Diseño de Algoritmo para Rehabilitación Virtual basado en la Cámara Kinect

Cortés, Guillermo, Aguilar, Gustavo y Dueñas, Oscar
Departamento de Informática y Electrónica, Universidad José Simeón Cañas UCA, El Salvador
gcortes@uca.edu.sv

***Abstract*—El presente trabajo de investigación está enfocado en el desarrollo de un algoritmo informático que permita detectar los movimientos de un paciente en terapia de rehabilitación y representarlos en un modelo humano tridimensional dentro de un entorno virtual. Para lograr la representación virtual del paciente se utiliza tecnología de captura de movimiento (MoCap) por medio de la cámara Kinect de Microsoft. Por otro lado, la ejecución de los ejercicios de rehabilitación es manejada a partir de un Autómata Finito Determinista (DFA) que identifica los estados del paciente en cada fase del ejercicio. La máquina de estados finita que representa este autómata se encarga de validar las acciones que el paciente realiza. La validación de las acciones se lleva a cabo a partir de la información que los ejercicios, previamente definidos, establecen. El software utiliza un enfoque data-driven para el flujo de la información. Por otra parte, la representación virtual del paciente se realiza a partir del motor de videojuegos Unity3D. Los resultados del proyecto reflejan la flexibilidad y viabilidad del algoritmo para el uso de la rehabilitación virtual, ya que es capaz de soportar una gran variedad de ejercicios y adecuarse a las condiciones del paciente.**

Palabras claves—Captura de Movimientos, Realidad Virtual, Rehabilitación Virtual, Kinect, Amputación, Rehabilitación de la Marcha

I. INTRODUCCIÓN

La rehabilitación virtual es una alternativa moderna para tratar a pacientes con discapacidades mediante el uso de las tecnologías de realidad virtual, realidad aumentada y captura de movimiento. En este tipo de rehabilitación los pacientes se someten a un ambiente virtual en el que desarrollan los ejercicios correspondientes a su tratamiento.

El sistema médico salvadoreño presenta muchas deficiencias al ofrecer servicios de rehabilitación debido a la alta demanda y a los recursos limitados que presentan muchas instituciones. De igual manera muchas personas no pueden afrontar los costos de clínicas privadas para estos tratamientos. Actualmente en El Salvador no existe clínica alguna que ofrezca servicios de rehabilitación virtual, la cual se presenta como una alternativa para resolver los problemas mencionados anteriormente.

El objetivo del presente trabajo es el desarrollo de un algoritmo informático que puede emplearse en un sistema de rehabilitación virtual. El algoritmo se enfoca en la detección de ejercicios de rehabilitación en pacientes con amputaciones en extremidades inferiores. Para ello se desarrolló un software empleando el motor de videojuegos Unity3D y escrito en el lenguaje de programación C#. El programa emplea la cámara Kinect para capturar los movimientos del usuario. La integración de la Kinect con Unity3D se logró mediante la instalación del paquete *Kinect with MS-SDK* creado por Rumen Filkov, ver [12] y el SDK que Microsoft proporciona para poder manipular los principios

II. MATERIALES Y MÉTODOS

A. Tecnologías utilizadas

El software fue escrito en el lenguaje de programación C# y desarrollado mediante el motor de videojuegos Unity3D. El proyecto emplea la cámara Kinect de la consola Xbox 360 para capturar los movimientos del usuario. Para poder controlar la Kinect y acceder a sus flujos de datos dentro del programa se empleó el Kinect SDK versión 1.8 provisto por Microsoft. Este SDK permite detectar las posiciones de hasta 20 articulaciones humanas, las cuales se muestran en la figura 1.

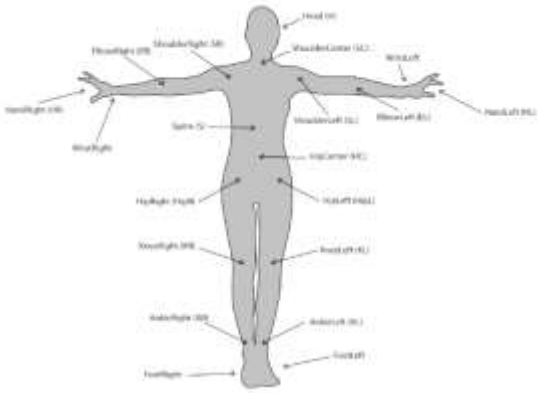

Fig. 1. Articulaciones detectadas por la Kinect.
Tomado de Kiforenko [2013: p. 17]

La integración del Kinect SKD con Unity3D se logró mediante el paquete *Kinect with MS-SDK* creado por Rumen Filkov. Dicho paquete se encuentra disponible de forma gratuita en la tienda de assets de Unity3D a la fecha de este artículo. La lista completa de software empleado en el desarrollo del proyecto se muestra en la tabla 1.

TABLA 1
LISTA DE SOFTWARES EMPLEADOS

Nombre	Versión	Descripción
Unity3D	4.6.1	Plataforma de desarrollo de videojuegos en 3D.

Kinect SDK	1.8	Kit de desarrollo de software para aplicaciones basadas en la Kinect.
Kinect with MS-SDK	1.12	Paquete gratuito de scripts en C# para el manejo de modelos tridimensionales en Unity empleando la Kinect versión 1.
Xamarin Studio	5.9.2	Entorno de desarrollo integrado (IDE) gratuito para la elaboración de aplicaciones en C#, entre otros lenguajes de programación.
Git	2.4.6	Software para el control de versiones, trabajo distribuido y mantenimiento de código fuente.
Bitbucket	--	Aplicación web para el almacenamiento y administración de repositorios de código fuente usando el sistema de versionamiento Git.
Blender	2.75a	Software multiplataforma para el modelado, animación y creación de gráficos tridimensionales.
Windows	Win 8	Sistema operativo.

B. Algoritmo base

El paquete *Kinect with MS-SDK* incluye una implementación de un algoritmo de detección de gestos. Un gesto se define como un conjunto de cambios de posiciones por las cuales deben de pasar una o más articulaciones en un tiempo determinado. Cada gesto está dividido en dos o más estados.

El primer estado de cada gesto (estado inicial) indica las posiciones en las que se deben de encontrar las articulaciones para empezar a detectar el gesto. Los estados siguientes verifican si las articulaciones han llegado a la posición deseada (o se han desplazado lo suficiente con respecto al estado anterior) de una parte del gesto. Si todas las partes del gesto son realizadas exitosamente dentro de un intervalo de tiempo (y por ende se llega al estado final), el gesto se considera completado. De lo contrario se considera como cancelado y puede volver a detectar desde el inicio.

Por ejemplo, consideremos el gesto de "mover las caderas hacia la derecha" como se aprecia en la figura 2.

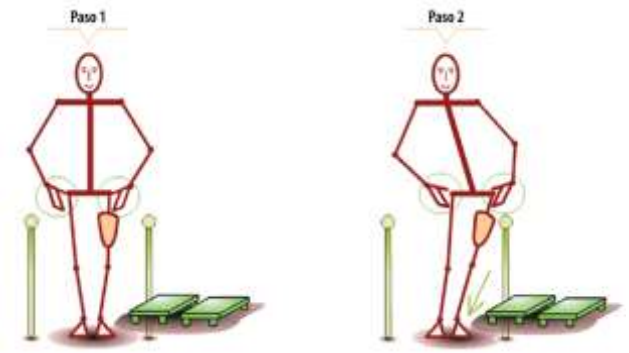

Fig. 2. Movimiento de caderas hacia la derecha.

En dicho gesto, el estado inicial consiste en detectar que el usuario se encuentre de pie con las manos en la cintura. Si se cumplen estas condiciones se puede considerar que el usuario está listo para realizar el gesto completo, y por lo tanto se pasa al siguiente estado. En el siguiente estado se busca que el usuario haya desplazado sus caderas hacia la derecha una cierta distancia. Para ellos se compara la posición actual de las caderas con su posición original en el estado inicial, si se cumple la distancia establecida entonces el gesto se considera completado.

El script base del paquete Kinect with MS-SDK es *KinectManager.cs*. Este script define la clase *KinectManager*, cuyo principal objetivo es la detección de los gestos. *Kinect with MS-SDK* puede detectar hasta dos jugadores simultáneamente. A cada jugador se le asigna un conjunto de gestos que pueden realizar. El *KinectManager* mantiene un array con las posiciones actualizadas de las articulaciones de cada jugador. Todas las posiciones se representan mediante vectores de tres dimensiones usando la clase *Vector3* que Unity3D provee.

Por cada gesto a detectar se crea una instancia del struct *GestureData*. Este struct almacena información del progreso de un gesto específico a medida que se avanzan por los estados del mismo. Los gestos están definidos en el enumerado llamado *Gestures* dentro de la clase *KinectGestures*. La clase *KinectGestures* posee varios métodos estáticos de entre los cuales hay que destacar el método *CheckForGesture*, el cual contiene toda la lógica de detección de los gestos. Este método está compuesto por una estructura de control *switch*, en donde cada sección (o case) corresponde con la definición de los estados de un gesto como se presenta a continuación:

```
switch(gestureData.gesture)
{
   case Gestures.Gesto0:
       switch(gestureData.state)
       {
           case 0:
               // Lógica del estado 0 (estado inicial)
               break;
           case 1:
               // Lógica del estado 1
               break;
           ...
           case n:
               // Lógica del estado n (estado final)
               break;
       }
       break;
    case Gestures.Gesto1:
   ...
}
```

En el estado inicial de cada gesto (estado 0) se valida la pose de inicio del gesto. El *KinectManager* ejecuta el método *CheckForGestures* en cada frame para todos los gestos que se quieren detectar. Cuando el usuario cumple las condiciones para pasar al siguiente estado de un ejercicio, se actualizan los campos de la instancia de *GestureData* recibida como parámetro. Los campos que se actualizan son: el estado y el progreso actual del gesto, la posición de la articulación principal, y el *timestamp* del gesto.

1) Manejo de modelo 3D: El paquete Kinect with MS-SDK también incluye un script que permite controlar un modelo humano 3D mediante los movimiento detectados por la Kinect. Este script es el *AvatarController.cs*. La clase *AvatarController* del script mencionado posee una serie de variables de tipo *Transform* las cuales deben de asociarse con cada una de las articulaciones del modelo

tridimensional. Para que un modelo humano 3D puede ser controlado usando el *AvatarControler* es necesario que posea un esqueleto de huesos y articulaciones. Cada hueso del modelo es manejado por un *GameObject* en Unity3D y como tal posee un componente *Transform* con los datos de posición, rotación y escala.

2) Limitantes del algoritmo base: El paquete *Kinect with MS-SDK* contiene un conjunto de scripts bastante completo para integrar el SDK de la Kinect con Unity3D. Sin embargo, cuenta con algunas limitantes en la detección de gestos, entre las cuales están:

- *Poca flexibilidad para añadir nuevos gestos:* Debido a que los gestos están definidos en el código fuente, agregar nuevos gestos (o modificar los existentes) implica volver a compilar la aplicación. Esto impide que personas no conocen de programación puedan añadir sus propios gestos.
- *Poca escalabilidad para manejar una gran cantidad de gestos:* A medida que se agregan más gestos a la aplicación, el tiempo de búsqueda de un gesto en el método *CheckForGestures* disminuye. Esto se debe a la manera en cómo la lógica de cada gesto se distribuye en en un switch. Agregar más gestos implica agregar más secciones (cases) al *switch*.
- *Capacidad limitada para guardar información entre estados:* Cuando se avanza de un estado a otro en un gesto, la clase *GestureData* permite guardar la posición de una articulación principal para tenerla como referencia en el siguiente estado. Sin embargo, gestos más complejos requieren comparar las posiciones de múltiples articulaciones con respecto a un estado anterior.
- *Pocas opciones de retroalimentación:* Cuando un gesto es realizado incorrectamente simplemente se indica al usuario que el gesto en cuestión ha sido cancelado. No hay manera de informar en qué parte del gesto se cometió el error.
- *Comportamiento extraño del modelo tridimensional:* En algunos casos el modelo tridimensional puede comportarse de manera extraña. Esto ocurre principalmente al realizar movimientos verticales, como saltar o agacharse.

C. Diseño del algoritmo

Una de las características principales de la rehabilitación virtual es el manejo de una gran variedad de ejercicios dentro del software. Para ello, el sistema debe ser capaz de soportar nuevos ejercicios demostrando robustez en el proceso.

La representación de los ejercicios de rehabilitación requiere una estructura que permita la manipulación de los datos de forma genérica. Es decir, los datos contenidos en cada ejercicio deben ser manejados de manera que, independientemente del ejercicio que se esté evaluando, el software pueda determinar qué acción realizar.

Cada ejercicio está compuesto por una secuencia ordenada de fases. Una fase establece la posición de las articulaciones del paciente en determinado momento. El seguimiento ordenado de cada una de las fases describe la trayectoria de las articulaciones.

Para establecer una estructura de datos que cumpla con estos requisitos es necesario identificar los tipos de ejercicios que existen y las diferencias que hay entre ellos y cada una de sus fases.

1) Dependencia entre articulaciones: La dependencia entre articulaciones es un patrón que consiste en el posicionamiento de una articulación con referencia a otra. Es decir, durante cada una de las fases del ejercicio se evalúa la distancia entre las dos articulaciones para determinar su correcta ejecución.

2) Posicionamiento absoluto de una articulación: El posicionamiento absoluto de las articulaciones es un patrón que consiste en manejar la posición de las articulaciones de forma independiente durante ciertas fases del ejercicio. Es decir, el posicionamiento de una articulación es evaluado a partir de un marco de referencia global y de tres dimensiones, sin tomar en cuenta la presencia de las demás articulaciones.

3) Trayectoria de una articulación: La trayectoria de una articulación es un patrón que permite evaluar la posición actual con respecto a la posición previa de una articulación. Este patrón no establece dependencias con otras articulaciones ni utiliza posiciones absolutas. Para determinar la correcta ejecución de un ejercicio que utilice la trayectoria de una articulación, es necesario establecer un rango de movimiento. Este rango se utiliza para evaluar la posición previa y la posición actual de la articulación. Si el movimiento de la articulación termina en un rango correcto con respecto a la posición anterior, entonces el ejercicio se considera correctamente realizado.

4) Ejercicios con barras paralelas. Debido a que los ejercicios analizados se enfocan en la rehabilitación de extremidades inferiores, las barras paralelas se vuelven un factor importante dentro de las terapias de rehabilitación.

Además, los ejercicios de rehabilitación requieren una estructura de datos que maneje los diferentes *criterios de evaluación* anteriormente establecidos, considerando *los rangos de evaluación* de las articulaciones y que permita identificar las distintas articulaciones. Esta estructura de datos se basa en el estándar JSON (JavaScript Object Notation).

El manejo de rangos se determina a través de un objeto compuesto de *máximos* y *mínimos* para cada uno de los tres ejes del sistema de coordenadas. Los valores son representados por números flotantes. Este objeto recibe el nombre de *Range*.

```
{
   "Range": {
       "MinX": 0.5,
       "MaxX": 0.0,
       "MinY": 0.1,
       "MaxY": 0.0,
       "MinZ": 0.0,
       "MaxZ": 0.0
   }
}
```

A partir del objeto *Range* se construyen los tres tipos de patrones identificados en los ejercicios. Cada uno de los tipos de ejercicio tiene su correspondiente objeto en formato JSON. Estos objetos JSON reciben los siguientes nombres:

1) JointDependencies. Es una colección de objetos *JointDependency*. Estos objetos se encargan del manejo de dependencias entre las articulaciones. Es decir, a partir de una articulación inicial (*SourceJointId*) se evalúan los rangos de posición (*Range*) con respecto a una articulación final (*EndJointId*). La

estructura del JSON del objeto *JointDependency* se muestra a continuación:

```
{
   "SourceJointId": 1,
   "EndJointId": 2,
   "Range": { … }
}
```

2) JointPositions. Es una colección de objetos *JointPosition*. Estos objetos son los encargados de definir las posiciones absolutas de las articulaciones en determinado momento del ejercicio. La estructura del objeto *JointPosition* se muestra a continuación:

```
{
   "JointId": 4,
   "Range": { … }
}
```

3) JointChanges. Es una colección de los objetos *JointChange*. Este objeto se utiliza para la evaluación de la trayectoria de una articulación. La evaluación se hace a partir del objeto *Range*, el cual establece la distancia mínima y máxima entre la posición anterior de la articulación y la posición actual que ocupa en determinada fase del ejercicio. El siguiente JSON es un ejemplo de este formato:

```
{
   "JointId": 4,
   "Range": { … }
   "Error:" {
       "X" : 0.15,
       "Y" : 0.0,
       "Z" : 0.0,
   }
}
```

4) ParallelBars. Esta estructura contempla la orientación, la altura y la separación de las barras por medio de los campos *Rotation*, *Height* y *Separation*, respectivamente. Por otro lado, la estructura también contempla los campos *LeftBarJoints* y *RightBarJoints*, los cuales son dos arreglos para el manejo de las articulaciones relacionadas a la barra correspondiente. La estructura JSON del objeto ParallelBars se muestra a continuación:

```
{
   "ParallelBars": {
     "Rotation": 0.0,
     "Height": 1.0,
     "Separation": 0.75,
     "LeftBarJoints": [],
     "RightBarJoints": [],
   }
}
```

Finalmente se construye la estructura de datos que abarca la información completa un ejercicio. Esta estructura está compuesta de un nombre (*Name*), y de una colección de fases (*Phases*) del ejercicio. Cada fase del ejercicio está definida por un intervalo de tiempo (*Interval*) y los tres tipos de colecciones para el manejo de los diferentes ejercicios. La colección de fases es una lista ordenada de elementos. Este orden establece la secuencia en la que se deben realizar cada una de las fases del ejercicio.

```
{
   "Name": "RightWave",
   "ParallelBars": { … },
   "Phases":
      [
         {
             "Interval": 5.0,
             "JointDependencies": [ ... ],
             "JointPositions": [ ... ],
             "JointChanges": [ ... ]
         },
         ...
      ]
}
```

Habiendo definido la estructura de la información, se procede al diseño de mecanismo que controla las diferentes fases en las que se encuentre el paciente durante la ejecución de los ejercicios. Este mecanismo está basado en un Autómata Finito Determinista (ver [1]) e implementado en una Máquina de Estados Finita Determinista (MEFD).

Para establecer formalmente la MEFD de los ejercicios de rehabilitación, primero se deben definir los estados y símbolos que la componen. El conjunto de estados identificados para el manejo de los ejercicios están definidos de la siguiente manera:

1) Iniciación. Es el estado inicial. Verifica que el paciente se encuentre en la postura inicial del ejercicio.

2) Rastreo. Rastrea todas las fases del ejercicio, verificando que cada una de ellas se cumpla de acuerdo a lo que establecen las restricciones de dichas fases.

3) Inválido. Es un estado final. Muestra al paciente las razones por las que el ejercicio ha sido mal ejecutado.

4) Resultado. Es un estado final. Llegar a este estado implica la correcta realización del ejercicio. En este estado se muestra al paciente un resumen de lo realizado.

Por otro lado, los símbolos, también conocidos como *alfabeto de entrada*, son los elementos encargados de determinar hacia qué estado se hará una transición. Los símbolos que componen la MEFD de los ejercicios de rehabilitación se definen de la siguiente manera:

1) Posición Adquirida. Representa que el paciente ha logrado llevar sus articulaciones a la posición que establece el ejercicio.

2) Ejercicio Completado. Este símbolo se utiliza para determinar que todas las fases han sido realizadas satisfactoriamente bajo las condiciones requeridas por el ejercicio.

3) Ejercicio Fallido. Simboliza que un ejercicio ha sido realizado incorrectamente.

4) Reiniciar Ejercicio. Como su nombre lo dice, este símbolo se utiliza para realizar el ejercicio nuevamente.

A partir de estos estados y símbolos, la MEFD se define como la siguiente tupla de cinco elementos:

Sea $M = (Q, \Sigma, \delta, q_0, F)$, donde:

- $Q = \{Iniciación, Rastreo, Inválido, Resultado\}$
- $\Sigma = \{PosiciónAdquirida,\ EjercicioCompletado,\ EjercicioFallido, ReiniciarEjercicio\}$
- $F = \{Inválido,\ Resultado\}$

- $q_0 = Iniciación$
- La función de transición de la siguiente manera:
  - $\delta(Iniciación,\ PosiciónAdquirida) = Rastreo$
  - $\delta(Rastreo,\ PosiciónAdquirida) = Rastreo$
  - $\delta(Rastreo,\ EjercicioCompletado) = Resultado$
  - $\delta(Rastreo,\ EjercicioFallido) = Inválido$
  - $\delta(Inválido,\ ReiniciarEjercicio) = Inciación$
  - $\delta(Resultado,\ ReiniciarEjercicio) = Iniciación$

Nótese que la definición de la función de transición es el producto cartesiano del conjunto de estados contra el alfabeto de entrada y que cada resultado provee un nuevo estado ($\delta: QX\Sigma \rightarrow$ Q). Sin embargo, en la función de transición de *M* solo se han definido las transiciones cuyo estado y símbolo producen un estado que pertenece a *Q*. Por tanto, hay transiciones que no se consideran en la definición, pero pueden ser representadas como $\delta(q_i, a_j) = \emptyset$, es decir, el estado q_i no posee información para efectuar una transición si lee al símbolo a_j. Para especificar estas transiciones se presenta la tabla 2.

TABLA 2
PRODUCTO CARTESIANO ENTRE SÍMBOLOS Y ESTADOS

Estados	**Alfabeto de entrada**			
	PA	*EC*	*EF*	*RE*
Iniciación	Rastreo	Ø	Ø	Ø
Rastreo	Rastreo	Resultado	Inválido	Ø
Inválido	Ø	Ø	Ø	Iniciación
Resultado	Ø	Ø	Ø	Iniciación

La máquina de estados *M* también puede ser representada a través de un *grafo*. Un grafo es un diagrama compuesto de nodos y arcos. Los nodos están conectados por medio de los arcos. En este caso, los nodos representan los estados y los arcos las transiciones. En la figura 3 se muestra la representación de un *diagrama de transición* para la máquina M.

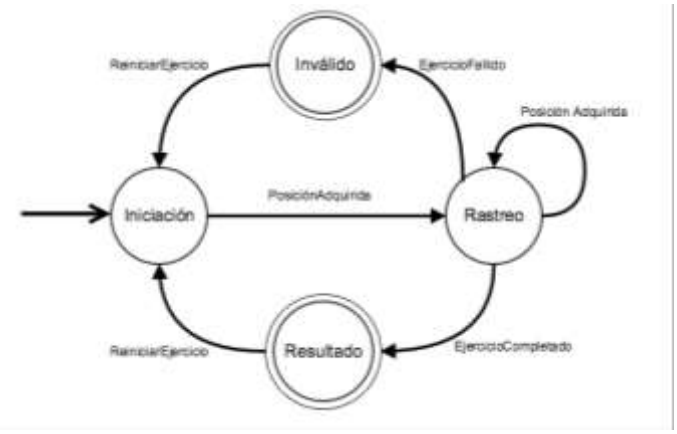

Fig. 3. Grafo dirigido de la máquina de estados finitos determinista M.

Los estados finales están representados por nodos con doble círculo, los arcos dirigidos son flechas que indican el sentido de la transición. Como se especificó anteriormente, el punto de entrada del grafo es el estado inicial del autómata es q0=Iniciación.

III. RESULTADOS Y DISCUSIÓN

Para comprobar la precisión del algoritmo desarrollado en la detección de ejercicios de rehabilitación, se realizaron pruebas con personas reales.

Los ejercicios seleccionados para las pruebas se extrajeron de la guía *"Ejercicios para los amputados de extremidades inferiores - Entrenamiento para la marcha"* realizada por la Universidad Don Bosco. Este guía de ejercicios a su vez está basado en un folleto interno del Comité Internacional de la Cruz Roja (CICR) de 1990, cuyo autor es Theo Verhoeff.

Los ejercicios a realizar en las pruebas se dividieron de acuerdo a su nivel de dificultad como básico, intermedio y avanzado. El nivel de dificultad hace referencia a la cantidad de esfuerzo que el paciente con prótesis necesita realizar para completar el ejercicio.

A. Objetivos de las pruebas

En las pruebas, el software tenía que cumplir con los siguientes objetivos:

- Representar con precisión los movimientos realizados por el usuario en un modelo humano tridimensional.
- Detectar si el usuario ha realizado correctamente los ejercicios de rehabilitación asignados.
- Identificar posibles casos en que un ejercicio realizado incorrectamente sea detectado como válido por el software. Falsos positivos.

Para asegurar consistencia en las pruebas, uno de los desarrolladores del software verificó que cada usuario realizara correctamente los ejercicios de rehabilitación. Si una repetición de un ejercicio se realizada incorrectamente a juicio del desarrollador, dicha repetición se ignoraba. Solo las repeticiones válidas fueron procesadas por el software.

Para identificar casos de falsos positivos, a cada usuario se le permitió realizar movimientos libres con el fin de "engañar" al software para que detectara repeticiones válidas de los ejercicios. Estos falsos positivos fueron posteriormente corregidos.

B. Presentación de resultados

1) Ejercicio básico 1 - Carga parcial del peso (apoyo en ambas manos): Este ejercicio consiste en colocarse de pie en medio de barras paralelas, apoyándose en ambas manos, y luego transferir el peso corporal de la pierna sin prótesis a la pierna con prótesis. Una ilustración de esta figura se puede ver en la figura 4.

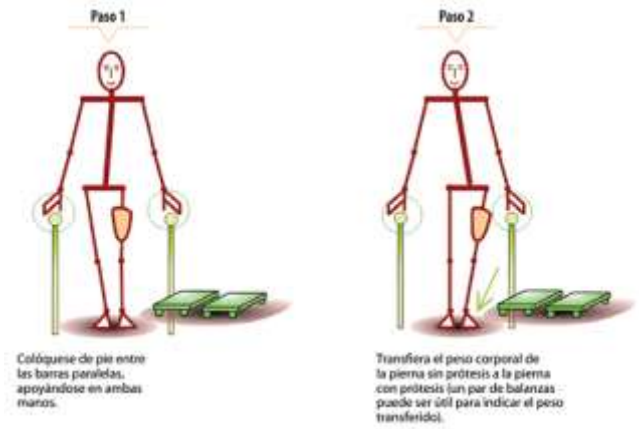

Fig. 4. Ejercicio básico 1 - Carga parcial del peso (apoyo en ambas manos)

En esta prueba se estableció la pierna derecha como la extremidad con prótesis. Para detectar que el usuario ha transferido su peso corporal hacia el lado derecho se verificó que hubiera movido el centro de su cadera hacia esa dirección. Dentro de la escena virtual se colocaron barras paralelas capaces de detectar colisiones físicas con las manos del modelo humano tridimensional. Los resultados de esta prueba se presentan en la tabla 3.

TABLA 3
RESULTADOS DE LAS PRUEBAS REALIZADAS PARA EL EJERCICIO BÁSICO 1

Ejercicio	Carga parcial del peso (apoyo en ambas manos)
Nivel	Básico
Porcentaje de repeticiones detectadas correctamente	88%
Inconsistencias mostradas por el modelo tridimensional	Ninguna
Falsos positivos detectados	1

- Razones de fallo al detectar el ejercicio:
  - El software no detecta que el usuario toque algunas de las barras paralelas.
- Falsos positivos detectados:
  - El usuario puede mover todo el cuerpo hacia un lado (no solo las caderas) y el ejercicio se marca como completado.
- Correcciones realizadas:
  - Se restringió el movimiento de los pies del usuario mientras se realiza el ejercicio. Los pies deben de permanecer fijos.
  - Se ajustó la altura de las barras dentro de la escena virtual.
  - Se ajustó el volumen de colisión de las barras dentro de la escena virtual.

2) Ejercicio intermedio 1 - Paso adelante con la pierna sana (apoyo en ambas manos): Este ejercicio consiste en colocarse de pie en medio de barras paralelas, apoyándose en ambas manos, y luego dar un paso hacia adelante con la pierna sana. La ilustración de este ejercicio se puede consultar en la figura 5.

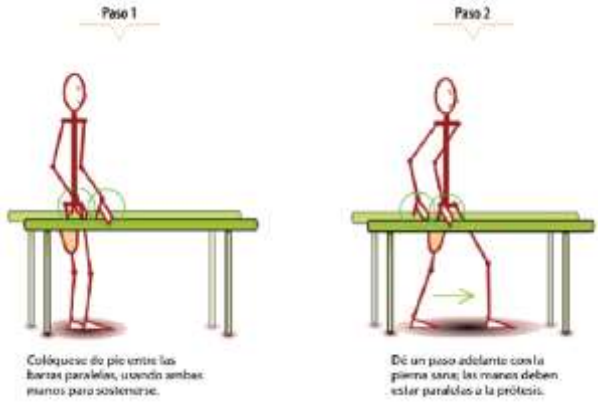

Fig. 5. Ejercicio intermedio 1 - Paso adelante con la pierna sana (apoyo en ambas manos)

En esta prueba se estableció la pierna derecha como la extremidad sana. Para detectar que el usuario haya dado un paso hacia adelante se detectó que el pie derecho se hubiera movido hacia adelante en el eje Z con respecto a la cámara Kinect. Dentro de la escena virtual se colocaron barras paralelas capaces de detectar colisiones físicas con las manos del modelo humano tridimensional.

Los resultados de esta prueba se presentan en la tabla 4.

TABLA 4
RESULTADOS DE LAS PRUEBAS REALIZADAS PARA EL EJERCICIO INTERMEDIO 1

Ejercicio	Paso adelante con la pierna sana (apoyo en ambas manos)
Nivel	Intermedio
Porcentaje de repeticiones detectadas correctamente	90%
Inconsistencias mostradas por el modelo tridimensional	Ninguna
Falsos positivos detectados	1

- Razones de fallo al detectar el ejercicio:
  - El software no detecta que el usuario toque algunas de las barras paralelas.
- Falsos positivos detectados:
  - El usuario puede mover todo el cuerpo hacia adelante (no solo la pierna) y el ejercicio se marca como completado.
- Correcciones realizadas:
  - Se restringió el movimiento del pie de apoyo del usuario.
  - Se ajustó la altura de las barras dentro de la escena virtual.

3) Ejercicio avanzado 1 - Equilibrio sobre la pierna con prótesis: Este ejercicio consiste en mantener el equilibrio sobre la

pierna con prótesis mientras la pierna sana se mantiene levantada. La ilustración de este ejercicio se puede consultar en la figura 6.

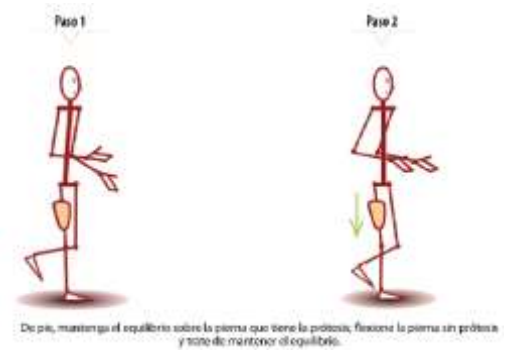

Fig. 6. Ejercicio avanzado 1 - Equilibrio sobre la pierna con prótesis

En esta prueba se estableció la pierna derecha como la extremidad con prótesis. Para detectar que el usuario se ha mantenido en equilibrio sobre la pierna con prótesis se detectó que la pierna sana estuviera separada del suelo una determinada distancia por unos cuantos segundos. Los resultados de esta prueba se presentan en la tabla 5.

TABLA 5
RESULTADOS DE LAS PRUEBAS REALIZADAS PARA EL EJERCICIO AVANZADO 1

Ejercicio	Equilibrio sobre la pierna con prótesis
Nivel	Avanzado
Porcentaje de repeticiones detectadas correctamente	100%
Inconsistencias mostradas por el modelo tridimensional	0
Falsos positivos detectados	0

Este ejercicio no requirió ningún tipo de corrección.

IV. CONCLUSIONES Y RECOMENDACIONES

A. Conclusiones sobre el algoritmo

1) El algoritmo base utilizado presenta problemas de escalabilidad y soporte de nuevos ejercicios. Esto se debe a que cada ejercicio se contempla directamente en el código fuente del algoritmo, por lo tanto, un nuevo ejercicio implica un nuevo cambio en el programa.

2) El algoritmo desarrollado brinda una gran flexibilidad al incorporar nuevos ejercicios. Esto se debe a que los ejercicios solo dependen de la información como tal, y no de cambios drásticos en el software. Sin embargo, este nivel de flexibilidad disminuye el control que se tiene sobre la información, ya que los ejercicios pueden tener errores conceptuales y el software no es capaz de detectarlos.

3) El paquete *Kinect with MS-SDK* provee una serie de scripts que facilitan en gran medida la interoperación entre Unity3D y el SDK de la Kinect.

4) El *AvatarController* del paquete *Kinect with MS-SDK* cumple con las funciones necesarias para manejar un modelo humano tridimensional mediante los movimientos detectados por la Kinect. Sin embargo presenta ciertas deficiencias al manejar movimientos verticales en el modelo.

B. Conclusiones sobre los resultados

1) El algoritmo desarrollado asegura un nivel de precisión bastante aceptable al detectar los movimientos del usuario e interpretarlos como ejercicios de rehabilitación de extremidades inferiores.

2) La interacción con elementos tridimensionales, como las barras paralelas, dificulta la detección correcta de ciertos ejercicios. Esto se debe a que además de interpretar lo movimientos del usuario, se tiene que detectar colisiones con objetos dentro del ambiente virtual.

3) La probabilidad de error al detectar un ejercicio es directamente proporcional a la cantidad de fases que dicho ejercicio posea. Un ejercicio con demasiadas fases puede significar un mal diseño del mismo o que dicho ejercicio es demasiado complejo para ser manejado por el software.

4) El algoritmo desarrollado no es perfecto, existen casos en que el software puede interpretar que el paciente está realizando correctamente el ejercicio cuando realmente no lo está. A esto se le ha denominado como un "falso positivo".

C. Recomendaciones

1) La elaboración y modificación de los ejercicios debe de realizarse a partir de una herramienta externa. Los ejercicios se almacenan en archivos externos al sistema bajo el formato JSON. En este sentido, La manipulación directa de estos archivos puede generar inconsistencias al realizar las terapias de rehabilitación.

2) En el caso de los ejercicios que implican el uso de barras paralelas, hay que asegurarse de que las medidas de rotación, altura y separación de las barras especificadas en la definición JSON del ejercicio correspondan con las medidas en el mundo real. De no ser así, el ejercicio puede ser detectado de manera incorrecta por el software.

3) Para resultados óptimos en la detección de los movimientos del paciente, procurar que la habitación en la que se realizan los ejercicios esté libre de objetos que puedan confundir a la cámara Kinect. Las barras paralelas no han mostrado impacto negativo en la detección de los ejercicios.

4) En el caso de reemplazar el modelo humano tridimensional incluído en el software, asegurarse que el nuevo modelo cuente con un esqueleto articulado y que dichas articulaciones correspondan con las especificadas por el script *AvatarController*. Si alguna articulación no coincide, el modelo puede comportarse de manera extraña.

5) El sistema no maneja ningún módulo de reportes acerca de los ejercicios que realiza el paciente. Esta es una de las principales mejoras a realizarse en las siguientes versiones del software, ya que

producir esta información mejoraría el diagnóstico por parte de los terapeutas.

REFERENCIAS

[1] ABARCA, R. A. (2002). *Notas sobre la teoría de autómatas finitos, expresiones regulares y gramática formal*. San Salvador: UCA editores.

[2] ALBIOL, S. (2014). *Rehabilitación Virtual Motora: una Evaluación al tratamiento de pacientes con Daño Cerebral Adquirido*. Valencia: Universidad Politécnica de Valencia.

[3] CAUDELL, T. y MIZELL, D. (1992). *Augmented Reality: An Application of Heads-Up Display Technology to Manual Manufacturing Processes.* IEEE Hawaii International Conference on Systems Sciences. 7-10 enero (en papel).

[4] COMEAU, C. P. y BRYAN, J. S. (1961). *Headsight television system provides remote surveillance. Electronics*, 86-90

[5] DE MAURO, A (2009). *Virtual Reality Based Rehabilitation and Game Technology*. Vicomtech: San Sebastián.

[6] BRONZINO, J. (2000). *The biomedical engineering handbook*. Biomechanics (18-37). Berlín: Springer-Verlag and Heidelberg GmbH & Co. K.

[7] GANDULLO, J. (2010). *Sistema Simple de Captura de Movimiento*. Sevilla: Editorial de la Universidad de Sevilla.

[8] MAREY, E. J. (1878). *La Machine Animale: Locomotion terrestre et sérienne*. Michigan: Librería de la Universidad de Michigan.

[9] MCCORMACK, A., HILL, P., BROWN, M. y HERBERT, S. (2004). *Eadweard Muybridge: The Kingston Museum Bequest.* Oeste de Sussex: The Projection Box.

[10] RAGHAVACHARY, S. (2004). *Rendering for Beginners: Image synthesis using RenderMan*. RenderMan (25-38). Burlington: Focal Press.

[11] RHEINGOLD, H. (1991). *Virtual Reality*. Filipinas: Summit Books

[12] R. Filkov. (2015) https://www.assetstore.unity3d.com/en/#!/content/7747