Muñoz Cardona, Henao Gallo y López Herrera, 2013).

1.3. Disponibilidad de sensores en el centro VERTEX

Para la realización de un proceso de captura de movimiento, el Centro de Entornos Interactivos 3D (VERTEX) de la UCI cuenta con algunos de los dispositivos mencionados en los epígrafes anteriores. Entre estos se encuentra el mando de la consola *Wii*, el *Wiimote*, así como *webcams* que pueden ser utilizadas para este fin haciendo uso de marcadores 3D para facilitar el proceso. Además, este centro pudo adquirir recientemente un sensor *Kinect* mediante una donación realizada por un grupo de doctores. La disponibilidad de este último dispositivo, provee al centro VERTEX de una poderosa herramienta de captura de movimiento, que si bien es menos precisa que otros sistemas de captura como los electromagnéticos, las mediciones que puede realizar se encuentran dentro de un rango confiable en aspectos como posiciones y ángulos de movimiento.

1.4. Herramientas y Metodologías

1.4.1. Metodologías de desarrollo

El desarrollo de un *software* de alta calidad, en el tiempo planificado, con los menores costos posibles y que además satisfaga las necesidades y expectativas del cliente, requiere que durante todo el ciclo de vida de este, se trabaje de forma organizada. Para esto se debe controlar y documentar toda la información relacionada con el proyecto, prever los posibles riesgos que se puedan presentar durante su ejecución y definir las medidas para mitigarlos. La correcta realización de estas tareas depende en gran medida del empleo de una metodología eficaz que se adapte a las características del producto deseado. Es por ello, que un cambio en algún momento del desarrollo de un *software*, que no sea la fase inicial cuando se realiza la captura de requisitos, puede devenir en atrasos en los cronogramas trazados o pérdidas monetarias. En base a este problema existen metodologías que son en cierta medida más o menos sensibles a estos cambios. Las mismas son conocidas como **tradicionales**, para el caso en que estas son mucho menos adaptables a un cambio en los requerimientos y **ágiles**, en el caso que estas sean más abiertas al mitigar atrasos o pérdidas monetarias

por un cambio en la concepción del producto por parte del cliente (Pressman, 2010).

Las **metodologías tradicionales**, están guiadas por una rigurosa planificación durante todo el proceso de desarrollo, en el cual se establecen estrictamente las actividades a realizar. Están orientadas a proyectos de gran envergadura, para los cuales se definen una gran cantidad de roles y artefactos a generar, contando con una detallada documentación (Canós, Letelier y Penadés, 2003). Entre estas metodologías se destaca *Rational Unified Process* (*RUP*).

- *RUP*.
  Basada en componentes e interfaces bien definidas. Esta metodología define un marco de trabajo genérico, el cual puede especializarse para una gran variedad de sistemas de *software*, en diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto. Esta comprende tres principios claves: dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental. Esto significa que los casos de uso describen los requisitos funcionales del sistema, desde la perspectiva del usuario. Estos no solo inician el proceso de desarrollo sino que también proporcionan un hilo conductor, en el que se verifica, luego de la implementación, que el producto implemente adecuadamente cada caso de uso. Por otra parte, la arquitectura de un sistema permite tener una visión común entre los desarrolladores y los usuarios, teniendo una vista del diseño completo del *software* con las características más importantes resaltadas. Además *RUP* propone un proceso iterativo e incremental, donde el trabajo se divide en partes más pequeñas. Cada una de estas partes se puede ver como una iteración de la cual se obtiene un incremento que produce un crecimiento en el producto (Jacobson, Booch y Rumbaugh, 1999).

Las **metodologías ágiles**, son aquellas que están orientadas a la producción de código, con ciclos muy cortos de desarrollo. Se dirigen por grupos pequeños, haciendo hincapié en aspectos humanos asociados al trabajo en equipo. Están orientadas a proyectos pequeños, los cuales pueden presentar cambios durante su realización e involucran activamente al cliente en el proceso de desarrollo (Canós, Letelier y Penadés, 2003). Entre estas metodologías se encuentran:

- Programación Extrema o *Extreme Programming* (*XP*).
  Basada en la simplicidad, la comunicación y la reutilización de código. Esta metodología trata de darle al cliente el *software* que él necesita y cuando lo necesita, lo que implica responder rápidamente a las necesidades de este. Además tiene como objetivo potenciar al máximo el trabajo en grupo, donde los clientes y los desarrolladores son parte del equipo de trabajo y están involucrados en el proceso de desarrollo del *software* durante todo su ciclo de vida. Esta metodología de desarrollo de *software* se recomienda para proyectos de corto plazo, poniendo más énfasis en la adaptabilidad que en la previsibilidad. Es definida especialmente como una metodología adecuada para proyectos con requisitos imprecisos y muy cambiantes. Esta metodología intenta reducir la complejidad del *software* por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción (Calero, 2003; Escribano, 2002).

- SCRUM.
  Esta metodología requiere trabajo duro puesto que no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto. Esta metodología además, está especialmente indicada para proyectos con un rápido cambio de requisitos. El desarrollo de *software* se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. Donde el resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. Otro aspecto importante de esta metodología son las reuniones diarias de 15 minutos que esta propone. En estas se debate lo que se ha hecho desde la última reunión, los posibles obstáculos que atentan contra la productividad del equipo, así como las tareas que se realizarán antes de la siguiente reunión (Pressman, 2010).

1.4.2. Lenguaje de programación

El lenguaje que se utilizará es C#, el cual es un lenguaje de propósito general diseñado por *Microsoft* para su plataforma *.NET*. La sintaxis y estructuración de este lenguaje es muy similar a la de C++, ya que la intención de *Microsoft* era facilitar la migración de códigos escritos en este lenguaje y su aprendizaje por parte de los desarrolladores habituados a C++. El lenguaje de programación C#, además de combinar las mejores características de lenguajes preexistentes como *Visual Basic*, *Java* y C++ (Seco, 2001), es el lenguaje en que está implementado el videojuego Danzo-Terapia.

1.4.3. Bibliotecas

En informática, una biblioteca (*library* en inglés) es un conjunto de funcionalidades, codificadas en un lenguaje de programación, la cual ofrece una interfaz bien definida para invocarlas. A diferencia de un programa ejecutable, una biblioteca no ejecuta automáticamente sus funcionalidades. Los servicios que esta proporciona pasan a formar parte del programa, o de otra biblioteca que los utilice. De esta manera se utiliza lo que ya está implementado en la biblioteca ahorrándose tiempo y recursos, además esto permite que el código y los datos se compartan y puedan modificarse de forma modular.

Bibliotecas de integración con el *Kinect*

Actualmente existen dos *Kit* de Desarrollo, *SDK*, que habilitan la interacción con el *Kinect*: *"Kinect for Windows"* de *Microsoft* y *"OpenNI + NITE"* de *PrimeSense*. Ambas fueron creadas para seguir el movimiento del esqueleto mediante la cámara de profundidad del *Kinect*. *Kinect for Windows* además de brindar total soporte en los lenguajes C++ y C#, cuenta con una amplia y actualizada documentación. Es capaz de seguir los movimientos realizados por hasta seis personas mediante la detección de esqueletos, sin necesitar que estos adopten una postura inicial. En su versión 1.8, puede realizar el seguimiento de 20 *joins* distribuidos por el cuerpo, los cuales forman la estructura básica de un esqueleto. Esta además logra determinar distancias entre objetos mediante la cámara de profundidad, capturar audio utilizando la cancelación de ruido y eco, así como localizar el origen del audio y reconocer comandos de voz. Por su parte *OpenNI + NITE* solo

soporta totalmente el lenguaje C++ y no cuenta con una documentación completa. *PrimeSense* suministró la *SDK* en dos parte, *OpenNI*, la cual incluye los *drivers* para acceder a la información de la imagen de profundidad y *NITE*, que realiza la detección de los *joins* de las personas que se encuentran dentro del rango de visión de la cámara. *OpenNI* está disponible bajo la licencia libre *LGPL*, mientras que *NITE* es un producto comercial de *PrimeSense*. Esta *SDK* funciona con todos los sensores de la compañía y tiene soporte para *Windows*, *Macintosh* y *Linux*. Además puede realizar el seguimiento de 24 *joins* de hasta seis personas, pero requiere una posición inicial de estas. El proyecto *OpenNI* fue abandonado por *PrimeSense*, luego de que *Apple* comprara esta compañía y sacara de circulación los sitios que daban soporte al proyecto (Borenstein, 2012; Isaac y Paczkowski, 2013; Microsoft Corporation, 2013).

1.4.4. Entorno de desarrollo

Para el desarrollo del módulo se utilizará el motor de videojuegos *Unity 3D* en su versión 4.3.0 del año 2013. Este es un motor de videojuegos multiplataforma creado por *Unity Technologies*, el cual brinda una plataforma de desarrollo flexible y poderosa para crear ambientes interactivos 3D y bidimensional (2D). *Unity 3D* posee un editor visual mediante el cual se pueden importar modelos 3D, texturas, sonidos o componentes prefabricados. Además incluye a *MonoDevelop* como editor de código por defecto, el cual es un *Integrated Development Environment* (*IDE*) multiplataforma, libre y gratuito, que está diseñado primordialmente para C#. Las aplicaciones desarrolladas en *Unity 3D* pueden ser compatibles con disímiles plataformas, entre las que se incluyen *Windows*, *Mac*, *Linux*, *XBox 360*, *PS3*, *Wii*, entre otros. (Technologies, 2013) Además, este es el *IDE* en que se desarrolló el videojuego Danzo-Terapia.

Conclusiones parciales

En este capítulo se analizaron diferentes dispositivos que pueden ser utilizados para capturar movimiento, así como el costo de su adquisición. Además se analizó la disponibilidad de sensores en el centro VERTEX, escogiéndose el sensor *Kinect* para la realización del presente trabajo, dado el nivel de confianza de las mediciones que realiza. También se eligió para la creación de la solución al problema planteando el motor de videojuegos *Unity 3D* y como *IDE MonoDevelop*, como lenguaje de programación C#, utilizando como biblioteca de integración para el *Kinect* la *SDK "Kinect for Windows"* y haciendo uso de *XP* como metodología para el desarrollo ágil de soluciones informáticas.

CAPÍTULO 2

Propuesta de solución

Introducción

El presente capítulo tiene como objetivo hacer una descripción de las principales características del módulo a desarrollar. Se hará mención de las primeras fases de la metodología de desarrollo a utilizar para la implementación de la solución que se propone y se expondrán los artefactos generados durante el transcurso de estas.

2.1. Descripción de la solución propuesta

Para el cumplimiento del objetivo de la investigación se propone el módulo *KMSkeletonTracking*, este permitirá mediante el uso de la cámara infrarroja integrada al dispositivo *Kinect* y haciendo uso de las funcionalidades que provee la *SDK "Kinect for Windows"*, el seguimiento y detección de los *joins* que conforman el cuerpo de la persona que se encuentre haciendo uso del dispositivo. Para su utilización se definen dos flujos de trabajo principales. En el primero, un especialista realiza los ejercicios propuestos por el videojuego Danzo-Terapia y el módulo generará un archivo con los datos de las acciones realizadas por el terapeuta. Estos datos serán utilizados por la IA del videojuego, como parte de su base de conocimiento, para evaluar los ejercicios realizados por los pacientes. Luego, en el segundo momento, mientras los pacientes realizan las rutinas de ejercicios, el módulo capturará los datos de los movimientos que estos efectúan, los que serán enviados a Danzo-Terapia en tiempo real, donde serán comparados con los datos de entrenamiento generados anteriormente por el módulo. En ambos procesos, *KMSkeletonTracking* podrá configurarse para modificar la tasa con que son grabados los datos. También se definirá el formato y la estructura del archivo generado, para que pueda ser utilizado por los desarrolladores del videojuego. Como parte de las funcionalidades que este ofrecerá, se encuentra la integración con el motor de videojuegos *Unity 3D* para integrar las funciones propias del *Kinect* con aplicaciones desarrolladas en este motor de videojuegos. El módulo además mantendrá en todo momento un indicador sobre el estado de la conexión con el dispositivo de entrada (*Kinect*), permitiéndole a la aplicación conocer si se realizó la conexión con éxito o si este presentó algún problema y

no se encuentra disponible.

Con la realización de la solución descrita anteriormente, el resultado obtenido permitirá a los desarrolladores del videojuego Danzo-Terapia integrar el *Kinect* a su aplicación. Así este será capaz de retroalimentarse de las acciones que realizan los pacientes al efectuar los ejercicios de rehabilitación. Esta información podrá ser procesada por el videojuego para detectar si el ejercicio se está realizando de la forma correcta.

2.2. Especificación de requisitos

2.2.1. Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir y describen lo que el sistema debe hacer. Estos dependen del tipo de *software* que se desarrolle, de sus posibles usuarios y del enfoque general tomado al redactar estos requisitos. Además describen con detalle las funciones y características de un sistema, así como las restricciones que gobiernan su desarrollo (Pressman, 2010). A continuación se presentan los requisitos funcionales del módulo.

RF 1. Integrar el módulo con *Unity 3D*.

RF 2. Detectar conexión del *Kinect*.

RF 3. Detectar posición de los *joins*.

RF 4. Determinar orientación de los *joins*.

RF 5. Generar datos de entrenamiento.

2.2.2. Requisitos no funcionales

Los requisitos no funcionales, como su nombre sugiere, son aquellos requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada, de salida y las representaciones de datos que se utilizan en las interfaces del sistema (Sommerville, 2007). A continuación se presentan los requisitos no funcionales del módulo.

RnF 1. *Hardware:*

- Procesador: *Intel Dual Core* a 2.6 *GHz*.
- Memoria: 2 *GB* de *RAM*.
- Dispositivos de entrada adicionales: *Kinect*.

RnF 2. *Software:*

- Sistema Operativo: *Windows 7* o superior.
- Biblioteca y *drivers* del *Kinect*: *SDK "Kinect for Windows"*.

2.3. Fase de exploración

Durante esta etapa los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto y los programadores se encargan de realizar las estimaciones correspondientes. El equipo de desarrollo se familiariza con las herramientas, las tecnologías y prácticas que se utilizarán en el proyecto. Además se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo (Anaya Villegas, 2007).

2.3.1. Descripción de historias de usuario

Las historias de usuario son utilizadas como herramienta para dar a conocer los requerimientos del sistema al equipo de desarrollo. Son pequeños textos en los que el cliente describe una actividad que realizará el sistema. Estos se redactan utilizando la terminología del cliente, de forma que sea clara y sencilla, sin profundizar en detalles. Se puede considerar que las historias de usuario en *XP* juegan un papel similar a los casos de uso en otras metodologías, pero en realidad son muy diferentes. Las historias de usuario solo muestran la silueta de una tarea a realizarse. Por esta razón es fundamental que el usuario o un representante suyo se encuentre disponible en todo momento para dar respuesta a las dudas que puedan surgir, ya que estas no proporcionan información detallada acerca de una actividad específica. Las historias de usuario también son utilizadas para estimar el tiempo que el equipo de desarrollo tomará para realizar las entregas. En una entrega se puede desarrollar una o varias historias de usuario, esto depende del tiempo que demore la implementación de cada una de ellas (Echeverry Tobón y Delgado Carmona, 2007).

Tabla 2.1. Historia de usuario # 1

Historia de usuario	
Número: 1	**Nombre:** Integrar el módulo con *Unity 3D*.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1.5	**Iteración asignada:** 1
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: El módulo debe ir integrándose con el motor de videojuegos *Unity 3D* a medida que se implementa.	
Observaciones:	

Tabla 2.2. Historia de usuario # 2

Historia de usuario	
Número: 2	**Nombre:** Detectar conexión del *Kinect*.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1.0	**Iteración asignada:** 1
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: Una vez que la aplicación se ejecute, debe conocer si existe algún dispositivo conectado.	
Observaciones: En caso de existir la conexión con el *Kinect*, la aplicación debe enviar una notificación al videojuego de que se ha detectado un dispositivo, para poder activar su función de evaluación. En caso contrario, la función de evaluación no podrá ser activada.	

Tabla 2.3. Historia de usuario # 3

Historia de usuario	
Número: 3	**Nombre:** Detectar posición de los *joins*.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3.0	**Iteración asignada:** 2
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: La aplicación debe ser capaz de detectar la posición de las principales partes del cuerpo, mediante los *joins* proporcionados por el *Kinect*.	
Observaciones: El *Kinect* puede inferir incorrectamente la posición de un punto producto a un error ajeno a él, como es el caso de la oclusión del cuerpo o de una parte de este.	

Tabla 2.4. Historia de usuario # 4

Historia de usuario	
Número: 4	**Nombre:** Determinar orientación de los *joins*.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3.0	**Iteración asignada:** 3
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: La aplicación debe ser capaz de detectar la orientación de las principales partes del cuerpo, mediante los *joins* proporcionados por el *Kinect*.	
Observaciones: El *Kinect* puede inferir incorrectamente la orientación de un punto producto a un error ajeno a él, como es el caso de la oclusión del cuerpo o de una parte de este.	

Tabla 2.5. Historia de usuario # 5

Historia de usuario	
Número: 5	**Nombre:** Generar datos de entrenamiento.
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3.0	**Iteración asignada:** 4
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: Cuando la aplicación lo requiera, el módulo debe generar un fichero con los datos capturados. Esta información formará parte de los datos de entrenamiento que utilizará la aplicación para evaluar a los pacientes.	
Observaciones:	

2.4. Fase de planificación

En esta fase el cliente establece la prioridad de cada historia de usuario y se acuerda el alcance del producto. Los programadores estiman cuánto esfuerzo requiere cada historia y a partir de esto se define el cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. El equipo de desarrollo mantiene un registro de la velocidad de desarrollo, totalizando el número de historias de usuario realizadas en una iteración. Esta medida ayuda a determinar la cantidad de historias que se pueden implementar en las siguientes iteraciones, aunque no de manera exacta. La revisión continua de esta métrica en el transcurso del proyecto se hace necesaria, ya que las historias varían según su grado de dificultad, haciendo inestable la velocidad de la realización del sistema (Anaya Villegas, 2007; Echeverry Tobón y Delgado Carmona, 2007).

2.4.1. Estimación de esfuerzo por historia de usuario

Las estimaciones de esfuerzo asociado a la implementación de las historias de usuario la establecen los programadores utilizando como medida el punto, el cual equivale a una semana ideal de programación. Esto es utilizado para estimar el tiempo que el equipo de desarrollo necesitará para realizar las entregas. En una entrega puede desarrollarse una o varias historias de usuario, dependiendo del tiempo que demore la implementación de cada una de ellas (Escribano, 2002).

Tabla 2.6. Estimación de esfuerzo por historia de usuario

Iteración	**Historias de usuario**		**Puntos estimados (semanas)**
1	1	Integrar el módulo con Unity 3D	1.5
	2	Detectar conexión del Kinect	1.0
2	3	Detectar posición de los joins	3.0
3	4	Determinar orientación de los joins	3.0

Continúa en la próxima página

Tabla 2.6. Continuación de la página anterior

4	5	Generar datos de entrenamiento	3.0
Total			**11.5**

2.4.2. Plan de iteraciones

El plan de iteraciones se encarga de organizar las historias de usuario por iteraciones y especifica las que se desarrollarán en cada iteración del proceso de implementación. La duración de una iteración varía entre una y tres semanas, al final de la cual habrá una entrega de los avances del producto. Estas entregas deberán ser completamente funcionales, superando las pruebas de aceptación que establece el cliente para verificar el cumplimiento de los requisitos. Las tareas que no se realicen en una iteración son tomadas en cuenta para la próxima, donde se define, junto al cliente, si se deben realizar o si deben ser removidas de la planeación del sistema (Echeverry Tobón y Delgado Carmona, 2007).

Luego de que fueron descritas las historias de usuario y se estimó el esfuerzo para cada una de ellas, se decide implementar el módulo en cuatro iteraciones. A continuación se describe cada una de ellas.

Iteración 1: Esta iteración tiene como objetivo darle cumplimiento a las historias de usuario 1 y 2, que realizan dos funcionalidades esenciales para el módulo. En esta se comprueba si el dispositivo está conectado y disponible para su utilización. Culminada esta iteración se efectúa la primera entrega del módulo, para de esta forma mostrarle al cliente lo realizado y recibir sus valoraciones con relación a esta entrega.

Iteración 2: Esta iteración tiene como objetivo darle cumplimiento a la historia de usuario 3, considerada una de las de mayor importancia para el módulo. En esta se obtienen parte de los datos que serán utilizados posteriormente para la evaluación de la ejecución de los ejercicios. Además se rectifican los errores de la iteración anterior.

Iteración 3: Esta iteración tiene como objetivo darle cumplimiento a la historia de usuario 4, otra de las de mayor importancia para el módulo. En esta se obtienen los restantes datos que serán utilizados posteriormente para la evaluación de la ejecución de los ejercicios. Además se rectifican los errores de la iteración anterior. Con la culminación de esta iteración el módulo contará con sus principales funcionalidades implementadas y se realizará la segunda entrega a los clientes.

Iteración 4: Esta iteración tiene como objetivo darle cumplimiento a la historia de usuario 5 que es de gran importancia para el módulo ya que posibilita la generación de datos de entrenamiento. Estos datos serán utilizados por la IA del videojuego para evaluar los ejercicios realizados por el paciente. Además se rectifican los errores de la iteración anterior. Culminada esta iteración se tiene el módulo listo para entregarle la primera versión completa al cliente.

Tabla 2.7. Plan de duración de las iteraciones

Iteración	Historias de usuario		Duración (semanas)
1	1	Integrar el módulo con Unity 3D	2.5
	2	Detectar conexión del Kinect	
2	3	Detectar posición de los joins	3.0
3	4	Determinar orientación de los joins	3.0
4	5	Generar datos de entrenamiento	3.0
Total			**11.5**

2.4.3. Plan de entregas

A partir del plan de iteraciones anterior se realiza el plan de entregas, que tiene como objetivo definir las entregas que se deben realizar y el orden de estas.

Tabla 2.8. Plan de entregas

Iteración	Entrega	Fecha
1	Versión 0.2	27 de marzo del 2015
2, 3	Versión 0.8	4 de mayo del 2015
4	Versión 1.0	30 de mayo del 2015

2.4.4. Tareas de ingeniería

Las historias de usuario son descompuestas en tareas de ingeniería y asignadas a los programadores para ser implementadas en cada iteración (Canós, Letelier y Penadés, 2003).

Tabla 2.9. Tarea de ingeniería # 1

Tarea	
Número de tarea: 1	**Número de historia de usuario:** 1
Nombre de la tarea: Enlazar las funciones exportadas por la *SDK "Kinect for Windows"*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.5
Fecha de inicio: 4 de marzo de 2015	**Fecha de fin:** 13 de marzo de 2015
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: Enlazar las funciones exportadas por la biblioteca implementada en C++ de la *SDK "Kinect for Windows"*, para lograr que el módulo sea completamente compatible con *Unity 3D*.	

Tabla 2.10. Tarea de ingeniería # 2

Tarea	
Número de tarea: 2	**Número de historia de usuario:** 2
Nombre de la tarea: Detectar conexión del *Kinect*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 16 de marzo de 2015	**Fecha de fin:** 28 de marzo de 2015
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: El módulo debe notificarle al videojuego si existe algún *Kinect* conectado para que este pueda activar su función de evaluación.	

Tabla 2.11. Tarea de ingeniería # 3

Tarea	
Número de tarea: 3	**Número de historia de usuario:** 3
Nombre de la tarea: Detectar posición de los *joins*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3.0
Fecha de inicio: 30 de marzo de 2015	**Fecha de fin:** 17 de abril de 2015
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: La aplicación debe detectar la posición de las principales partes del cuerpo, las cuales pueden ser utilizadas para la generación de los datos de entrenamiento que utilizará la IA del videojuego o para evaluar la ejecución de los ejercicios.	

Tabla 2.12. Tarea de ingeniería # 4

Tarea	
Número de tarea: 4	**Número de historia de usuario:** 4
Nombre de la tarea: Determinar orientación de los *joins*	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3.0
Fecha de inicio: 20 de abril de 2015	**Fecha de fin:** 8 de mayo de 2015
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: La aplicación debe detectar la orientación de las principales partes del cuerpo, representándolas como matrices de rotación y *quaternions*. Estos datos serán utilizados para la generación de los datos de entrenamiento y para evaluar la ejecución de los ejercicios.	

Tabla 2.13. Tarea de ingeniería # 5

Tarea	
Número de tarea: 5	**Número de historia de usuario:** 5
Nombre de la tarea: Generar datos de entrenamiento para un ejercicio	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3.0

Continúa en la próxima página

Tabla 2.13. Continuación de la página anterior

Fecha de inicio: 11 de mayo de 2015	**Fecha de fin:** 29 de mayo de 2015
Programador responsable: Ernesto Antonio Leyva Piñeda	
Descripción: Cuando la aplicación lo requiera, el módulo debe generar un fichero con los datos capturados durante la ejecución de un ejercicio. Esta información formará parte de los datos de entrenamiento que utilizará la aplicación para evaluar a los pacientes.	

2.5. Fase de diseño

En *XP* se considera que no es posible tener un diseño completo y sin errores del sistema desde el principio, dada la naturaleza cambiante del proyecto. Por lo tanto, hacer un diseño muy extenso en las fases iniciales del proyecto para luego modificarlo, se considera un malgasto de tiempo. Es por ello, que esta tarea es permanente durante la vida del proyecto, partiendo de un diseño inicial el cual va siendo corregido y mejorado durante el proceso de desarrollo (Echeverry Tobón y Delgado Carmona, 2007).

2.5.1. Arquitectura

La arquitectura de *software* es la estructura de un sistema, la cual comprende los componentes por los que está formado, las propiedades de estos componentes visibles externamente, y las relaciones entre ellos. En el contexto del diseño arquitectónico, un componente de *software* puede ser tan simple como un módulo, pero también puede ser algo tan complicado como incluir bases de datos que permita la configuración de una red de clientes y servidores. Las propiedades de los componentes son aquellas características necesarias para entender cómo los componentes interactúan con otros componentes, dejando de lado las propiedades internas específicas de cada uno. Las relaciones entre los componentes pueden ser tan sencillas como una llamada de procedimiento de un módulo a otro, o tan complicadas como el protocolo de acceso a bases de datos. La arquitectura constituye un modelo relativamente pequeño y comprensible de cómo está estructurado el sistema y de cómo trabajan sus componentes en conjunto. Esta brinda la posibilidad de analizar la efectividad del diseño para darle cumplimiento a los requisitos fijados. De esta forma, el diseño arquitectónico y los patrones arquitectónicos pueden ser aplicados en el diseño de otros sistemas y representados a través de un conjunto de abstracciones que facilitan la descripción de la arquitectura de un modo predecible (Pressman, 2010).

Arquitectura basada en capas

En una arquitectura basada en capas el sistema se organiza jerárquicamente, de modo que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En esta arquitectura los componentes de la capa externa se encargan de la interacción con el usuario mediante una interfaz. En la capa interna, los componentes realizan operaciones exclusivas al ámbito del *software*, como la obtención de datos, la interacción con el sistema operativo o el control de

un dispositivo. Mientras, las capas intermedias proporcionan mecanismos necesarios para que la capa externa muestre los datos obtenidos por la capa inferior, controlando la lógica del sistema. Esta arquitectura permite la creación de un diseño basado en varios niveles de abstracción, en el cual un problema complejo puede dividirse en partes más pequeñas para darle solución. A cada nivel se le confía una misión simple, lo que permite el diseño de una arquitectura escalable, la cual puede ampliarse con facilidad en caso de que las necesidades aumenten. En una arquitectura basada en capas, cada nivel puede ser modificado de forma transparente. Conociendo la interfaz de la capa inferior y manteniendo la utilizada por la capa exterior, puede ser modificado el funcionamiento de la capa intermedia con el fin de optimizar un algoritmo o modificar el tratamiento de los datos (Pressman, 2010; Reynoso y Kicillof, 2004). A continuación se describe la arquitectura que se utilizará para el desarrollo del módulo *KMSkeletonTracking*.

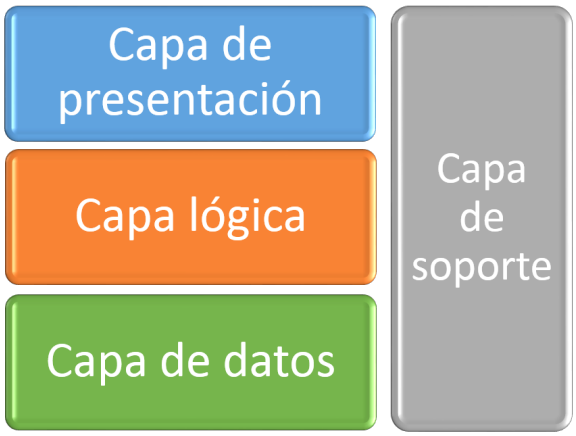

Figura 2.1. Arquitectura del módulo *KMSkeletonTracking*.

Capa de presentación

Esta es la capa con la que interactúa el usuario. A través de la interfaz del módulo, este puede obtener los datos capturados durante la realización de un ejercicio.

Capa lógica

En esta capa se controla el proceso de recolección de datos del *Kinect* para la generación de los datos de entrenamiento de la IA del videojuego, o para su envío directo a la aplicación.

Capa de datos

En esta capa se enlazan las funciones proporcionadas por la *SDK* del *Kinect* implementada en C++ para poder ser utilizadas en C#. Las estructuras de C++ que utiliza la *SDK*, son migradas a C# para poder trabajar con ellas. De esta forma pueden ser obtenidos los datos capturados por el *Kinect*.

Capa de soporte

En esta capa se encuentran las clases que provee el motor de videojuegos *Unity 3D*, las cuales son utilizadas para almacenar la información que provee el módulo.

2.5.2. Patrones de diseño y de asignación de responsabilidades

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software* orientado a objetos. Estos se caracterizan por su efectividad, la cual ha sido comprobada resolviendo problemas similares en ocasiones anteriores. Así como por su capacidad para ser reutilizables, ya que pueden ser aplicados a diferentes problemas de diseño en distintas circunstancias. Estos modelos pretenden estandarizar el modo en que se realiza un diseño, evitando la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente (Gamma et al., 1995). Tras la publicación del libro *Design Patterns* escrito por el grupo *Gang of Four* (*GOF*), compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, los patrones de diseño han tenido un gran éxito en el mundo de la informática. La utilización de estos, así como de los Patrones de Asignación de Responsabilidades, o *General Responsibility Assignment Software Patterns* (*GRASP*), constituye una de las buenas prácticas de diseño de clases a seguir. A continuación se exponen los patrones utilizados en el módulo desarrollado:

Experto

Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria sobre este. De este modo se obtiene un diseño con mayor cohesión, manteniendo la información encapsulada (Larman, 1999). Por su naturaleza este patrón fue utilizado en todas las clases.

Creador

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. Este patrón ayuda a identificar la clase que debe ser responsable de la creación de nuevos objetos (ibíd.). Un ejemplo de su utilización es la clase *KinectSensor* que tiene la responsabilidad de crear las estructuras que contendrán la posición y orientación de los *joins*.

Alta cohesión

Indica que la información que almacena una clase debe de ser coherente y debe estar relacionada con la clase (ibíd.). Por su naturaleza este patrón fue utilizado en todas las clases.

Bajo acoplamiento

Permite tener las clases lo menos enlazadas entre ellas. De esta forma, en caso de producirse una modificación en alguna clase, esta tendrá la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre ellas (ibíd.). Este patrón se evidencia en las relaciones que poseen las clases implementadas para la realización del módulo.

Singleton

Este patrón está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella. El patrón *singleton* se implementa creando un método que crea una instancia de la clase solo si aún no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se restringe el alcance del constructor para que solo sea accesible desde la propia clase. La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, solo uno de ellos debe lograr crear el objeto. La solución para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón. Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación (Freeman et al., 2005). Este patrón es utilizado en la clase *KinectSensor* para obtener una única instancia capaz de interactuar con el sensor *Kinect*.

2.5.3. Tarjetas CRC

La metodología *XP* propone el uso de tarjetas Clase-Responsabilidad-Colaboración (CRC) en lugar de una serie de complejos diagramas que probablemente tomen más tiempo y sean menos instructivos. La principal funcionalidad que tienen estas, es ayudar a dejar el pensamiento procedimental para incorporarse al enfoque orientado a objetos. Cada tarjeta representa una clase y cuenta con tres secciones fundamentales. En la parte superior de la tarjeta se encuentra el nombre de la clase que representa, en la parte inferior izquierda se encuentran las responsabilidades o funcionalidades y en la parte inferior derecha los colaboradores o clases que le sirven de soporte (Echeverry Tobón y Delgado Carmona, 2007). A continuación se presentan las tarjetas CRC que se identificaron en el módulo.

Tabla 2.14. Tarjeta CRC # 1

Tarjeta CRC	
Clase: KinectInterface	
Responsabilidad	**Colaboración**
• Permitir el acceso a las funcionalidades del módulo. • Posibilitar el enlace con el evento NewSkeletonFrame, el cual contendrá los datos capturados y será ejecutado según la frecuencia de captura establecida.	SkeletonTracking KMNewSkeletonFrameArgs KMNewColorFrameArgs

Tabla 2.15. Tarjeta CRC # 2

Tarjeta CRC	
Clase: KinectSensor	
Responsabilidad	**Colaboración**
• Detectar el estado del *Kinect*. • Comenzar el seguimiento de movimientos. • Detener el seguimiento de movimientos. • Comenzar el seguimiento de movimientos para entrenamiento de la IA del videojuego. • Detener el seguimiento de movimientos para entrenamiento de la IA del videojuego. • Obtener la posición de los *joins*. • Obtener la orientación de los *joins*. • Establecer la frecuencia de grabación.	KinectSDK KinectAPI

Tabla 2.16. Tarjeta CRC # 3

Tarjeta CRC	
Clase: KinectAPI	
Responsabilidad	**Colaboración**
• Enlazar las principales funciones de la *SDK "Kinect for Windows"* implementada en C++. • Proveer las estructuras y enumeradores necesarios para obtener los datos con los que trabajará el módulo.	Kinect10

2.6. Datos de entrenamiento

El módulo que se desea implementar debe generar un archivo con los datos capturados de la correcta realización de un ejercicio, los cuales serán utilizados por la IA del videojuego para evaluar los ejercicios realizados por los pacientes. Para cumplir con este requisito se diseña un reporte con formato *XML*, cuya estructura se muestra en el *(Anexo A.1)*, y recogerá los siguientes datos:

- Nombre del ejercicio.
- Tipo de ejercicio.
- Duración del ejercicio.
- Frecuencia con que fueron grabados los ejercicios.
- Posición de las partes del cuerpo durante la ejecución del ejercicio.
- Orientación de las partes del cuerpo durante la ejecución del ejercicio.

Conclusiones parciales

En este capítulo se analizó la propuesta del sistema, así como sus requisitos funcionales y no funcionales. Además se analizaron las primeras fases de la metodología de desarrollo de *software* utilizada. Durante el transcurso de estas se documentaron los artefactos generados por ellas, los cuales facilitaron la creación del diseño del módulo, mostrando los elementos más importantes a tener en cuenta durante su desarrollo.

CAPÍTULO 3

Resultados

Introducción

En el presente capítulo se presentará el estándar de codificación utilizado en la implementación de la solución propuesta. Además se expondrán los resultados obtenidos en las pruebas de aceptación realizadas al módulo con el objetivo de validar los resultados obtenidos en la implementación de cada historia de usuario. Por último se mostrarán los resultados que se obtuvieron durante la validación del módulo.

3.1. Fase de desarrollo

En metodologías pesadas, la codificación es un proceso al cual solo se llega después de largas fases de análisis y diseño. En estas se genera una gran cantidad de documentación a partir de la cual el proceso de codificación es relativamente sencillo. En *XP*, la codificación es un proceso que se realiza en forma paralela con el diseño. Esto favorece el logro del objetivo de hacer entregas frecuentes al cliente. Algunos de los elementos más importantes de esta fase son que, el cliente siempre debe estar presente durante esta, se debe trabajar en parejas y debe haber una propiedad colectiva del código (Echeverry Tobón y Delgado Carmona, 2007).

3.1.1. Estándar de codificación

Los estándares de codificación constituyen pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física. Estos estándares crean un buen hábito en el equipo de desarrollo al utilizar un estilo de programación común para todos. También permiten que todos los involucrados en el proyecto entiendan el código fácilmente y en menor tiempo (ibíd.). A continuación se presenta el estándar de codificación utilizado en el módulo.

Indentación

La indentación será de cuatro espacios, utilizándose la tabulación. No poseerán indentación las llaves asociadas a los espacios de nombre, definición de clases, implementación de métodos, condicionales y bucles.

Líneas y espacios en blanco

Se insertará una línea en blanco entre las definiciones de clases y métodos. Los operadores binarios estarán separados de sus operandos por un espacio en blanco, excepto en el caso de los operadores unitarios de incremento (++) y decremento (--). De igual forma serán separados los argumentos de un método, insertando un espacio después de la coma.

Declaración de variables

Se declarará una variable por línea. Además se evitará nombrar a las variables con abreviatura siempre que sea posible. Todos los caracteres serán escritos en minúscula, excepto en caso de que existan nombres compuestos. En este caso, se escribirá con minúscula el primer nombre y los restantes con letra inicial mayúscula, evitando usar el guion bajo.

Declaración de métodos

Siguen el mismo convenio de las variables.

Declaración de clases

Los nombres de las clases serán sustantivos y comenzarán con letra inicial mayúscula. Si un nombre es compuesto, se escribirá la letra inicial de cada palabra, evitando usar el guion bajo.

Código fuente 3.1. Ejemplo del estándar de codificación.

```
namespace KMSkeletonTracking
{

    public class KinectSDK
    {

        private static bool validResult(int result) {
            return result == 0;
        }

        public static long getElevationAngle() {
```

```
            long angle = 0;
            bool result = validResult(KinectAPI.
                nuiCameraElevationGetAngle(ref angle));

            if(!result)
                throw new Exception("No se pudo obtener la elevación de
                    la cámara.");
            return angle;
        }

    }

}
```

3.2. Obtención de datos

Para obtener los datos capturados por el *Kinect* en *Unity 3D*, fue necesario utilizar la *SDK* compilada en C++. Las funciones que esta exporta fueron enlazadas en C#, así como las estructuras que en estas se utilizan. A continuación se muestra un ejemplo de cómo se realizó este proceso.

Código fuente 3.2. Ejemplo del enlace de funciones y estructuras.

```
class KinectAPI {
    [DllImportAttribute(@"C:\Windows\System32\Kinect10.dll", EntryPoint = "NuiInitialize")]
    public static extern int nuiInitialize(NuiInitializeFlags dwFlags); \\permite iniciar el proceso de
        captura

    [DllImportAttribute(@"C:\Windows\System32\Kinect10.dll", EntryPoint = "
        NuiSkeletonTrackingEnable")]
    public static extern int nuiSkeletonTrackingEnable(IntPtr hNextFrameEvent,
        NuiSkeletonTrakingFlags dwFlags); \\permite activar la detección de cuerpos
}
```

3.3. Generación de los datos de entrenamiento

Los datos capturados durante la correcta realización de un ejercicio, son guardados en un archivo con formato *XML*, el cual será utilizado por la IA del videojuego para evaluar los ejercicios realizados por los pacientes. En este tipo de archivo los datos son guardados en nodos, también conocidos como etiquetas, las

cuales son identificadas por un nombre y pueden tener varios atributos u otras etiquetas dentro, en forma de árbol jerárquico. En el *(Anexo A.1)* se muestra el esquema del *XML* generado por el módulo, en el cual, dentro de la etiqueta ***SesionData***, se guarda el nombre del ejercicio, las partes del cuerpo que son utilizadas en este, su duración en segundos y la frecuencia con que son grabados los datos en milisegundos. Además en la etiqueta ***SkeletonsInformation*** se almacenan todos los *frames* capturados, dentro de los cuales se encuentran los esqueletos detectados en cada uno de ellos. Dentro de la etiqueta ***Joins*** de los esqueletos se almacenan los *joins*, en los cuales se indican a que parte del cuerpo corresponden, su posición y su orientación, tanto absoluta como jerárquica. En el *(Anexo A.2)* se muestra un ejemplo del archivo generado por el módulo.

3.4. Pruebas

La metodología *XP* enfatiza los aspectos relacionados con las pruebas, clasificándolas en diferentes tipos y funcionalidades específicas, indicando quién, cuándo y cómo deben ser implementadas y ejecutadas. Del buen uso de las pruebas depende el éxito de otras prácticas, tales como la propiedad colectiva del código y la refactorización. Cuando se tienen bien implementadas las pruebas no habrá temor de modificar el código, ya que si este se daña, las pruebas mostrarán el error y permitirán encontrarlo. Según esta metodología se debe ser muy estricto con las pruebas. Solo se deberá liberar una nueva versión si esta ha pasado satisfactoriamente la totalidad de las pruebas. En caso contrario se empleará el resultado de estas para identificar el error y solucionarlo con mecanismos ya definidos. Las pruebas en cada iteración son de gran importancia, pues permiten corregir los errores mientras se programa. De esta forma se van cubriendo todas las dificultades que cada versión presente (Echeverry Tobón y Delgado Carmona, 2007).

3.4.1. Pruebas de aceptación

Las pruebas de aceptación, son básicamente pruebas funcionales sobre el sistema completo que se deben realizar en intervalos regulares y busca validar los resultados obtenidos. El cliente es el encargado de controlar estas pruebas para verificar el correcto funcionamiento del sistema, basándose en los requerimientos reflejados en las historias de usuario. En todas las iteraciones, cada una de las historias de usuario seleccionadas por el cliente deberá tener una o más pruebas de aceptación. Estas pruebas deben identificar los errores que posteriormente serán erradicados y establecer los casos de prueba para cada iteración. Las pruebas de aceptación son pruebas de caja negra, que representan un resultado esperado de determinada transacción con el sistema. Para que una historia de usuario se considere aprobada, deberá pasar todas las pruebas de aceptación elaboradas para dicha historia. Mientras mayor sea el conjunto de pruebas de aceptación de historias de usuario satisfactorias, mayor será la garantía de buenos resultados en el final del proyecto (Echeverry Tobón y Delgado Carmona, 2007; Rodriguez y Bonilla, 2005).

Tabla 3.1. Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Integrar el módulo con *Unity 3D*.	
Descripción: Prueba para verificar la integración del módulo en *Unity 3D*.	
Condiciones de ejecución: • El cliente debe comprobar que el módulo puede ser utilizado en *Unity 3D*.	
Pasos de ejecución: • Confirmar que el módulo puede ser utilizado en *Unity 3D*.	
Resultados esperados: • El módulo se integra correctamente en diferentes aplicaciones desarrolladas en *Unity 3D*.	
Evaluación: Satisfactoria	

Tabla 3.2. Prueba de aceptación # 2

Caso de prueba de aceptación	
Código: HU2_P2	**Historia de usuario:** 2
Nombre: Detectar conexión del *Kinect*.	
Descripción: Prueba para la funcionalidad detectar conexión del *Kinect*.	
Condiciones de ejecución: • El cliente debe comprobar que se realice la conexión del dispositivo correctamente.	
Pasos de ejecución: • Verificar que se conecte el dispositivo en la aplicación gráfica.	
Resultados esperados: • El módulo detecta si el *Kinect* está conectado.	
Evaluación: Satisfactoria	

Tabla 3.3. Prueba de aceptación # 3

Caso de prueba de aceptación	
Código: HU3_P3	**Historia de usuario:** 3
Nombre: Detectar posición de los *joins*.	
Descripción: Prueba para la funcionalidad detectar la posición de los *joins*.	

Continúa en la próxima página

Tabla 3.3. Continuación de la página anterior

Condiciones de ejecución: El usuario debe haber: • Enlazado la interfaz *KinectInterface*. • Iniciado el proceso de captura.
Pasos de ejecución: • El usuario inicia el proceso de captura. • El usuario enlaza el evento *NewSkeletonFrame* para obtener la posición de los *joins* en cada *frame*.
Resultados esperados: • Se captura exitosamente la posición de los *joins* en cada *frame*.
Evaluación: Satisfactoria

Tabla 3.4. Prueba de aceptación # 4

Caso de prueba de aceptación	
Código: HU4_P4	**Historia de usuario:** 4
Nombre: Determinar orientación de los *joins*.	
Descripción: Prueba para la funcionalidad detectar la orientación de los *joins*.	
Condiciones de ejecución: El usuario debe haber: • Enlazado la interfaz *KinectInterface*. • Iniciado el proceso de captura.	
Pasos de ejecución: • El usuario inicia el proceso de captura. • El usuario enlaza el evento *NewSkeletonFrame* para obtener la orientación de los *joins* en cada *frame*.	
Resultados esperados: • Se captura exitosamente la orientación de los *joins* en cada *frame*.	
Evaluación: Satisfactoria	

Tabla 3.5. Prueba de aceptación # 5

Caso de prueba de aceptación	
Código: HU5_P5	**Historia de usuario:** 5
Nombre: Generar datos de entrenamiento.	
Descripción: Prueba para la funcionalidad de generación de datos de entrenamiento para la IA del videojuego Danzo-Terapia.	
Condiciones de ejecución: El usuario debe haber: • Enlazado la interfaz *KinectInterface*. • Iniciado el proceso de captura de datos de entrenamiento.	
Pasos de ejecución: • El usuario inicia el proceso de captura de datos de entrenamiento. • Una persona realiza un ejercicio de la forma adecuada frente al *Kinect*. • Los datos capturados son guardados en un archivo *XML* para su posterior utilización por la IA del videojuego.	
Resultados esperados: • Los datos capturados durante la correcta ejecución del ejercicio son guardados satisfactoriamente en el archivo *XML*.	
Evaluación: Satisfactoria	

3.5. Validación del sistema

La validación es el proceso en el cual se comprueba que el sistema desarrollado cumple con las necesidades del usuario. Con las pruebas realizadas se constató que el módulo creado puede ser utilizado en aplicaciones desarrolladas en *Unity 3D*. Este además puede obtener la posición y orientación de las principales partes del cuerpo durante el proceso de captura de movimiento, así como generar un archivo con estos datos durante una sesión de entrenamiento. Los datos capturados por el módulo *KMSkeletonTracking* son devueltos según la frecuencia con que este se configure, estableciéndose 100 milisegundos como frecuencia estándar. Este también puede configurarse para que utilice la cámara a color del *Kinect*, así como para modificar la elevación del dispositivo, con el objetivo de realizar una mejor captura, encuadrando mejor al paciente.

Conclusiones parciales

En este capítulo se describió el estándar de codificación utilizado en la implementación del módulo *KMSkeletonTracking*. Se mostró cómo se realizó el enlace de las funciones de la biblioteca implementada en C++ para poder utilizarlas en *Unity 3D*. Además, se mostraron las pruebas de aceptación realizadas a cada una de las historias de usuario y los resultados que en estas se obtuvieron. Estos resultados fueron constatados durante la validación de la solución, en la que se comprobó que el módulo desarrollado cumple con las expectativas iniciales, capturando la posición y orientación de las principales partes del cuerpo durante la realización de un ejercicio.

Conclusiones

Con la realización de este trabajo se arribaron a las siguientes conclusiones:

- Con el módulo desarrollado, el dispositivo *Kinect* puede integrarse con el videojuego Danzo-Terapia, permitiéndole detectar los movimientos realizados por los pacientes para poder evaluarlos.
- Los datos capturados por el módulo pueden ser guardados en un archivo con el objetivo de ser utilizados por una Inteligencia Artificial como parte de su base de conocimiento, para detectar diferentes posturas o gestos.

Recomendaciones

Con el objetivo de realizar una mejor captura de los movimientos realizados por los pacientes se recomienda la utilización de un algoritmo que infiera la posición y orientación de los *joins* durante la oclusión de una parte del cuerpo.

Glosario

LGPL Abreviatura de *Lesser General Public License*, es una licencia de *software* de código abierto que detalla cómo este y su código fuente pueden ser copiados, modificados y distribuidos libremente. 17

XML Siglas en inglés de *Extensible Markup Language*, 'lenguaje de marcas extensible', es un lenguaje de marcas desarrollado por el *World Wide Web Consortium (W3C)* utilizado para almacenar datos en forma legible. 31, 34, 35, 38

driver Un controlador de dispositivo (*driver* en inglés), es un programa informático que permite al sistema operativo interactuar con un periférico. 17

ictus cerebral interrupción del flujo sanguíneo al cerebro y el consecuente aporte de oxígeno que necesita para funcionar, su efecto depende de la intensidad y de la zona cerebral afectada. 1

LED Sigla de la expresión inglesa *light-emitting diode*, 'diodo emisor de luz'. 5, 7

Acrónimos

DFF *Depth From Focus*. 10

DFS *Depth From Stereo*. 10

FPS *Frames per second*. 7, 9

GOF *Gang of Four*. 28

GRASP *General Responsibility Assignment Software Patterns*. 28

IDE *Integrated Development Environment*. 17

PS2 *PlayStation 2*. 7

PS3 *PlayStation 3*. 7, 17

PS4 *PlayStation 4*. 8

RUP *Rational Unified Process*. 15

SDK *Software Development Kit*. 14, 16–18, 20, 24, 27, 31, 34

XP *Extreme Programming*. 15, 17, 20, 26, 29, 32, 35

2D bidimensional. 17

3D tridimensional. 1, 7, 9, 14, 17

CRC Clase-Responsabilidad-Colaboración. 29

IA Inteligencia Artificial. 10, 18, 23, 25, 27, 30, 31, 34, 38

OMS Organización Mundial de la Salud. 1

RV Realidad Virtual. 1, 2

TIC Tecnologías de la Informática y las Comunicaciones. 1

UCI Universidad de las Ciencias Informáticas. 2, 14

Referencias bibliográficas

Anaya Villegas, Adrian (2007). *A propósito de programación extrema XP* (vid. págs. 20, 22).

Beltrán Álvarez, Anxo et al., (2004). *MoCap Tecnologías de captura de movimiento*. URL: http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/Peliculas/Mocap/index.htm (vid. págs. 6, 12).

Borenstein, Greg (2012). *Making Things See*. O'Reilly Media, Inc (vid. págs. 10, 17).

Butler, Daniel Paul y Keith Willett (2010). «Wii-habilitation: is there a role in trauma?» En: *Injury* 41.9, págs. 883-885 (vid. pág. 14).

Calero, Manuel (2003). «Una explicación de la programación extrema (XP)». En: *MADRID* (vid. pág. 15).

Canós, José H, Patricio Letelier y Mª Carmen Penadés (2003). «Metodologías Ágiles en el desarrollo de Software». En: *VIII Jornadas de Ingeniería de Software y Bases de Datos, JISBD* (vid. págs. 15, 24).

DiCYT (2009). *El mando de la consola Wii puede convertirse en un instrumento de gran valor en el campo de la Medicina*. URL: http://www.dicyt.com/noticias/el-mando-de-la-consola-wii-puede-convertirse-en-un-instrumento-de-gran-valor-en-el-campo-de-la-medicina (vid. pág. 14).

Echeverry Tobón, Luís Miguel y Luz Elena Delgado Carmona (2007). *Caso práctico de la metodología ágil XP al desarrollo de software* (vid. págs. 20, 22, 23, 26, 29, 32, 35).

Escribano, Gerardo Fernández (2002). *Introducción a Extreme Programming* (vid. págs. 15, 22).

Fernández Baena, Adso, Antonio Susin y Xavier Lligadas (2012). «Biomechanical validation of upper-body and lower-body joint movements of kinect motion capture data for rehabilitation treatments». En: *Intelligent Networking and Collaborative Systems (INCoS), 2012 4th International Conference on*. IEEE, págs. 656-661 (vid. pág. 13).

Freeman, Eric et al., (2005). *Head First Design Patterns*. ISBN: 0-596-00712-4. O'Reilly Media, Inc (vid. pág. 29).

Gamma, Erich et al., (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley (vid. pág. 28).

González, Ramón Méndez (2006). *Análisis Xbox Live Vision Xbox 360*. URL: http://www.meristation.com/xbox-360/xbox-live-vision/analisis-juego/1522201 (vid. pág. 9).

Holden, Maureen K (2005). «Virtual environments for motor rehabilitation: review». En: *Cyberpsychology & behavior* 8.3, págs. 187-211 (vid. pág. 2).

Inc., Sony Computer Entertainment (2013). *Sony Computer Entertainment Introduces Wireless Controller For Playstation®4 (Dualshock®4) And Playstation®4 Eye*. Sony Computer Entertainment Inc. URL: http://www.scei.co.jp/corporate/release/130221b_e.html (vid. pág. 8).

Isaac, Mike y John Paczkowski (2013). *Apple Confirms Acquisition of 3-D Sensor Startup PrimeSense*. AllThingsD. URL: http://allthingsd.com/20131124/apple-confirms-acquisition-of-3d-sensor-startup-primesense/ (vid. pág. 17).

Jacobson, Ivar, Grady Booch y James Rumbaugh (1999). *The Unified Software Development Process*. Ed. por Addison Wesley (vid. pág. 15).

Kandil, Amr, Makarand Hastak y Phillip S Dunston (2014). «Application of Microsoft Kinect sensor for tracking construction workers». En: *Bridges* 10, págs. 9780784412329-087 (vid. pág. 9).

Lange, BS et al., (2010). «The potential of virtual reality and gaming to assist successful aging with disability». En: *Physical medicine and rehabilitation clinics of North America* 21.2, págs. 339-356 (vid. pág. 14).

Larman, Craig (1999). *UML y Patrones*. Pearson (vid. págs. 28, 29).

LLC, Sony Computer Entertainment America (2014). *PlayStation Move*. URL: https://www.playstation.com/en-us/explore/accessories/playstation-move/ (vid. pág. 8).

MacCormick, John (2011). «How does the kinect work?» En: *Presentert ved Dickinson College* 6 (vid. pág. 10).

Maresh, Nathan (2014). «Effects of Active Video Games in the Rehabilitation of Ankle Sprains and Chronic Ankle Instability». En: *Journal of Health Informatics* (vid. pág. 13).

Marks, Richard (2010). *EyeToy Innovation and beyond*. Senior Researcher, Sony Computer Entertainment America. URL: http://blog.us.playstation.com/2010/11/03/eyetoy-innovation-and-beyond/comment-page-2/#comment-478157 (vid. pág. 7).

MetaMotion (2011a). *Gypsy 7 Motion Capture System*. URL: http://www.metamotion.com/gypsy/gypsy-motion-capture-system.htm (vid. pág. 12).

— (2011b). *IGS-190M Inertial Motion Capture System*. URL: http://www.metamotion.com/gypsy/IGS-190Sale.html (vid. pág. 11).

Microsoft Corporation (2013). *Kinect for Windows SDK 1.8*. URL: http://msdn.microsoft.com/en-us/library/hh855348.aspx (vid. pág. 17).

Muijzer, Frodo (2014). «Development of an automated exercise Detection and Evaluation system using the Kinect depth camera.» En: (vid. pág. 14).

Muñoz Cardona, John E., Oscar A. Henao Gallo y José F. López Herrera (2013). «Sistema de Rehabilitación basado en el Uso de Análisis Biomecánico y Videojuegos mediante el Sensor Kinect». En: *Tecno Lógicas*, págs. 43-54 (vid. págs. 13, 14).

OptiTrack (2015). *OptiTrack Hardware*. URL: http://www.optitrack.com/hardware/ (vid. pág. 6).

Ortega, H et al., (2001). «Sistema de captura de movimiento para la emulación de interfaces hombre/máquina en ambientes virtuales». En: *MEMORIAS SOMI XV* (vid. pág. 4).

Pressman, Roger S. (2010). *Sotware Engineering. A Practitioner´s Approach.* Ed. por McGraw Hill. Seventh Edition (vid. págs. 15, 16, 19, 26, 27).

Reynoso, Carlos y Nicolás Kicillof (2004). «Estilos y Patrones en la Estrategia de Arquitectura de Microsoft». En: *Universidad de Buenos Aires* (vid. pág. 27).

Rodriguez, Cristian A y José E Bonilla (2005). «Pruebas en Programación Extrema». En: *IV SIMPOSIO INTERNACIONAL DE SISTEMAS DE INFORMACIÓN E*, pág. 126 (vid. pág. 35).

Schechner, Yoav Y y Nahum Kiryati (2000). «Depth from defocus vs. stereo: How different really are they?» En: *International Journal of Computer Vision* 39.2, págs. 141-162 (vid. pág. 10).

Seco, José Antonio González (2001). *El lenguaje de programación C#* (vid. pág. 16).

Shotton, Jamie et al., (2013). «Real-time human pose recognition in parts from single depth images». En: *Communications of the ACM* 56.1, págs. 116-124 (vid. pág. 10).

Sommerville, Ian (2007). *Software Engineering*. Eighth Edition. International computer science series. Addison-Wesley. ISBN: 9780321313799 (vid. pág. 19).

SouVR (2008). *Ascension MotionStar Wireless 2*. URL: `http://en.souvr.com/product/200712/266.html` (vid. pág. 12).

Stocker, Sarah (2007). *PlayStation Eye, A Little More Info*. URL: `http://blog.us.playstation.com/2007/10/10/playstation-eye-a-little-more-info/` (vid. pág. 8).

Suárez, Alcides Alvear y Graciela E Quintero Ramírez (2012). «Ambientes virtuales para rehabilitación física y cognitiva». En: *Tenth LACCEI* (vid. pág. 2).

Taylor, Marco Chacón y Esteban Ortiz Cubero (2013). *Desarrollo de una herramienta para captura del movimiento humano para el análisis biométrico* (vid. pág. 11).

Technologies, Unity (2013). *La mejor plataforma de desarrollo para crear juegos*. URL: `http://unity3d.com/es/unity` (vid. pág. 17).

Vlasic, Daniel et al., (2007). «Practical motion capture in everyday surroundings». En: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM, pág. 35 (vid. págs. 6, 11, 12).

Apéndices

APÉNDICE A

Datos de entrenamiento

A.1. Esquema del *XML* generado por el módulo

Código fuente A.1. *DTD* del *XML* generado por el módulo.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- KMSkeletonTraining DTD -->
<!ELEMENT KMSkeletonTraining ((SesionData, SkeletonsInformation))>
<!ELEMENT SesionData ((Name, SkeletonType, Duration, Frecuency))>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT SkeletonType (#PCDATA)>
<!ELEMENT Duration (#PCDATA)>
<!ELEMENT Frecuency (#PCDATA)>
<!ELEMENT SkeletonsInformation ((KMSkeletonFrame+))>
<!ELEMENT KMSkeletonFrame ((Skeletons))>
<!ELEMENT Skeletons ((Skeleton+))>
<!ELEMENT Skeleton ((Joins))>
<!ELEMENT Joins ((Join+))>
<!ELEMENT Join ((Position, AbsoluteRotation, HierarchicalRotation))>
<!ATTLIST Join
    bodyPart (WristRight | WristLeft | Spine | ShoulderRight | ShoulderLeft |
        ShoulderCenter | KneeRight | KneeLeft | HipRight | HipLeft | HipCenter |
        Head | HandRight | HandLeft | FootRight | FootLeft | ElbowRight | ElbowLeft
         | AnkleRight | AnkleLeft) #REQUIRED
>
<!ELEMENT Position ((x, y, z))>
<!ELEMENT AbsoluteRotation ((x, y, z, w))>
```

```
<!ELEMENT HierarchicalRotation ((x, y, z, w))>
<!ELEMENT x (#PCDATA)>
<!ELEMENT y (#PCDATA)>
<!ELEMENT z (#PCDATA)>
<!ELEMENT w (#PCDATA)>
```

A.2. Archivo generado durante la realización de un ejercicio

Código fuente A.2. Ejemplo del *XML* generado por el módulo con los datos capturados.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE KMSkeletonTraining SYSTEM "KMSkeletonTracking.dtd">
<KMSkeletonTraining>
  <SesionData>
    <Name>Ejercicio de prueba</Name>
    <SkeletonType>All</SkeletonType>
    <Duration>60</Duration>
    <Frecuency>100</Frecuency>
  </SesionData>
  <SkeletonsInformation>
    <KMSkeletonFrame>
      <Skeletons>
        <Skeleton>
          <Joins>
            <Join bodyPart="Head">
              <Position>
                <x>0.155148</x> <y>-0.029814</y> <z>1.490763</z>
              </Position>
              <AbsoluteRotation>
                <x>0.094966</x> <y>0.914627</y> <z>-0.065914</z> <w>0.387361</w>
              </AbsoluteRotation>
              <HierarchicalRotation>
                <x>0.156783</x> <y>-0.029333</y> <z>0.184147</z> <w>0.969841</w>
              </HierarchicalRotation>
            </Join>
            <Join bodyPart="ShoulderLeft">
              <Position>
```

```
      <x>0.015609</x> <y>-0.327522</y> <z>1.408723</z>
    </Position>
    <AbsoluteRotation>
      <x>0.846156</x> <y>-0.405037</y> <z>0.282857</z> <w>-0.198325</w>
    </AbsoluteRotation>
    <HierarchicalRotation>
      <x>-0.004269</x> <y>0.006747</y> <z>-0.830615</z> <w>0.557215</w>
    </HierarchicalRotation>
  </Join>
  <Join bodyPart="ElbowLeft">
    <Position>
      <x>-0.006504</x> <y>-0.455573</y> <z>1.317467</z>
    </Position>
    <AbsoluteRotation>
      <x>0.9191627</x> <y>-0.147755</y> <z>0.239752</z> <w>-0.275124</w>
    </AbsoluteRotation>
    <HierarchicalRotation>
      <x>0.106221</x> <y>-0.139024</y> <z>-0.217941</z> <w>0.960334</w>
    </HierarchicalRotation>
  </Join>
  <Join bodyPart="WristLeft">
    <Position>
      <x>-0.030962</x> <y>-0.595883</y> <z>1.225356</z>
    </Position>
    <AbsoluteRotation>
      <x>0.925937</x> <y>-0.143633</y> <z>0.238522</z> <w>-0.254919</w>
    </AbsoluteRotation>
    <HierarchicalRotation>
      <x>-0.019995</x> <y>-0.000195</y> <z>-0.009742</z> <w>0.999522</w>
    </HierarchicalRotation>
  </Join>
  <Join bodyPart="HandLeft">
    <Position>
      <x>-0.038733</x> <y>-0.639994</y> <z>1.199046</z>
    </Position>
    <AbsoluteRotation>
      <x>0.931573</x> <y>-0.139473</y> <z>0.238432</z> <w>-0.235561</w>
```

```
        </AbsoluteRotation>
        <HierarchicalRotation>
          <x>-0.018317</x> <y>0.000159</y> <z>-0.008816</z> <w>0.999796</w>
        </HierarchicalRotation>
      </Join>
      <Join bodyPart="ShoulderRight">
        <Position>
          <x>0.252124</x> <y>-0.364054</y> <z>1.607362</z>
        </Position>
        <AbsoluteRotation>
          <x>0.827034</x> <y>0.435311</y> <z>0.337584</z> <w>0.112859</w>
        </AbsoluteRotation>
        <HierarchicalRotation>
          <x>-0.001838</x> <y>-0.005759</y> <z>0.942461</z> <w>0.334693</w>
        </HierarchicalRotation>
      </Join>
      <Join bodyPart="ElbowRight">
        <Position>
          <x>0.332364</x> <y>-0.499623</y> <z>1.590692</z>
        </Position>
        <AbsoluteRotation>
          <x>0.962386</x> <y>0.263466</y> <z>-0.039138</z> <w>-0.043243</w>
        </AbsoluteRotation>
        <HierarchicalRotation>
          <x>0.250558</x> <y>-0.308727</y> <z>0.212647</z> <w>0.892866</w>
        </HierarchicalRotation>
      </Join>
    </Joins>
  </Skeleton>
 </Skeletons>
</KMSkeletonFrame>
</SkeletonsInformation>
</KMSkeletonTraining>
```