UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM ATLACOMULCO

"Sistema de Visión Artificial para la Detección y Corrección de Posturas en Ejercicios realizados por Fisicoculturistas"

T E S I S

Que para obtener el Título de:

Ingeniero en Computación

Presenta:

Juan Manuel Martínez José

Director de Tesis:

Dr. en C. I. E. Everardo Efrén Granda Gutiérrez

Atlacomulco, México; Octubre de 2018

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco

Área de Titulación

Atlacomulco, Méx., a 8 de Mayo de 2018.

P.ICO. JUAN MANUEL MARTÍNEZ JOSÉ
P R E S E N T E .

Por este conducto me permito comunicar a usted (es) que el Comité Técnico de Titulación autorizó el proyecto de TESIS que presentó (aron) y que será asesorado por el (la) DR. EVERARDO EFRÉN GRANDA GUTIÉRREZ titulado:

"SISTEMA DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN Y CORRECCIÓN DE POSTURAS EN EJERCICIOS REALIZADOS POR FISICOCULTURISTAS"

Al mismo tiempo solicito a usted (es) de la manera más atenta que al concluir el desarrollo del trabajo sea (n) tan amable (s) de comunicarlo por escrito a este Departamento.

A T E N T A M E N T E
PATRIA, CIENCIA Y TRABAJO
"2018, Año del 190 Aniversario de la Universidad Autónoma del Estado de México"

M. EN P.C. NEPHTALÍ PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADÉMICO
CENTRO UNIVERSITARIO UAEM ATLACOMULCO

Km. 60 Carretera Toluca – Atlacomulco
C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco

Área de Titulación

Atlacomulco, Méx., a 17 de Abril de 2018.

DR. EVERARDO EFRÉN GRANDA GUTIÉRREZ
P R E S E N T E :

Por este conducto me permito comunicar a usted que la Comisión Revisora de Proyectos de Tesis, acordó nombrarlo ASESOR del Trabajo de TESIS titulado:

"SISTEMA DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN Y CORRECCIÓN DE POSTURAS EN EJERCICIOS REALIZADOS POR FISICOCULTURISTAS"

Que presenta (n) el (los) Pasante (s) de ICO. JUAN MANUEL MARTÍNEZ JOSÉ al mismo tiempo solicito a usted de la manera más atenta, que al concluir su asesoría sea tan amable de comunicarlo por escrito a este Departamento.

Recibí original
Everardo Granda
30-05-2018

A T E N T A M E N T E
PATRIA, CIENCIA Y TRABAJO
"2018, Año del 190 Aniversario de la Universidad Autónoma del Estado de México"

M. EN P.C. NEPHTALÍ PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADÉMICO
CENTRO UNIVERSITARIO UAEM ATLACOMULCO

Km. 60 Carretera Toluca – Atlacomulco
C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco

Atlacomulco, México a 8 de octubre de 2018

M. EN P.C. NEPHTALI PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADEMICO DEL CENTRO
UNIVERSITARIO UAEM ATLACOMULCO

P R E S E N T E:

En contestación a su atento oficio con fecha del 17 de abril de 2018, en el cual se me designó como **ASESOR** del Trabajo de **TESIS** titulado **"Sistema de visión artificial para la detección y corrección de posturas en ejercicios realizados por fisicoculturistas"**, presentado por el pasante de la Licenciatura en Ingeniería en Computación **Juan Manuel Martínez José,** informo que:

He concluido satisfactoriamente la revisión del trabajo, haciendo las observaciones pertinentes, mismas que han sido solventadas por el interesado. Así mismo, **manifiesto que el trabajo cumple con los requisitos y cualidades que corresponden a la opción de evaluación profesional**; por lo cual no tengo ningún inconveniente en dar mi **Voto Aprobatorio** para que el egresado pueda continuar con los trámites de titulación.

Sin otro particular por el momento, agradezco su atención.

A T E N T A M E N T E

DR. EN C. I. E. EVERARDO EFRÉN GRANDA GUTIÉRREZ
PROFESOR DE TIEMPO COMPLETO

Recibí 8-Oct-18

Km. 60 Carretera Toluca – Atlacomulco, C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx
Centro Universitario UAEM Atlacomulco

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco
Departamento de Evaluación Profesional

Atlacomulco, Méx., a 8 de Octubre de 2018.

DR. EN C. EN I. JOSÉ ARTURO PÉREZ MARTÍNEZ
P R E S E N T E:

Por este conducto me permito comunicar a usted que la Comisión Revisora de Proyectos de Tesis acordó nombrarlo **REVISOR(A)** del Trabajo de **TESIS** titulado:

"SISTEMA DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN Y CORRECCIÓN DE POSTURAS EN EJERCICIOS REALIZADOS POR FISICOCULTURISTAS"

Que presenta(n) el (los) Pasante(s) de **ICO. JUAN MANUEL MARTÍNEZ JOSÉ,** al mismo tiempo solicito a usted de la manera más atenta, concluir su revisión sea tan amable en comunicar por escrito su **VOTO APROBATORIO** a este Departamento, en un lapso **no mayor a 10 días a partir de la recepción del trabajo.**

A T E N T A M E N T E
PATRIA, CIENCIA Y TRABAJO
"2018, Año del 190 Aniversario de la Universidad Autónoma del Estado de México"

Recibí Original
08/oct/2018
Dr. Arturo Pérez

M. EN P.C. NEPHTALÍ PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADÉMICO
CENTRO UNIVERSITARIO UAEM ATLACOMULCO

c.c.p. expediente del (la) interesado(a)

Km. 60 Carretera Toluca – Atlacomulco
C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco

Atlacomulco, México a 15 de octubre de 2018

M. EN P.C. NEPHTALI PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADEMICO DEL CENTRO
UNIVERSITARIO UAEM ATLACOMULCO

P R E S E N T E:

En contestación a su atento oficio con fecha del 05 de Octubre de 2018, en el cual se me designó como **REVISOR** del Trabajo de **TESIS** titulado **"Sistema de Visión Artificial para la detección y corrección de posturas en ejercicios realizados por fisicoculturistas"**, presentado por el pasante de la Licenciatura en Ingeniería en Computación **Juan Manuel Martínez José,** informo que:

He concluido satisfactoriamente la revisión del trabajo, haciendo las observaciones pertinentes, mismas que han sido solventadas por el interesado. Así mismo, **manifiesto que el trabajo cumple con los requisitos y cualidades que corresponden a la opción de evaluación profesional**; por lo cual no tengo ningún inconveniente en dar mi **Voto Aprobatorio** para que el egresado pueda continuar con los trámites de titulación.

Sin otro particular por el momento, agradezco su atención.

A T E N T A M E N T E
PATRIA, CIENCIA Y TRABAJO
"2018, Año del 190 Aniversario de la Universidad Autónoma del Estado de México"

DR. EN C. I. E. JOSÉ ARTURO PÉREZ MARTÍNEZ
PROFESOR DE TIEMPO COMPLETO

Km. 60 Carretera Toluca – Atlacomulco, C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx
Centro Universitario UAEM Atlacomulco

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco
Departamento de Evaluación Profesional

Atlacomulco, Méx., a 8 de Octubre de 2018.

DR. EN C. EN I. ALLAN ANTONIO FLORES FUENTES
P R E S E N T E:

Por este conducto me permito comunicar a usted que la Comisión Revisora de Proyectos de Tesis acordó nombrarlo **REVISOR(A)** del Trabajo de **TESIS** titulado:

"SISTEMA DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN Y CORRECCIÓN DE POSTURAS EN EJERCICIOS REALIZADOS POR FISICOCULTURISTAS"

Que presenta(n) el (los) Pasante(s) de **ICO. JUAN MANUEL MARTÍNEZ JOSÉ,** al mismo tiempo solicito a usted de la manera más atenta, concluir su revisión sea tan amable en comunicar por escrito su **VOTO APROBATORIO** a este Departamento, en un lapso **no mayor a 10 días a partir de la recepción del trabajo.**

29/10/2018

A T E N T A M E N T E
PATRIA, CIENCIA Y TRABAJO
"2018, Año del 190 Aniversario de la Universidad Autónoma del Estado de México"

M. EN P.C. NEPHTALÍ PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADÉMICO
CENTRO UNIVERSITARIO UAEM ATLACOMULCO

c.c.p. expediente del (la) interesado(a)

Km. 60 Carretera Toluca – Atlacomulco
C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco

Atlacomulco, México a 15 de octubre de 2018

M. EN P.C. NEPHTALI PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADEMICO DEL CENTRO
UNIVERSITARIO UAEM ATLACOMULCO

P R E S E N T E:

En contestación a su atento oficio con fecha del 05 de octubre de 2018, en el cual se me designó como **REVISOR** del Trabajo de **TESIS** titulado **"Sistema de visión artificial para la detección y corrección de posturas en ejercicios realizados por fisicoculturistas"**, presentado por el pasante de la Licenciatura en Ingeniería en Computación **Juan Manuel Martínez José,** informo que:

He concluido satisfactoriamente la revisión del trabajo, haciendo las observaciones pertinentes, mismas que han sido solventadas por el interesado. Así mismo, **manifiesto que el trabajo cumple con los requisitos y cualidades que corresponden a la opción de evaluación profesional**; por lo cual no tengo ningún inconveniente en dar mi **Voto Aprobatorio** para que el egresado pueda continuar con los trámites de titulación.

Sin otro particular por el momento, agradezco su atención.

A T E N T A M E N T E

DR. EN C. I. E. ALLAN ANTONIO FLORES FUENTES
PROFESOR DE TIEMPO COMPLETO

Km. 60 Carretera Toluca – Atlacomulco, C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx
Centro Universitario UAEM Atlacomulco

Universidad Autónoma del Estado de México
Centro Universitario UAEM Atlacomulco
Departamento de Evaluación Profesional

Atlacomulco, Méx., a 22 de Octubre de 2018.

P. ICO. JUAN MANUEL MARTÍNEZ JOSÉ
P R E S E N T E.

Por este conducto me permito informar a usted(es) que habiendo concluido el Trabajo de **TESIS** denominado:

"SISTEMA DE VISIÓN ARTIFICIAL PARA LA DETECCIÓN Y CORRECIÓN DE POSTURAS EN EJERCICIOS REALIZADOS POR FISICOCULTURISTAS"

Se autoriza la impresión y/o digitalización de dicho trabajo a fin de que continúe(n) con los trámites para obtener el Título de **INGENIERO EN COMPUTACIÓN.**

A T E N T A M E N T E
PATRIA, CIENCIA Y TRABAJO
"2018, Año del 190 Aniversario de la Universidad Autónoma del Estado de México"

M. EN P.C. NEPHTALÍ PIERRE ROMERO NAVARRETE
SUBDIRECTOR ACADÉMICO
CENTRO UNIVERSITARIO UAEM ATLACOMULCO

c.c.p. expediente del (la) interesado(a)

Km. 60 Carretera Toluca – Atlacomulco
C.P. 50450
Atlacomulco, Estado de México
Tels. (712) 122 04 36, 122 04 46, 122 0535
e-mail: cuatla@uaemex.mx

DEDICATORIAS

A mis Padres, por ser mi soporte y sostén durante mi carrera; por sus consejos, lecciones, amor y comprensión a lo largo de 5 años de estudio en la Universidad, y durante el tiempo que tomó la realización de este proyecto de tesis. Gracias por ser mi inspiración, por secar mis lágrimas cuando sentía que no podía más y cuando finalmente lo logré. Los amo mucho.

A mis hermanos, por su apoyo y cariño; sin ustedes nada de esto podría haber sido posible. A mi niña Camila por darme las fuerzas y la felicidad que necesitaba cuando me encontraba frustrado.

A todos mis ex compañeros de generación, ICO 17, por su soporte a lo largo de esta travesía que compartimos juntos; por brindarme la oportunidad de representarlos, fue un honor hacerlo.

A toda mi familia en general y personas que contribuyeron de forma directa o indirecta en mi formación Universitaria y en mi vida personal en este lapso de tiempo; igualmente para aquellos que no creyeron en mí.

AGRADECIMIENTOS

Al Centro Universitario UAEM Atlacomulco por facilitar sus instalaciones y laboratorios para la realización de este proyecto. Por haberme otorgado una formación académica integral y a cada uno de los profesores que contribuyeron en mi educación universitaria.

A mi asesor de tesis, Dr. Everardo Efrén Granda Gutiérrez, por su apoyo incondicional, por su paciencia a lo largo de más de un año de trabajo, y por sus oportunidades. Por estar al pendiente de mí en cada duda y cada problema que surgía.

Al maestro Guillermo Peruyero Argueta por prestar el espacio del CAA para realizar pruebas de entorno.

Agradecimientos especiales al dueño, administradora y representantes del gimnasio "TOTAL GYM" de la ciudad de Atlacomulco, Estado de México, por prestar sus instalaciones y las facilidades ofrecidas durante la realización de pruebas preliminares y finales.

Agradecimientos al Instructor certificado por la Federación Mexicana de Fisicoconstructivismo y Fitness A.C., Daniel Lovera Martínez, por su generosidad y tiempo para la captura de movimiento de los ejercicios que contiene el sistema, además por su disponibilidad para realizar pruebas finales y brindar su valoración final del Software "MuscleKIN".

RESUMEN

Hasta la fecha de elaboración de este documento no se ha identificado alguna herramienta tecnológica que permita definir y garantizar que el fisicoculturista realice de manera correcta los ejercicios de su plan de entrenamiento. Un problema general es que cada gimnasio cuenta con sólo un instructor por turno de trabajo; esto implica que el experto no se encuentre al pendiente de la ejecución en cada individuo, dando como resultado lesiones, sobre entrenamiento o carencia de estímulo muscular, efectos contrarios al esperado por dicha actividad.

Actualmente hay cierto interés por imitar, de manera artificial, las capacidades y sentidos naturales de los seres humanos. Entre algunos ejemplos se contemplan: la vista, con el reconocimiento de objetos o patrones, el seguimiento del cuerpo humano y la captura de movimiento (Mo Cap). Para ello se hace uso de herramientas computacionales, como los llamados *exergame*, que incitan el movimiento del usuario ante entornos virtuales, y la programación de algoritmos que intentan dar solución a estos problemas.

A partir de los dos puntos anteriores, se ha propuesto este proyecto como área de oportunidad en el ámbito tecnológico-computacional. El objetivo de dicho proyecto de tesis es desarrollar un sistema de Visión Artificial interactivo, que permita reconocer y seguir los movimientos de los fisicoculturistas para determinar si sus posturas son correctas en tiempo real, mediante la posición de sus articulaciones en el sistema de ejes de tres dimensiones. Se contempla la creación de un algoritmo que permite una comparativa de la posición que presenta el usuario y las preestablecidas en el sistema, haciendo uso del dispositivo de captura de movimiento Kinect. El resultado es una herramienta computacional que apoya a los fisicoculturistas a estimular de forma correcta los músculos involucrados en cada ejercicio, evitando las posibles consecuencias negativas a corto o largo plazo que puedan surgir de una mala práctica.

Palabras clave: Visión Artificial; Captura de Movimiento; Exergame; Kinect; Fisicoculturismo.

ABSTRACT

At the day of preparation of this document, does not was identified a tool that allows defining and guaranteeing the bodybuilder performs the exercises of his training plan correctly. A general problem is that every gym has an instructor per work shift; this implies that the expert is not aware of the performance of each person, resulting in injuries, overtraining or lack of muscle stimulation, contrary to the expected effects.

Nowadays, there is some interest in artificially imitating the natural abilities and senses of human beings such as sight with the recognition of objects or patterns, the tracking of the human body, the capture of movement (Mo Cap), etc., by means of computational tools such as the so-called exergame that promote the movement of the user, before virtual environments, and the programming of algorithms that try to solve these problems, as well as the contribution that can suppose to the functioning of the current systems.

Mentioned the two previous points this point was detected as an area of opportunity in the technological-computational field. The objective of the following thesis project is to combine these aspects with the development of an interactive Artificial Vision System that follows the movements of the bodybuilders to determine if their positions and their body´s degrees of flexion are correct in real time, through the position of their articulations in the system of axes of 3 dimensions, with the creation of an algorithm that allows a comparison of the position presented by the user and the pre-established in the system, making use of the Kinect motion capture device. Resulting in the creation of a computational tool that supports bodybuilders to correctly stimulate the muscles involved in each exercise, avoiding the possible negative consequences in the short or long term that may arise from malpractice.

Keywords: Artificial vision; Motion Capture; Exergame; Kinect; Bodybuilding.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

El estudio de la actividad física deportiva está ligado a varias disciplinas y áreas de la ciencia que describen, desde su perspectiva, lo que el entrenamiento físico puede contribuir, sea negativa o positivamente en el potencial desarrollo del ser humano; por ejemplo, en el área de la salud y de la psicología. También existen aquellas áreas que ayudan cuando se pretende generar un control, un mejor aprovechamiento, así como el mejoramiento en la realización de dicha actividad; ejemplo de éstas es la computación.

A lo largo de toda su historia el hombre ha tenido un gran interés por hacer, de ciertas tareas que podrían resultar complicadas o laboriosas, algo más sencillo, mediante mecanismos de diferentes grados de complejidad, que pudieran ayudar en la realización de éstas, o que puedan suplir a otras; por ejemplo: las funciones y la forma en que trabaja cada componente del cuerpo humano.

Un ejemplo particularmente interesante se presenta en aquellos individuos que carecen de alguna extremidad, como pueden ser los brazos, manos o piernas. Con el fin de suplir el órgano faltante se han utilizado exoesqueletos robóticos, también llamados servoarmaduras, exomarcos o exotrajes. El término en el ámbito biológico se refiere a los esqueletos externos con los que cuentan los insectos o crustáceos, que les permiten realizar actividades complejas como el cargar peso, como medida de defensa ante depredadores o para su movilidad. Por extensión, en el área de la robótica los exoesqueletos son mecanismos estructurales consistentes en armazones externos, cuyos segmentos o articulaciones se adaptan o acoplan al cuerpo humano, y tienen la función de proteger, recubrir y dar soporte al portador, mediante un sistema de potencia de motores o elementos hidráulicos que proporcionan parte de la energía para el movimiento de las extremidades que se están suplantando o apoyando. Éstos pueden ser útiles para el movimiento del portador y para realizar actividades determinadas, en aplicaciones en el ámbito de la salud o militar (Ruíz Olaya, 2008).

De manera similar, se ha tratado de imitar el funcionamiento de órganos internos, o incluso sentidos completos. En este último caso ha habido un gran desarrollo e innovación para tratar de suplir artificialmente el uso de los sentidos humanos, mediante sistemas basados en la tecnología actual. Algunos de los sentidos, capacidades o

características humanas que se pretenden emular artificialmente con la ayuda de la tecnología son las relacionadas con la audición, la inteligencia o el procesamiento de información del cerebro humano, el tacto y la visión.

La visión es uno de los sentidos que más tienen importancia para el ser humano; por ello es uno de los que más han sido investigados y desarrollados. Los humanos contamos con todo un sistema visual, que permite detectar estructuras, figuras, colores, tonalidades, profundidad o distancia, luminosidad y lo que se encuentre en nuestro entorno (gracias a la visión periférica); además, cuenta con la capacidad de seguimiento de objetos, el reconocimiento de rostros (humanos, de otros animales o incluso plantas); puede a su vez adaptarse a los cambios de iluminación, cambios externos en el ambiente y problemas de enfoque, sin dificultad alguna y rápidamente. Los investigadores y tecnólogos de la visión artificial han conseguido, mediante diversas técnicas, imitar algunas de estas funciones con apoyo de algoritmos computacionales y haciendo uso de distintos dispositivos, tales como: cámaras, lentes, espejos, sistemas de iluminación, sensores ópticos, entre otros (Shapiro & Stockman, 2000).

En esencia, un sistema de Visión Artificial trata de conseguir que las imágenes que captan los dispositivos puedan ser correctamente procesadas para su interpretación, y que la computadora sea capaz de hacer tareas con dicha información (Shapiro & Stockman, 2000). Para el desarrollo de estos sistemas, desde el enfoque computacional, se hace uso de Hardware y Software: las cámaras son capaces de capturar una imagen, pero sin una correcta interpretación, esta es prácticamente inservible.

Es importante mencionar que, a pesar de todos los avances que se han conseguido, aún no se ha logrado hacer un sistema que pueda imitar en su totalidad al sentido de visión y su capacidad de seguimiento, puesto que estos llegan a tener cierto grado de error, dependiendo de los algoritmos que se utilizan, así como las características del dispositivo del que se hace uso. Además, los sistemas artificiales sólo se enfocan en una parte de lo que la visión puede hacer, y no en la totalidad de las capacidades de ésta.

El incremento en investigaciones para el seguimiento del cuerpo humano con sistemas de visión ha sido de interés para diversas empresas, que usan los gestos de las manos, cara, o el cuerpo entero para la seguridad informática, o hacer de un programa algo más

inmersivo, es decir, que el usuario se sienta perteneciente e introducido al sistema, y si así sea el caso, sumergirlo al ambiente virtual que posea interactuando de forma directa, de este modo prescindiendo del uso de mandos o de intervención humana mediante el tacto. El desarrollo de recientes dispositivos, como las cámaras de profundidad en 3D, permite que el seguimiento sea más sencillo para aplicaciones reales. El sensor Kinect, de la compañía Microsoft, es un dispositivo de este tipo que fue concebido en un inicio para su uso en videojuegos para la consola XBOX 360, pero que ha abierto una gran brecha para el desarrollo de sistemas o programas multimedia, especialmente por su accesibilidad y bajo costo, además de su alta efectividad en cuanto a aplicaciones se refiere, como el reconocimiento y seguimiento del cuerpo humano, en sistemas de visión artificial, en el ámbito de la electrónica, robótica y otros campos de la ingeniería.

En este trabajo, se busca la implementación de una interfaz de software que emplea un sistema de visión artificial para la identificación, procesamiento y apoyo en la corrección de posturas durante la realización de ejercicios físicos en personas practicantes del fisicoculturismo en todos sus niveles, pero enfocado primordialmente a los principiantes o aficionados. Este sector fue seleccionado principalmente por que se ha observado que usualmente carecen de conocimientos en el ámbito, y que requieren de un apoyo externo, un instructor, o la investigación empírica para llevar a cabo la ejecución de los ejercicios, lo que, en muchos casos, puede no ser la mejor opción para el individuo.

El sistema propuesto hace uso del dispositivo Kinect, que cuenta con una cámara, un sensor de profundidad, además de un procesador personalizado, que proporciona la captura de movimiento de todo el cuerpo en 3D, con la cual se realiza una comparación de la forma correcta de ejecución del movimiento durante el ejercicio, y la forma que presenta el individuo, de manera que ayudará a corregir la postura, para mejorar la ejecución.

Con base en la literatura consultada se sustenta lo que se pretende realizar con el presente proyecto de Tesis, el cual busca la implementación de tecnología de captura de movimiento, el desarrollo de algoritmos de comparación de posturas mediante la posición de las articulaciones y la programación de una interfaz interactiva mediante los gestos, creando así un apoyo computacional para en el área del fisicoculturismo.

2 PLANTEAMIENTO DEL PROBLEMA

En la práctica de la actividad física existe el riesgo de ejecutar de manera incorrecta los ejercicios del programa de entrenamiento que se sigue; de esta forma no se aprovecha de una manera eficaz el trabajo y, por lo tanto, el efecto que tiene sobre el cuerpo humano dicho ejercicio es nulo o, incluso, perjudicial.

Como una contribución de la Ingeniería en Computación para la solución de este problema, se busca la implementación y desarrollo de una interfaz de software a través de un sistema de visión artificial, que permita la identificación, procesamiento y apoyo en la corrección de las posturas en ejercicios físicos, para personas practicantes del fisicoculturismo en sus diversos niveles, enfocado primordialmente a nivel principiante.

2.1 Definición del problema

Existen diversos niveles de ejercitación en el fisicoculturismo: para aficionados, de rendimiento, de elite y profesional. Es en el primero de éstos donde surgen diversos problemas, debido a la falta de conocimiento en dichas áreas o la falta de atención del personal capacitado en el gimnasio. Los problemas pueden ir desde un sobre entrenamiento, es decir hacer más ejercicio del que el cuerpo requiere en una sola sesión, perjudicando el desarrollo del mismo, o falta de entrenamiento, que por el contrario es la carencia de estimulación muscular necesaria para el desarrollo; también es posible que se ocasione una lesión debida a alguna mala postura en el ejercicio realizado (N. Platanov & M. Bulatova, 2001).

Por ello se propone el uso de un sistema tecnológico que apoye a la correcta realización del ejercicio físico en cuestión, complementando, o incluso sustituyendo parcialmente, al entrenador físico pendiente de la ejecución, y coadyuvando a evitar los problemas mencionados a corto o a largo plazo.

2.2 Objetivos de investigación

Objetivo general:

Implementar una interfaz de software, a través de un sistema de visión artificial, para realizar la identificación y procesamiento de posturas, ayudando a la corrección de ejercicios físicos en personas practicantes del fisicoculturismo, mediante la utilización del dispositivo Kinect, permitiendo realizar una comparación de la forma correcta del movimiento en determinado ejercicio y la forma que presenta el individuo y proporcionando una retroalimentación de su realización mediante indicativos visuales.

Objetivos específicos:

- Estudiar el funcionamiento del sensor Kinect para usarlo en la captura de la escena y seguimiento del esqueleto humano.
- Evaluar la precisión y problemas de visión del sensor ante diversos entornos, especialmente en un gimnasio de pesas convencional.
- Identificar, y de ser necesario personalizar, una biblioteca de software adecuada para la detección y seguimiento del cuerpo humano para Kinect con un entorno de programación adecuado.
- Seleccionar el protocolo de comunicación entre el entorno de desarrollo y el sensor Kinect.
- Implementar la comunicación entre los entornos de desarrollo de software y el dispositivo.
- Diseñar un algoritmo que permita la comparación del movimiento del usuario con patrones de ejercicios previamente establecidos en tiempo real (o lo más cercano a este concepto que sea posible con un sistema de cómputo de oficina).
- Desarrollar una interfaz gráfica mediante la gesticulación para la interacción entre el usuario y el sistema.
- Implementar el algoritmo desarrollado en cinco ejercicios realizados en el área libre de un gimnasio para fisicoculturistas: sentadilla, curl de bíceps, levantamientos laterales, press militar y peso muerto.

- Evaluar los resultados del monitoreo del cuerpo humano durante la práctica del estímulo físico para la comparación de la postura presentada por el usuario y las preestablecidas en el sistema, bajo condiciones especificadas por un instructor de pesas certificado.

2.3 Preguntas de investigación

¿El sensor Kinect es conveniente para el desarrollo del sistema para la identificación y corrección de errores durante la realización de ejercicios?

¿Qué lenguaje de desarrollo se debe emplear para la programación del sistema de visión artificial para esta aplicación en específico?

¿De qué manera o mediante qué datos se podrá hacer la comparación de las posturas realizadas por el usuario y las predefinidas de una manera correcta?

¿Qué factores externos (ambientales) afectarán la visión del dispositivo?

¿Qué herramientas matemáticas se harán uso para el procesamiento de la imagen y el seguimiento del cuerpo humano durante la realización de ejercicios de fisicoculturismo?

¿Cuáles son las especificaciones de diseño de una interfaz adecuada para el uso de la herramienta en un entorno similar a un gimnasio?

2.4 Justificación

El tema de investigación propuesto surge de la necesidad de innovar en el ámbito tecnológico y computacional, teniendo aplicación en el ámbito de la salud y el fisicoculturismo, mediante la creación de una herramienta para la correcta práctica de este último.

Según la Organización Mundial de la Salud (OMS), el 60% de la población mundial no realiza actividad física. Haciendo énfasis en que la inactividad física es el cuarto factor de riesgo en lo que respecta a la mortalidad mundial lo cual corresponde el 6% de las muertes registradas en todo el mundo, además de ser causante de diversas enfermedades

como cáncer, diabetes, y problemas cardiacos. En julio del 2013, la Organización de las Naciones Unidas emitió un reporte que colocó a México como el país con mayor obesidad en el mundo (OMS, 2017).

Sin embargo, según el estudio realizado por el Módulo de Práctica Deportiva y Ejercicio Físico del INEGI en el año 2016, la práctica de la actividad física en la actualidad va en ascenso, con un 44% de la población mexicana que son activos físicamente en comparación a las cifras registradas en 2014 realizadas por el mismo módulo, que presentó un 43.6%. La buena costumbre del mantener el cuerpo en buenas condiciones algo que preocupa a la sociedad de la actualidad, por estética o cuestiones de salud son las principales razones (INEGI, 2014), (INEGI, 2016).

Muchas de las personas que inician alguna actividad física, lo realizan sin contar con conocimientos previos, una certificación por una institución avalada en el ámbito, o en su caso no cuentan con un entrenador personalizado preparado, por lo que suelen ocurrir eventos contraproducentes a corto o a largo plazo como lesiones o que el ejercicio no genere los resultados deseados, causando desmotivación y una pérdida del interés por la continuidad de la ejercitación.

Este proyecto pretende servir como incentivo indirecto para fomentar la actividad física, específicamente en el fisicoculturismo, que es una disciplina deportiva completa y que está ganando adeptos, sobre todo por la cuestión estética, pero no menos importante por su impacto en la salud. Es por ello que, con el apoyo de un sistema de visión artificial que permita corregir de manera adecuada la ejecución de los ejercicios realizados por las personas más inexpertas, intermedias hasta avanzadas, se puedan reducir las consecuencias que puede haber cuando el personal encargado no esté capacitado, o en su defecto no se encuentre presente; evitando así, lesiones que puedan surgir de la mala estructura, a corto o a largo plazo. Además, generando así una ejecución más funcional, aprovechando el potencial que tiene dicho ejercicio sobre los músculos en los cuales se enfoca.

Todo esto considerando condiciones específicas del entorno de un gimnasio y de los usuarios (Iluminación, espacio, limitaciones, obstáculos, distancia, ropa, estatura y

complexión), así como la calibración específica con respecto a las características del dispositivo.

El proyecto es, por tanto, un ejemplo de la aplicación real de conocimientos del área de Ingeniería en Computación, que son empleados para la solución de un problema en otras disciplinas. Tal es el perfil de Egreso de la carrera de Ingeniería en computación, como se establece en su plan de Estudios, donde se enuncia que "[…] el egresado de la Licenciatura en Ingeniería en Computación […], es un profesional que será capaz de realizar, poseer, desarrollar, administrar, proporcionar y realizar el análisis, el diseño, la implementación para crear tecnología de los sistemas computacionales y dar la solución a los problemas propios y de otras disciplinas, mediante el uso de herramientas computacionales para poder adaptarse al entorno y a la sociedad" (UAEMEX, 2018).

2.5 Impactos

El impacto que tiene el presente trabajo de investigación es de ámbito tecnológico, sustentado en el desarrollo de un software, así como de algoritmos para la detección y corrección de posturas en ejercicios realizados por fisicoculturistas en el área libre de un gimnasio mediante el uso del dispositivo Kinect para la captura de movimiento.

En el ámbito social, se brindará a la comunidad de personas de los diversos niveles en esta área deportiva una herramienta tecnológica sencilla de usar para el correcto aprovechamiento de la actividad física, y evitar así, sus posibles consecuencias negativas en la salud. Potencialmente, esto también podría redundar en ahorro monetario para los individuos practicantes del fisicoculturismo, al reducir el costo en instructores personales, y en médicos al evitar lesiones.

3 META DE INGENIERÍA

El desarrollo de un sistema de visión artificial consistente en la integración de un sensor tipo exergame “Kinect”, algoritmos de detección de movimiento, de comparación de posturas y una interfaz de usuario específicamente diseñada, permitirá la detección de los movimientos del cuerpo humano para ayudar a la corrección de posturas en cinco ejercicios realizados por fisicoculturistas durante un entrenamiento típico en el área libre de un gimnasio, siendo éstos: levantamientos laterales, press militar, peso muerto, curl de bíceps, y sentadilla, a partir de la posición en tiempo real de las articulaciones con respecto a los grados de flexión del cuerpo detectados del usuario, dentro de un rango de tolerancia de 5° de error, en comparación a los grados de flexión pre establecidos en el sistema, bajo el régimen de un instructor certificado para la ejecución correcta de cada ejercicio.

4 ESTADO DEL ARTE

Para comprender cómo se debe concebir un sistema de visión artificial es recomendable familiarizarse con el sentido de la vista; por ello es necesario reconocer cómo funciona dicho sentido y sus características. También deben identificarse los conceptos importantes para comprender de manera clara y concisa el presente trabajo de investigación y los procesos con los que éste se involucra. Lo anterior se detalla en la presente sección.

4.1 Culturismo

El culturismo, también conocido como Fisioculturismo, Fisicoconstructivismo o Fisicoculturismo, es una actividad basada en ejercicios físicos intensos, generalmente ejercicios anaeróbicos. La mayoría de las veces el entrenamiento hace uso de pesas en gimnasios, mediante diversos tipos de ejercicios de fuerza/hipertrofia muscular. Esta actividad tiene como finalidad la obtención de musculación fuerte y/o definida. También se suele llamar musculación a la actividad encaminada a hipertrofiar el músculo, ocasionando un crecimiento en el tamaño de las células musculares, lo cual supone un aumento de tamaño de las fibras y por lo tanto del músculo (país, 2017).

La práctica del culturismo es uno de los principales objetivos por los que la gente acude a los gimnasios. No es necesaria la práctica de este a nivel profesional o de competición para ser llamado culturista o fisicoculturista, se le acuñe este apelativo a toda persona que hace uso de la actividad física en cualquiera de sus distintas ramas para cambiar su aspecto físico o mejorar la salud. Pero, teniendo en cuenta la tendencia actual de los pesistas la atención se enfoca únicamente en el mejoramiento de la preparación física.

Hay cuatro niveles de práctica del culturismo: iniciación, medio, superior y de competición. El culturismo es pensado para que haya un desarrollo y progresión constante e ininterrumpida en el cuidado de la salud y la forma física en oposición al mantenimiento de un nivel establecido. Cada entrenamiento debe ser pensado y programado aumentado la dificultad de la anterior, por ello se recomienda mantener un registro de los progresos (Pearl & Moran, 1990).

La preparación técnica se efectúa en mayor cantidad en los primeros años de entrenamiento, y a medida que pasan el tiempo, se mejoran las posturas y la realización de los ejercicios, a su vez el cuerpo se acostumbra a la demanda física y, por ende, se debe incrementar la dificultad de los entrenamientos.

Se debe tomar en consideración el proceso que conlleva la práctica de la preparación física, que consiste en lo siguiente (Cherebetiu, et al., 1988):

- La caracterización del esfuerzo específico. Se debe tomar en consideración las cualidades con las que se cuenta. Pues son aquellos puntos que se deben desarrollar en el entrenamiento de levantamiento de pesas.
- Dinámica del esfuerzo. Se refiere a la intensidad y el volumen que se usan para los entrenamientos, se hace siempre un hincapié que para el desarrollo y progresión muscular es necesario un aumento de la intensidad del esfuerzo, a base del crecimiento de la carga (cantidad de peso que se es capaz de levantar).
- Factores del entrenamiento. Se consideran varios puntos importantes en la realización de la actividad, estos factores deben ser atacados de manera simultánea, pueden o no variar acorde a los objetivos, acorde al tipo de deportista en relación con las particularidades del individuo, pero los primeros de ellos deben ser puntuales y precisos:
    a) Preparación física
    b) Preparación técnica
    c) Preparación táctica
    d) Preparación teórica
    e) Preparación psicológica
- Métodos de entrenamiento. Son aquellas maneras de entrenamiento que se usan para llegar a cierto objetivo; como el método del pesista, culturista, power-training, entrenamiento en circuito, entrenamiento piramidal, contracciones isométricas, etc.
- Medios de entrenamiento.
    a) Medios específicos (Figura 4.1): Son todos aquellos ejercicios enfocados a un solo musculo. Ejemplos: Especializados en el pecho; press de banca, press de banca inclinado, press al marco isométrico, etc.

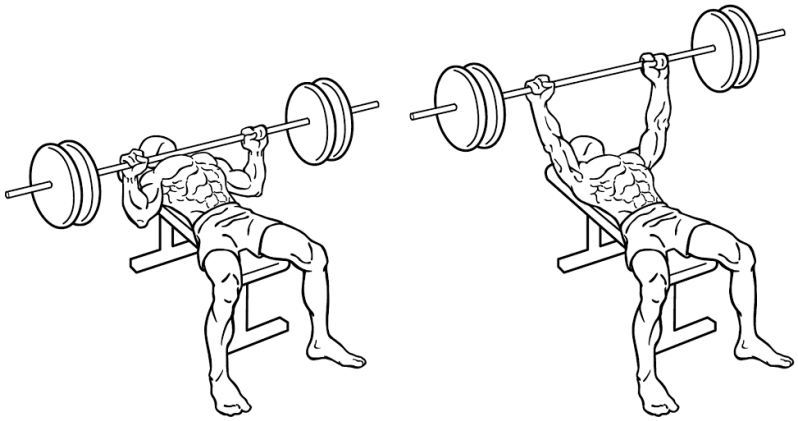

Figura 4.1 Imagen ilustrativa ejercicios de tipo medio específico (Musculación para principiantes, 2017).

b) Medios no específicos (Figura 4.2): Ejercicios que involucran varios músculos o que son usados por otros deportes, ejemplos: sprint, saltos, lanzamientos, etc.

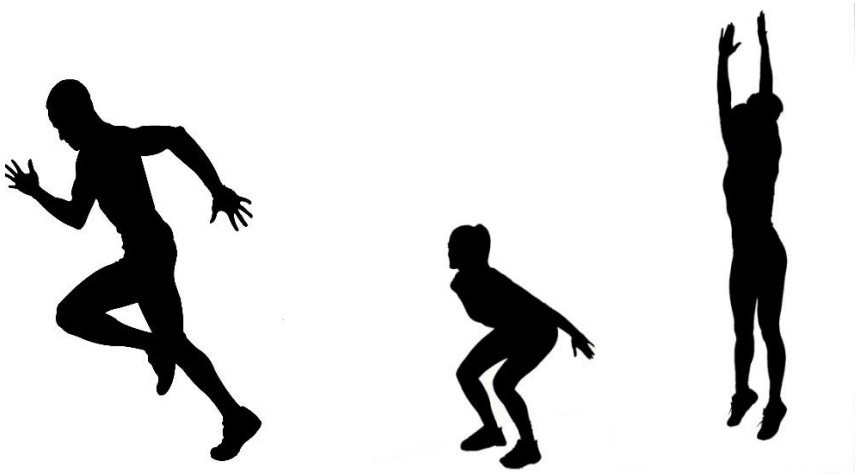

Figura 4.2 Imagen ilustrativa ejercicios de tipo medio no específico (Hernández, 2014).

- Prueba de control. Son todas aquellas pruebas que indican el rendimiento actual del individuo, se debe llevar un registro del progreso en éstas, y siempre debe ir en aumento para la progresión muscular; caso contrario se debe hacer una reestructuración del entrenamiento.

En el libro “Enciclopedia general del ejercicio: Musculación”, se proponen 12 consejos dirigidos a los culturistas para lograr correctamente el mejoramiento de la salud y del aspecto físico; se citan los siete primeros puntos debido a que son estrictamente necesarios, si no se toman en cuenta la gran mayoría de ellas, prácticamente cualquier ejercicio o rutina que se practique quedará obsoleto (Pearl & Moran, 1990):

1. Hacer un reconocimiento médico.
2. Ser realista.
3. Hacer una preparación cardiovascular.
4. Añadir variedad.
5. Empezar con suavidad.
6. Descansar cuando sea necesario.
7. Trabajar los puntos débiles.

Con el apoyo de la documentación mencionada con anterioridad y del Fisicoculturista de competición Amateur e Instructor a nivel profesional avanzado, Daniel Lovera Martínez avalado por la Federación Mexicana de Fisicoconstructivismo y Fitness A.C., se entablaron las condiciones específicas para la ejecución de ejercicios anaeróbicos, y se llevó a cabo la captura de movimiento; se tomaron en consideración 5 ejercicios en el área libre de un gimnasio de pesas (ejercicios a base de mancuernas o barras olímpicas), siendo estas:

1. Curl de bíceps con mancuerna de pie.
2. Sentadilla normal con peso corporal.
3. Press militar de hombros con mancuerna.
4. Levantamientos laterales con mancuerna para hombro.
5. Peso muerto.

A continuación, se enuncia las posturas e indicaciones precisas para la correcta ejecución de cada ejercicio que se incluye en el sistema “MuscleKIN”. Así como como la ilustración de la correcta postura [Todas las imágenes obtenidas de: (Musculación para principiantes, 2017)]:

4.1.1 Curl de Bíceps con mancuerna de pie

Es uno de los ejercicios más recomendables en el entrenamiento de brazos; como su nombre lo indica, está enfocado en el desarrollo del bíceps de un modo concentrado, es decir, sin hacer partícipes a otros músculos. Este ejercicio se puede llevar a cabo con tensores, ligas o mancuernas, tal como se aprecia en la Figura 4.3.

Figura 4.3 Ejecución Curl de Bíceps.

Las recomendaciones para efecutar este ejercicio son las siguientes:

- Se deben aguantar las pesas durante el tiempo o repeticiones que indica el plan de entrenamiento.
- Mantenerse erguido con los pies separados por unos 40 cm.
- Mantener la espalda, la cabeza alta, las piernas y las caderas tiesas.
- Empezar con las pesas a la altura de los hombros con las palmas hacia adentro, a ambos lados de los muslos.
- Hacer curl de pesa con la mano derecha y la palma hacia adentro hasta pasado el muslo, después girar la palma hacia arriba para el resto del curl hasta el hombro.
- Mantener la palma hacia arriba mientras se baja hasta más allá del muslo, luego girar la palma hacia adentro.
- Mantener la parte superior del brazo pegada al costado.
- Hacer una repetición con el brazo derecho, después hacer el curl con el izquierdo.
- Inspirar hacia arriba, espirar hacia abajo.
- Puede realizarse sentado en banco recto, en banco inclinado, o con ambas manos.

4.1.2 Sentadilla normal con peso corporal, mancuernas o barra olímpica

Ejercicio compuesto (influye en más de un músculo) enfocado al desarrollo muscular, primordialmente en las piernas, y dependiendo de la posición en los que se sitúen los pies, el trabajo se enfocará más en las distintas zonas del cuádriceps o femoral, además de los glúteos, requiriendo fuerza adicional del abdomen (Figura 4.4).

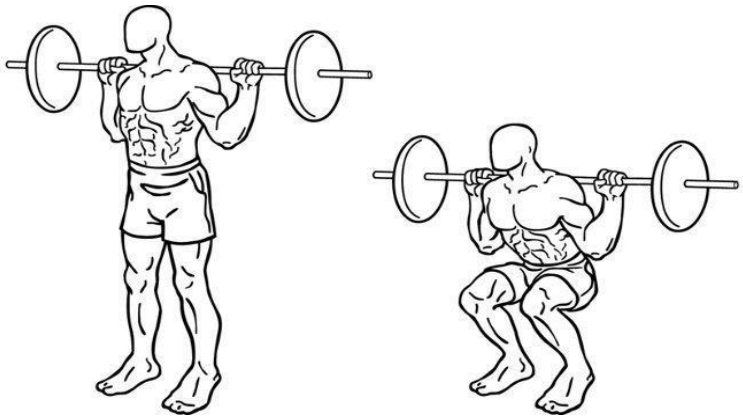

Figura 4.4 Ejecución Sentadilla.

A continuación, se describe como realizar de manera adecuada la sentadilla en su modo convencional:

- Colocar la barra en la parte posterior de la espalda o en su caso con el peso corporal se debe omitir este paso.
- Agarrar cómodamente con las manos.
- Mantener la cabeza levantada, la espalda recta y los pies separados unos 75 cm o a nivel de los hombros.
- Sentadillas hasta que la parte superior de los muslos quede paralela al suelo.
- Continuar con la cabeza levantada, la espalda recta y las rodillas separadas.
- Volver a la posición de inicio.
- Inspirar hacia abajo, espirar hacia arriba.

4.1.3 Press militar de hombros con mancuerna

Es unos de los ejercicios más conocidos para el desarrollo muscular de los hombros, ejercicio compuesto que requiere además trabajo adicional del tríceps, puede realizarse con mancuernas de pie o sentado; en el caso particular del sistema únicamente admite la realización a pie de este (Figura 4.5).

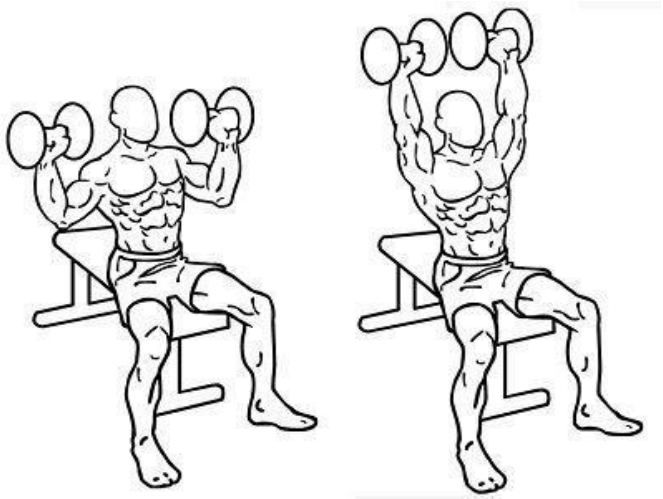

Figura 4.5 Ejecución Press Militar.

A continuación, se describe la realización correcta:

- Levantar las pesas a la altura de los hombros.
- Mantener las piernas y las caderas tiesas.
- Colocar los codos hacia afuera, los pulgares hacia adentro.
- Press de pesas hacia arriba en toda la longitud del brazo.
- Bajar las pesas hasta la posición inicial.
- Inspirar al inicio del press, espirar al final.
- También puede hacerse con las palmas hacia adentro.
- También puede hacerse sentado, con las palmas hacia adentro o hacia afuera.

4.1.4 Levantamientos laterales con mancuerna

Ejercicio esencial enfocado al desarrollo muscular de la parte trasera de los hombros, en uno de los lugares más débiles del músculo: el deltoides; aísla el trabajo en dicha área y protege ciertas partes involucradas como el manguito rotador de la mano (Figura 4.6).

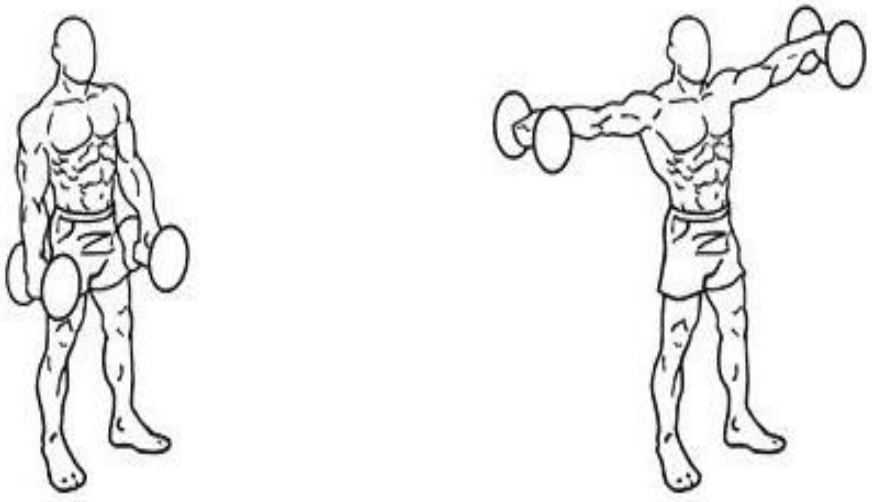

Figura 4.6 Ejecución Levantamientos Laterales.

Para efectuar este ejecicio debe observarse lo siguiente:

- Mantenerse erguido con los pies separados por unos 40 cm.
- Mantener la espalda, la cabeza alta, las piernas y las caderas tiesas.
- Agarrar las pesas, las palmas hacia adentro, los brazos rectos, hacia abajo en los costados.
- Levantar las pesas con un movimiento semicircular un poco más arriba que la línea de los hombros.
- Pausa, después de bajar a la posición del inicio utilizando el mismo camino.
- Mantener los brazos rectos.
- Inspirar hacia arriba, espirar hacia abajo.
- También puede hacerse sentado.

4.1.5 Peso muerto

Ejercicio compuesto en los cuales se involucran casi todos los músculos del cuerpo como lo son los de las piernas, dependiendo de la posición de los pies será el área que se trabajará: cuádriceps, femoral; espalda y brazos (Figura 4.7).

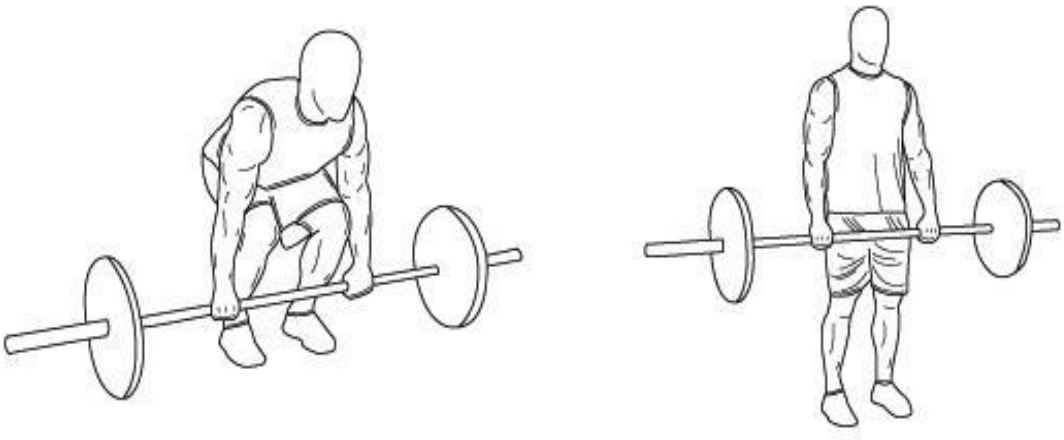

Figura 4.7 Ejecución Peso Muerto.

Para la correcta ejecución de esta rutina, las recomendaciones generales se describen a continuación:

- De pie, separar los pies unos 20 cm.
- Colocar una pesa en la parte exterior de cada pie.
- Inclinarse y agarrar las pesas.
- Mantener las piernas tiesas, la espalda recta, y la cabeza hacia arriba.
- Incorporarse con los codos tiesos.
- Bajar las pesas al suelo con las piernas tiesas.
- Inspirar hacia arriba, espirar hacia abajo.
- Puede realizarse también con barra.

4.2 Trabajos previos reportados en la literatura

Para la realización del presente proyecto de tesis se recaudaron trabajos y literatura en el que se enuncia de manera concreta una aplicación que realiza una función similar a la propuesta en este documento. Algunos ejemplos de trabajos similares atienden de manera puntual el problema de la captura de movimiento o señas, así como en la corrección de posturas, dando como resultado la obtención de información de sistemas que incluyen el movimiento de un robot a través de señas. Se han identificado también aplicaciones para el reconocimiento de objetos y figuras utilizando Kinect, sistemas de información de ejercicios, aplicaciones o videojuegos publicados por Microsoft o terceros desarrolladores que incitan al ejercicio físico. Además, se identificaron investigaciones de la eficacia del dispositivo Kinect en la rehabilitación de enfermedades. Sin embargo, el problema particular de una herramienta para atender la disciplina de fisicoculturismo no ha sido atendida aún.

La desarrolladora de videojuegos *Sumo Digital*, en conjunto con *Microsoft Studios* desarrollaron y comercializaron el software llamado "Nike + Kinect Training"; en la Figura 4.8 se muestra un ejemplo de su uso. De manera similar, *Ubisoft* con el título "Your Shape: Fitness Evolved" y *THQ* con "UFC: Personal Trainer", desarrollaron videojuegos compatibles con la consola de videojuegos XBOX 360 y su periférico Kinect. Estos incluyen diversas actividades, minijuegos y ejercicios principalmente aeróbicos que activan el cuerpo del jugador, incitando al usuario a la actividad física y por ende a la quema de calorías. El tipo de actividades que se pueden realizar van desde el Boot Camp, Cardio Boxing, saltos, hasta clases de baile, enfocando el trabajo físico en los diversos grupos musculares como son los brazos, abdominales, piernas, espalda, entre otros (Microsoft/Nike, 2012) (Microsoft/UFC, 2011) (Microsoft/Fitness Evolved, 2011) (Navarrete, 2010).

Es pertinente mencionar que los títulos mencionados fueron pensados con la finalidad de promover la movilidad y mejorar la salud del usuario, pero primordialmente están enfocados a videojugadores que requieran de un método accesible y llamativo para la pérdida de grasa corporal, recomendable para niños y adultos con problemas de sobrepeso y sedentarismo. En contraste, el sistema "MuscleKIN" incluye ejercicios

anaeróbicos y especializados, como los que se realizan por fisicoculturistas, cuyos mecanismos están enfocados a la ganancia de masa muscular, y no a la actividad física. Además, los juegos que se identificaron no tienen la capacidad de ser compatibles con accesorios adicionales como lo son mancuernas o barras.

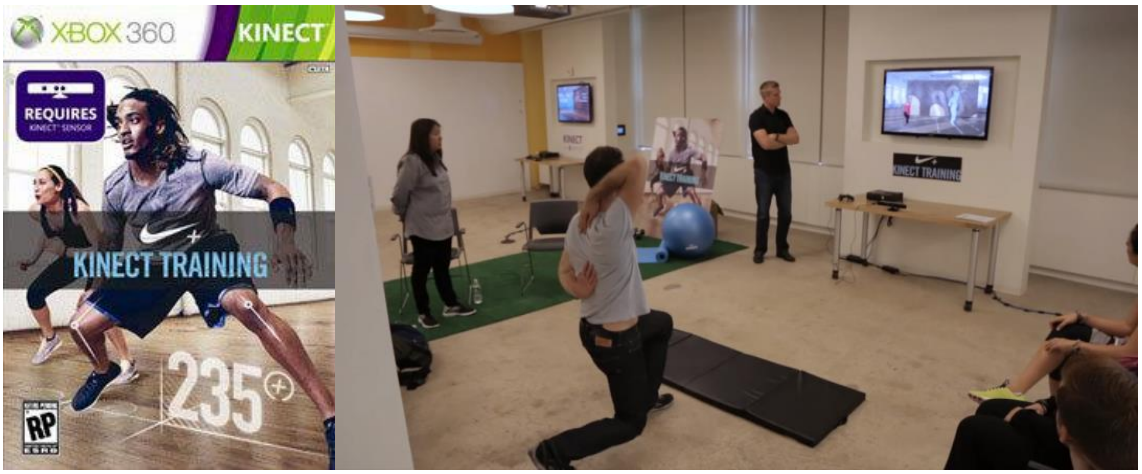

Figura 4.8 Videojuego: "Nike + Kinect Training" (Microsoft/Nike, 2012).

En un ámbito más especializado en el deporte se encontró que, en Egipto, la empresa *SmartSpot* bajo el liderazgo de Moawia Eldeeb, desarrolló una tecnología basada en el concepto de Kinect para los gimnasios (ver Figura 4.9), que brinda información acerca de los grados de flexión que presentan los fisicoculturistas al realizar más de 60 ejercicios diferentes. Esta herramienta tiene la posibilidad de imprimir dicha información para que las personas puedan analizar su desempeño y mejorar la próxima vez que realice actividad física. Cuenta con funciones adicionales como cronometraje del tiempo en que se realiza la actividad física, repetición de video, con detección de objetos, concretamente de la barra olímpica que se usa para la realización de algunos ejercicios como la sentadilla o el "peso muerto", generando únicamente información en la impresión final de las imágenes de cuantos grados de flexión realizaron en cada ejercicio, sin un indicativo en tiempo real que les permita conocer si realizaron de manera correcta los ejercicios (SmartSpot, 2017).

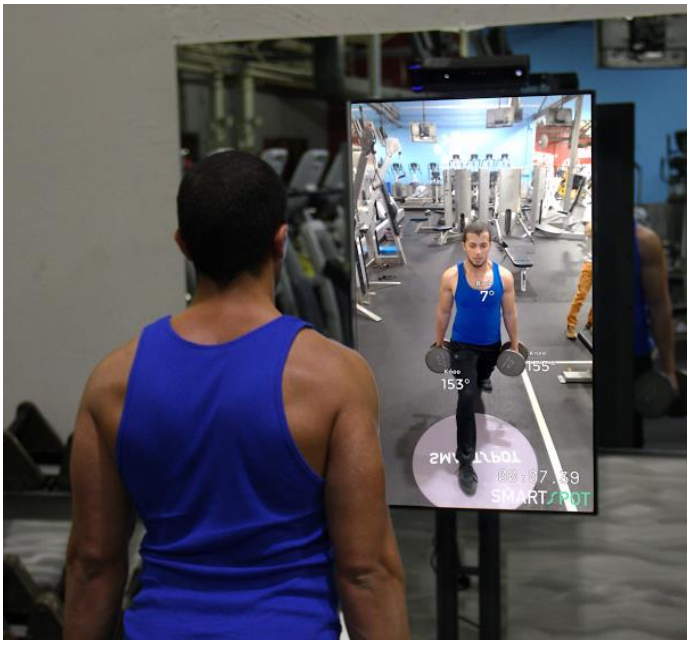

Figura 4.9 Sistema SmartSpot.

En un estudio realizado por Muñoz-John (Muños Cardona, et al., 2013) se hace empleo de un software en un ambiente de realidad virtual programada en conjunto con UNITY 3D; que permite la rehabilitación de pacientes con ciertas discapacidades motoras o de atención complementado con la tecnología de captura de movimiento Kinect un tipo de exergame y sensores para la captura de señales neurológicas, el cual permitirá que dichos pacientes ejecuten de manera correcta su tratamiento y desarrollen su capacidad mental de concentración. La interfaz es programada en el motor de desarrollo de videojuegos Unity 3D, el cual permitirá una interacción más intuitiva cerebro-máquina (ver Figura 4.10).

Los exergames en este caso estimulan el movimiento en los pacientes para que su rehabilitación sea menos abrumadora. El programa desarrollado en cuestión cuenta con tecnología de captura de movimiento, el cual permite una perfecta integración con el ordenador y el usuario arrojando buenos resultados en su rehabilitación. Combina las mediciones de balance y equilibrio junto a una interfaz mucho más amigable como los videojuegos, haciendo diagnósticos más asertivos.

Figura 4.10 Sistema de rehabilitación basado en Kinect (Muñoz, John).

En el trabajo realizado por Virginia González Rosquete (González Rosquete, 2013), se buscó el desarrollo de una aplicación que facilitara la estimulación física en personas con algún tipo de discapacidad, además, adaptándose a las necesidades y limitaciones que puedan tener los usuarios (ver Figura 4.11). Por lo cual se contempló el sistema desarrollado por Microsoft ya que brinda un buen reconocimiento e interacción con el usuario; así como crearlo de manera abierta para que ésta pueda ser adaptable acorde a las necesidades finales.

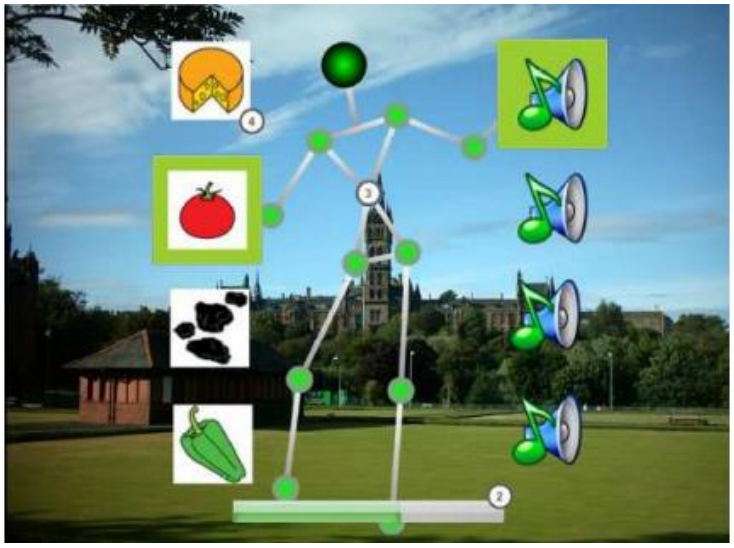

Figura 4.11 Sistema de rehabilitación basado en Kinect (Rosquete, Virginia).

En la Tabla 4.1 se muestra el estado del arte que influyó al proyecto de investigación y el área de oportunidad del que fueron ayuda en la programación del sistema. En conclusión, los sistemas de rehabilitación, así como las aplicaciones y videojuegos basados en el sensor Kinect encontrados en la búsqueda de literatura sirven como base para la formación del sistema “MuscleKIN”, debido a que en su totalidad hicieron uso de la misma tecnología y su potencial para la detección del esqueleto humano.

Tabla 4.1 Matriz de referencias.

Fuente	**Resumen**	**Metodología empleada**	**Áreas de oportunidad**
M. Martinez Zarzuela, et. al. **Artículo de investigación, 2011** **"Monitorización del cuerpo humano en 3D mediante tecnología Kinect"**	Investigación en la cual se detallan características, librerías para la monitorización y detección del cuerpo humano mediante Kinect.	Características y librerías para la utilización de Kinect en PC.	Ocupar toda información recabada en este trabajo de investigación para la correcta elección de entornos de desarrollo y librerías.
J. E. Muños Cardona, et. al. **Artículo de investigación, 2013** **"Sistema de Rehabilitación basado en el uso de análisis Biomecánico y Videojuegos mediante el sensor Kinect"**	Sistema de Software que se emplea a personas con discapacidad para realizar adecuadamente ejercicios de rehabilitación.	Sistema de rehabilitación mediante Kinect.	Bajo el mismo principio de la detección de movimiento mediante Kinect, emplear un sistema para la detección de movimiento de ejercicios en el gimnasio.
J. Edison Muñoz, et al. **Artículo de investigación, 2011** **"Exergames: Una herramienta tecnológica para la actividad física"**	Artículo de investigación acerca de los dispositivos también llamados exergames y su importancia para promover el ejercicio.	Reconocimiento y características de los dispositivos Exergames.	Recabar información acerca de los exergames, y su impacto en la actividad física.

L. Shapiro y G. Stockman **Libro** **”Computer Vision”**	Describe el concepto de visión artificial.	es aquel que intenta imitar la capacidad del ser humano de la visión, una manera de percepción de su ambiente.	Conocer el concepto de la V.A. y en área se encuentra el trabajo de investigación.
Microsoft **Página Web** **2017** **“Kinect Hardware”**	Características de Hardware del dispositivo Kinect.	Cuenta con una cámara RGB, una de profundidad infrarroja, y un proyector infrarrojo.	Conocer las características del sensor Kinect.
"Naity" **Artículo de Página Web, 2011** **“User Tracking with LabVIEW and Kinect based on the OpenNI Interface”**	Seguimiento del cuerpo humano mediante Kinect y la utilización de LabView basada en la librería de OpenNI.	Compatibilidad de Sensor Kinect y entorno de desarrollo LabView.	Investigar y emplear la librería más adecuada para el uso de Kinect y LabView.
S. Pfeiffer **Artículo de investigación, 2011** **“Guiado gestual de un robot humanoide”**	Control de un robot humanoide mediante la detección de movimiento obtenida de una cámara.	OpenNI es un Framework multiplataforma para escribir aplicaciones donde la interacción humana y un dispositivo basado en uno de los sentidos naturales (vista, oído) actúen entre sí.	Conocer el control y detección de movimiento mediante el sensor Kinect.

J. C. Quinche Curtidor **Artículo de investigación, 2011** **"Dispositivos de captura de movimiento (Kinect), para la navegación de experiencias formativas en ambientes virtuales 3D"**	Utilización de un sensor Kinect para el movimiento de una animación en un ambiente 3D.	El desarrollo de Sistema puede ser mediante la utilización de la suite del dispositivo SDK (Software Development Kit) liberado por su desarrollador, u OpenNI de desarrollo libre.	La detección del movimiento y la interpretación en una animación para la corrección de las posturas.
L. D. Sardi **Artículo de investigación, 2011** **"Algoritmo de segmentación Online de Imágenes en secuencias de video"**	Analiza las técnicas para la segmentación de imágenes.	Dificultades del dispositivo Kinect ante el entorno físico.	El procesamiento de imágenes obtenidas mediante cámaras de video.
V. González Rosquete **Artículo de investigación, 2011** **"Advant y Advanted: plataforma para el entrenamiento cognitivo y físico con Kinect"**	Desarrollo de una aplicación cognitiva que ayuden a personas con discapacidad a realizar su rehabilitación.	Sistema de rehabilitación mediante Kinect.	Utilizar el mismo principio para la rehabilitación en los ejercicios realizados en el gimnasio, empleando una interfaz gráfica en lenguaje de desarrollo C#.

4.3 Visión y fisiología del ojo

La visión es unos de los sentidos naturales más importantes de los seres humanos, y es uno de los mecanismos sensoriales de percepción más significativos con los que se cuenta, aunque, a pesar de esto, no impide el desarrollo físico o mental de los individuos que presenten alguna incapacidad parcial o la ausencia total del mismo (Pajares Martinsanz & de la Cruz García, 2008).

De los cinco sentidos humanos naturales: visión, oído, olfato, gusto y tacto, la visión es indudablemente, del que nos podría hacer más dependientes que los demás. Además, es el que provee más información al cerebro, aproximadamente un total de 10 Megabits por segundo (Geijo Vegas, 2016).

Los ojos trabajan las 24 horas del día, incluso cuando el cuerpo se encuentra en reposo o mientras dormimos, y es el que más información envía al cerebro a alta velocidad para que éste la procese y pueda obtener resultados de lo que sucede en el entorno: capta información de objetos, siluetas, colores, contornos, además puede hacerlo en distintas velocidades y es capaz de adaptarse a variaciones del entorno (Colegio de Ópticos, 2017).

El ojo humano (Figura 4.12) se compone de los siguientes partes (Admira Visión, 2016):

- Esclerótica: es la parte blanca del globo ocular, cubre la mayor parte del ojo y su función es proteger básicamente, ya que está formada por un material muy resistente. Además, contiene muchos de los vasos sanguíneos que llevan la sangre al ojo.
- Córnea: es una membrana resistente y transparente que se encuentra en la superficie ocular. Está compuesta por cinco capas, y es a través de ella que la luz penetra en la parte interior del ojo.
- Iris: es una estructura pigmentada suspendida entre la córnea y el cristalino, y tiene una abertura circular en el centro, la pupila. El diámetro de la pupila depende de un músculo que rodea los bordes del iris y controla la cantidad de luz que pasa a través de ella al contraerse o relajarse.
- Cristalino: se trata de una lente natural, que está constituida por un gran número de fibras transparentes dispuestas en capas. Su función es enfocar las imágenes

correctamente en el fondo del ojo. Según la distancia del objeto que se observa (lejos o cerca), el cristalino se engrosará o adelgazará (variando su curvatura) para facilitar una visión nítida.

- Retina: considerada como una prolongación del cerebro, es parecida a una película fotográfica: una lámina compuesta especialmente de millones de células nerviosas. Las células receptoras, que son sensibles a la luz, se encuentran en su superficie exterior, detrás de un tejido pigmentado; estas, según su forma pueden ser llamadas conos y bastones. La retina está situada detrás de la pupila, y tiene una pequeña mancha amarilla llamada mácula; en su centro se encuentra la fóvea central, la zona del ojo con mayor agudeza visual.
- Nervio óptico: mide aproximadamente 4 cm de longitud, y está compuesto por células fotorreceptoras capaces de convertir la luz en impulsos nerviosos. Trabaja en coordinación con el cerebro para realizar funciones de reconocimiento de imágenes o patrones. Estos impulsos eléctricos provienen de los 100 millones de bastones que son los que reconocen el color negro y sus matices, y de los 3 millones de conos que son los responsables de reconocer el resto de los colores.

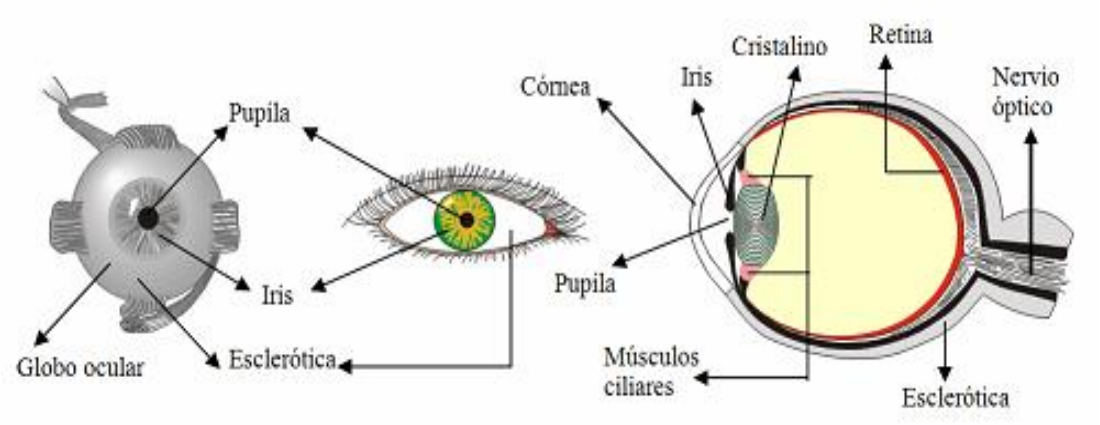

Figura 4.12 Fisiología del ojo humano (Osuna, 2018).

4.4 Diferencias y semejanzas entre la óptica natural y una cámara

El cuerpo humano, con sus distintos sentidos y órganos como un conjunto, es una máquina casi perfecta. Siempre se ha intentado replicar esto mediante el desarrollo de

diversos dispositivos, software y sistemas tecnológicos, pero es prácticamente imposible igualarlos debido a la complejidad de los primeros.

La cámara fotográfica, al igual que el ojo humano, tiene la tarea de capturar imágenes para que posteriormente puedan ser interpretadas por la máquina o el cerebro, según corresponda. Su función es la de hacer que los rayos de luz que inciden se enfoquen en un área determinada, dónde se ubica un sensor, que traduce la imagen en información procesable. Para comprender mejor los párrafos siguientes, refiérase a la Figura 4.13.

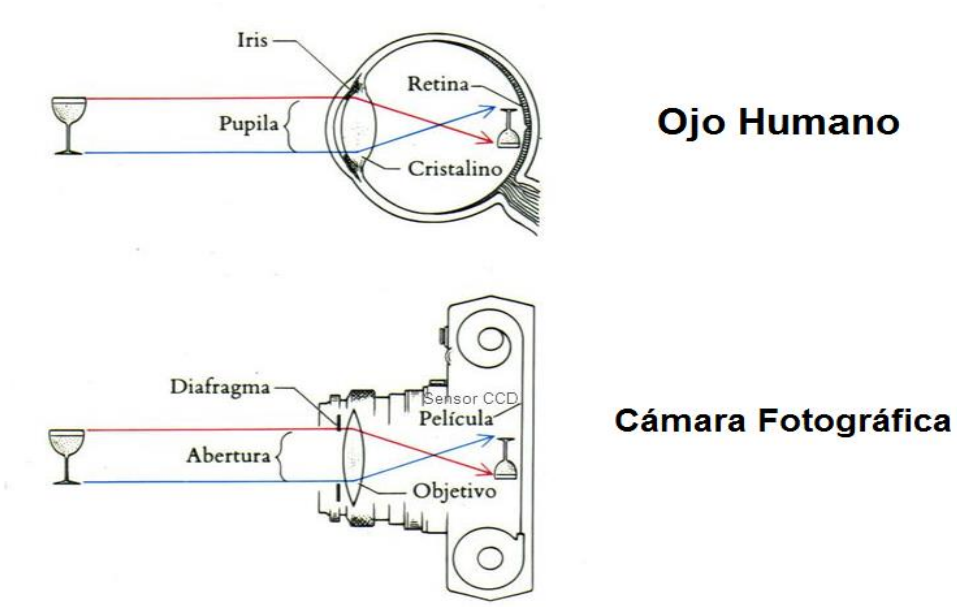

Figura 4.13 Comparativa entre ojo humano y cámara fotográfica (E. Hannula, et al., 2005).

En cuanto a las semejanzas entre el ojo humano y la cámara fotográfica, se puede comenzar con la que tiene el iris y el diafragma. Este último es capaz de regular la densidad de luz que llega a la película, que es necesaria para obtener una buena fotografía. Mientras que el iris funciona de una manera muy similar: éste se contrae y se relaja haciendo pasar más o menor cantidad de luz en la retina; y por obvias razones la pupila haría similitud a la apertura del diafragma (Geijo Vegas, 2016).

También se puede comparar la retina del ojo con la película de la cámara (o un sensor electrónico, en el caso de las cámaras digitales), ya que es el lugar donde va a incidir la luz y va a formarse la imagen.

La cornea y el cristalino son lentes que enfocan los rayos de la luz en un punto exacto de la retina; esta función es la misma que realiza el objetivo de la cámara: si el objetivo no se coloca de una manera adecuada, la imagen que se captará será borrosa o mal enfocada. El nervio óptico es el medio por donde se enviará y procesará la imagen, por lo que se podría comparar con los cables y sistemas de comunicación de una cámara. El cerebro, que interpreta las imágenes captadas por el ojo y permite la toma de decisiones, se puede comparar al procesador digital de una cámara, y en los sistemas modernos de visión artificial, con los algoritmos de procesamiento y reconocimiento de imágenes.

También existen diferencias, como el que la retina es curva, mientras que la del sensor es plana. La retina tiene mayor densidad de células fotosensibles en la parte central de la retina (esclerótica), mientras que, en lo usual, el sensor es uniforme en toda su superficie.

Las cámaras convencionales sólo pueden captar imágenes en dos dimensiones, mientras que el sistema de visión humana cuenta con dos ojos, por lo que se puede obtener una imagen en tres dimensiones y es posible calcular distancias gracias a esto. Algunos sistemas modernos de visión artificial utilizan la visión estereoscópica, o incluso colocan un mayor número de cámaras, con la finalidad de imitar la capacidad de visualizar una escena en tres dimensiones (Martínez, 2010) .

El ojo humano puede captar aproximadamente el doble de tonos entre lo oscuro y lo blanco que la cámara más sofisticada, además de que puede ver de forma detallada en ambientes que cuenten con estos dos tonos; también tiene la capacidad de detectar más colores que cualquiera de las cámaras convencionales modernas (Geijo Vegas, 2016) .

Finalmente, una característica muy importante que ha sido difícil de imitar por los sistemas de visión es que el ojo humano es capaz de adaptarse a cambios de iluminación más rápidamente que una cámara (Rodríguez Mier, 2017) .

4.5 Visión Artificial

La Visión artificial es un campo de la Inteligencia Artificial que, mediante el uso de diversas técnicas, permite la obtención, procesamiento y análisis de diversos tipos de información a través de imágenes digitales. Es una disciplina que abarca la informática, la óptica y la ingeniería. El concepto apareció en el intento por dotar a las máquinas de un sistema de visión parecida al de los seres humanos o animales, así como brindarles un procesamiento de los datos de la escena para la percepción de éstas de forma autónoma (Pajares Martinsanz & de la Cruz García, 2008).

De manera general, la visión artificial es la capacidad de una máquina de ver y percibir el mundo real y todo lo que le rodea; es una manera de percepción del ambiente, con la capacidad de obtener la información de este y procesarla, así como determinar sus propiedades bidimensionales o tridimensionales a partir de imágenes bidimensionales. Concretamente busca imitar la capacidad del ser humano de la visión. Teniendo en cuenta esto, este tipo de sistemas es unos de los más complicados y difíciles de concebir debido a las limitantes de hardware y al complejo sistema de visión natural de los seres humanos; en la actualidad no es posible abarcar e imitar todo un sistema de visión humana (Shapiro & Stockman, 2000).

Algunas de las tareas que se pueden realizar mediante la Visión Artificial incluyen (Visión Artificial, 2012):

- Automatizar tareas repetitivas.
- Detectar colores, patrones, señales, objetos o movimiento.
- Realizar controles de calidad de productos que no es posible verificar por métodos tradicionales.
- Realizar inspecciones de objetos sin contacto físico y generar una calidad casi en su totalidad a una gran velocidad.
- Reducir el tiempo de ciclos en procesos automatizados.
- Determinar la posición de objetos en el espacio.
- Establecer relaciones espaciales entre varios objetos (guiado de robots).
- Mediciones tridimensionales y angulares.
- Determinar coordenadas importantes de un objeto.

El proceso global de un sistema de visión artificial se presenta en la Figura 4.14, en la cual se distinguen 4 módulos:

1. Digitalización, el cual se apoya con la cámara para la obtención de la imagen y se convierte la señal analógica proporcionada por la cámara a una señal digital.
2. Procesamiento se encarga de obtener información de la imagen y su tratamiento con instrucciones precisas.
3. Memoria, que permite el almacenamiento de la información.
4. Visualización, en el cuál convierte la señal digital residente en la memoria, en señal de video analógica para poder ser visualizada.

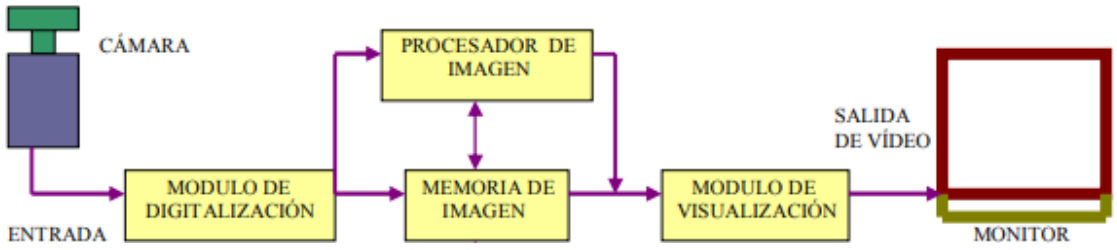

Figura 4.14 Proceso de visión artificial (Centro Integrado Politécnico ETI Tudela, 2017).

El proceso general de un sistema de visión artificial puede ser visualizado en la Figura 4.15, en la que las cajas representan datos y las burbujas procesos. Se parte de la escena tridimensional, pasando por la toma de muestras o imágenes para su posterior segmentación de los bordes y regiones, finalmente terminando con su procesamiento para su aplicación.

En el área de visión artificial se pueden distinguir tres procesos generales: Análisis, Procesamiento y Aplicaciones. El análisis consiste en identificar todas aquellas estructuras elementales de la imagen, tales como los bordes o regiones, así como la relación que hay entre ellas. El procesamiento de enfoca en la manipulación de las imágenes como señales digitales, para extraer la información más elemental de la misma. Finalmente, las aplicaciones tratan de dar solución al uso que pueda surgir en el mundo real, a saber: reconocimiento, movimiento, reconstrucción 3D, entre otros (Pajares Martinsanz & de la Cruz García, 2008).

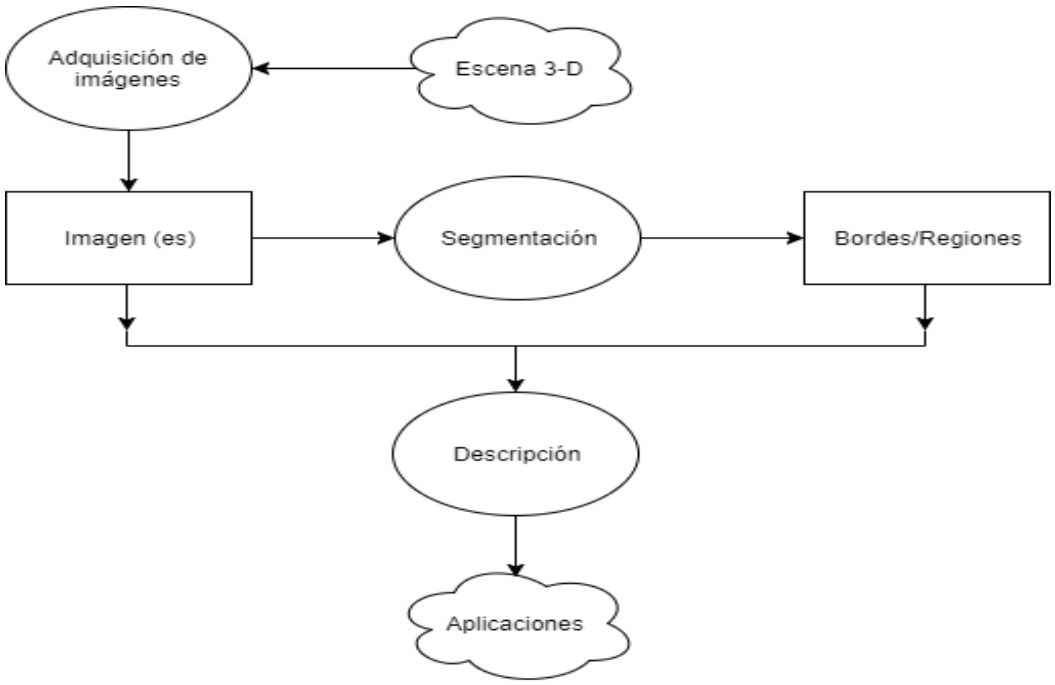

Figura 4.15 Diagrama de bloques Visión Artificial (Shapiro & Stockman, 2000).

El proceso operativo de un sistema de visión (Figura 4.16) se compone de la captación, que es la obtención de la imagen visual del objeto que se inspecciona, las instrucciones, que son todo el conjunto de operaciones a realizar para la resolución del conflicto, el procesado que es el tratamiento de imágenes mediante instrucciones aplicadas y la actuación sobre el entorno (si es necesario como resultado el movimiento de un aparato, pieza o elemento, así como de indicativos adicionales).

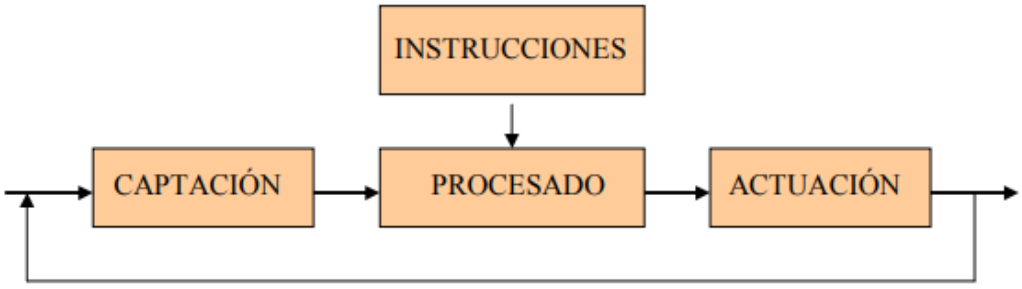

Figura 4.16 Proceso operativo de sistema de visión artificial [(Centro Integrado Politécnico ETI Tudela, 2017)].

4.6 Sensor Kinect®

La monitorización de movimiento del cuerpo humano en la visión artificial es en parte posible gracias a los constantes avances en tecnologías de cámaras y sensores, que permiten la obtención del movimiento a un relativo bajo costo (Martinez Zarzuela, et al., 2011). En el caso particular del Sistema de Visión Artificial propuesto en este documento, "MuscleKIN", para la corrección de posturas en fisicoculturistas se hizo uso del sensor Kinect®.

Kinect® fue desarrollado por PrimeSense para Microsoft Research, que dedicó 20 años de desarrollo en la tecnología, y es responsable de su distribución. Inicialmente conocido bajo el pseudónimo de "Project Natal", anunciado el 1 de junio de 2009 en la Electronic Entertainment Expo (E3) (Webb & Ashley, 2011) (Miramon, 2013).

Es un dispositivo que fue diseñado como periférico de videojuegos de la consola XBOX 360, de la misma compañía, que prescinde de mandos debido a que es capaz de realizar detección de movimiento, de voz, reconocimiento de rostros y objetos en tiempo real; permite al jugador adentrarse a ambientes artificiales y ser partícipe de manera física en el ambiente de los videojuegos, controlar aplicaciones en base a la voz, movimientos y señas.

El propósito inicial de Kinect® fue la de ser usado como un simple controlador de juego; éste permite a los jugadores controlar e interactuar con la consola y sus juegos sin necesidad de un control convencional de juegos o un mando. Todo esto lo puede hacer mediante una interfaz natural de usuario, que es capaz de reconocer gestos, comandos de voz, objetos e imágenes. Es preciso mencionar que es compatible con aplicaciones desarrolladas por estudios de Microsoft o independientes.

Existen tres tipos de dispositivos Kinect®: Kinect v.1 para XBOX 360, Kinect para Windows, y Kinect v.2 para la consola XBOX ONE. El segundo de estos fue desarrollado para la programación de diversas aplicaciones debido a que los investigadores encontraron en este dispositivo un gran potencial para la realización de distintos sistemas debido a las características con las que cuenta, que con las cámaras convencionales sería muy difícil programar o desarrollar. Estos dispositivos simplifican en muchos aspectos aplicaciones reales para problemas reales. Además de ser una

tecnología relativamente económica en comparación con las cámaras que hay en el mercado y con un buen desempeño en relación costo-beneficio.

Cabe aclarar que los otros dos dispositivos desarrollados para sus respectivas generaciones de consolas de videojuegos también son aptos para el desarrollo de aplicaciones, con la única diferencia de requerir una fuente de alimentación externa que a su vez cuenta con un adaptador a USB para su conexión a un equipo de cómputo convencional ya que ésta incluye una entrada específica de sus consolas para las que fueron diseñadas.

Figura 4.17 Versiones de Kinect (Microsoft: Kinect hardware, 2017).

El dispositivo Kinect® cuenta con cámaras y sensores en todas sus presentaciones, siendo más precisas en la última versión de ellas, los cuales, en conjunto, permitirán una detección de movimiento del esqueleto humano y de objetos en tiempo real, haciendo un modelado en 2D y 3D (Microsoft: Kinect hardware, 2017).

El sensor Kinect® cuenta con los siguientes elementos, que son enumerados acorde a como se muestra en la Figura 4.18 (Microsoft: Kinect hardware, 2017):

1. Cámara RGB:

   Una cámara RGB que capta imágenes del entorno; tiene un flujo de informaciónde 32 bits de color, y proporciona imágenes a una frecuencia de actualización de 30 Hz (30 FPS, *frames per second* o cuadros por segundo), con una resolución VGA (640x480 píxeles).
2. Sensores de profundidad 3-D:

Los sensores tridimensionales hacen un seguimiento de cuerpo dentro del área limitada de visión. Está compuesto por un proyector de luz infrarroja, combinado con un sensor CMOS monocromo. El flujo de información es de 16 bits, con la misma frecuencia que la cámara RGB y resolución de 640x480 píxeles a 30 FPS.

3. Multiarray de micrófonos:

   Usa un conjunto de cuatro micrófonos en el borde frontal inferior del sensor, especialmente distribuidos que le permiten determinar la fuente del sonido y eliminar ruido ambiente; exclusivo para su uso en aplicaciones de sonido y el reconocimiento de voz, capaces de procesar audio con 16 bits de resolución y con una frecuencia de muestreo de 16 kHz.

4. Inclinación motorizada:

   Un impulso mecánico en la base del sensor Kinect® inclina de manera automática el sensor hacia arriba o abajo según sea necesario, proveyendo movilidad y adaptabilidad en su visión. El rango de cabeceo que permite es de 54º (± 27º).

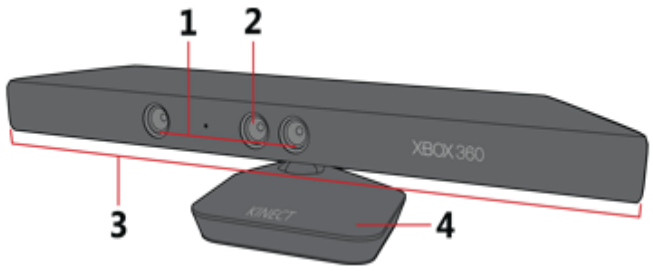

Figura 4.18 Componentes de Kinect (Microsoft: Kinect hardware, 2017).

Lo campos de visión de Kinect son las siguientes:

- Rango de profundidad: 1,2 – 3.5 metros (Figura 4.21).
- Campo de visión horizontal: 57º (Figura 4.19).
- Campo de visión vertical: 43º (Figura 4.20).
- Con la posibilidad de rastreo de hasta 5 personas, pero 2 esqueletos a la vez a través de la detección de 20 de las articulaciones del cuerpo humano.

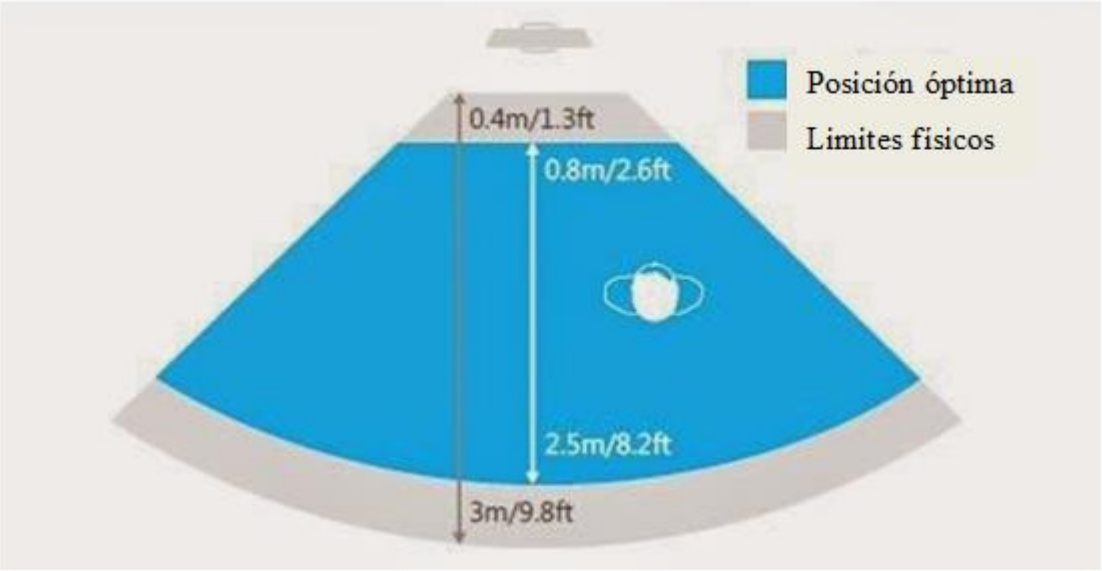

Figura 4.19 Campo de visión Horizontal Kinect.

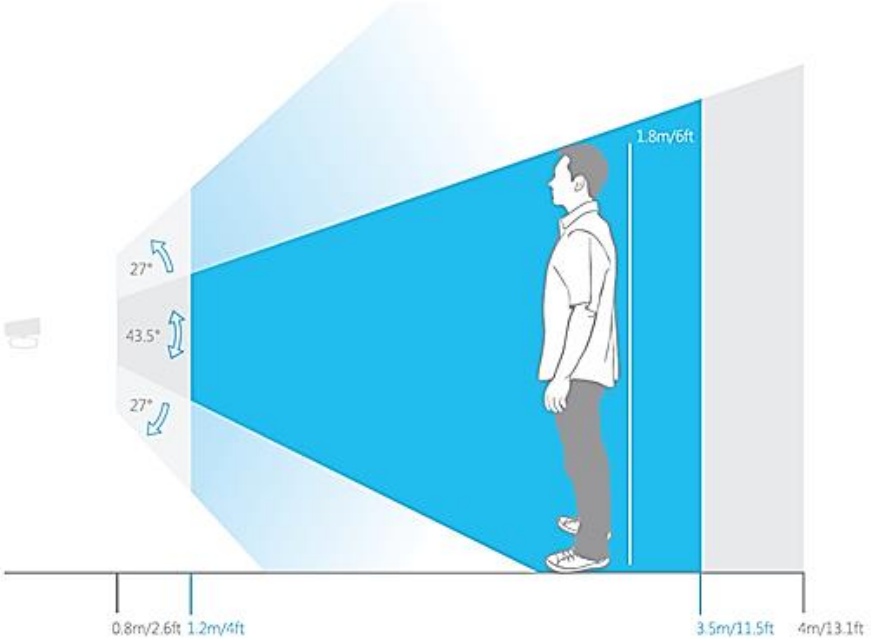

Figura 4.20 Campo de Visión vertical Kinect.

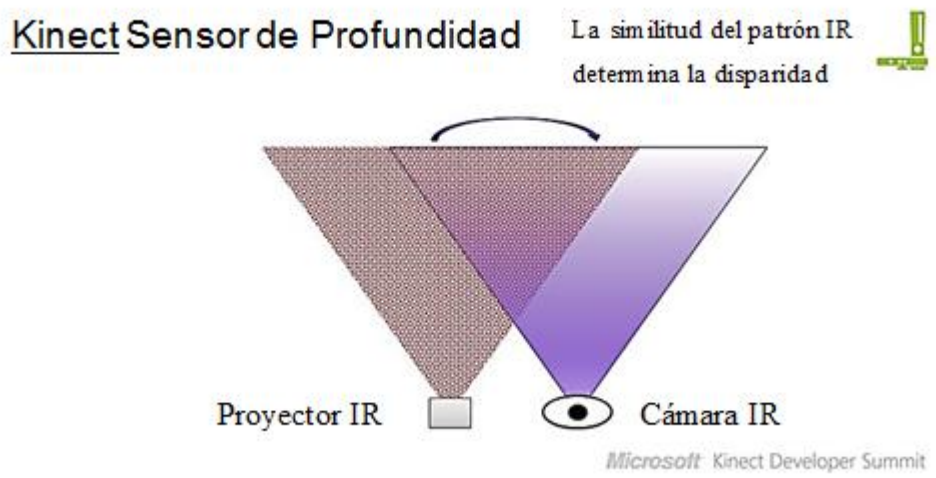

Figura 4.21 Campo de visión de cámara de Profundidad.

Gracias a las características con las que cuenta, ha sido objeto de diversas investigaciones; los desarrolladores de software ahora pueden hacer uso del dispositivo para programar toda una serie de aplicaciones, cuyo principal objetivo es la interacción con ambientes virtuales a través de los distintos movimientos del cuerpo humano, señas y/o voz (Murillo, 2017).

Cabe mencionar que existen factores externos que pueden afectar a la captura de la imagen del sensor Kinect, así mismo sólo se puede detectar hasta 5 personas en una sola escena, pero la detección de dos esqueletos, por ello se debe analizar la biblioteca de instrucciones adecuada para la aplicación, ya que también radica en ella la precisión y desempeño de las cámaras con las que cuenta (Martinez Zarzuela, et al., 2011).

En el trabajo realizado por Leandro Sardi se presentan algunas de las dificultades presentes en la captura de una imagen. Esto se debe tomar en cuenta, ya que Kinect cuenta con cámaras sensibles por lo que es susceptible de presentar algunas de ellas, por ejemplo: iluminación, movimientos, fondos, ropa del usuario, obstáculos (Sardi, 2012).

Las características con las que cuenta Kinect en el seguimiento y asignación automática del cuerpo permiten el desarrollo de aplicaciones integrando como control de esta al usuario haciendo de ésta una buena opción para la práctica de la actividad física. Los

exergames se convierten en partes fundamentales de la actividad física que suple necesidades en los usuarios al momento de realizar la actividad física, a través del juego y la competencia. Despertando así un interés, éstos pudiendo ser más aplicables al área del fitness o de la rehabilitación, desarrollando una herramienta que incremente dicho interés por la actividad física, o en su caso supla lo tedioso de realizarla, así como que ésta misma proporcione información de su correcta realización (Edison Muñoz, et al., 2013).

4.7 Entornos de Desarrollo Integrado (IDE), Bibliotecas y Suite de Desarrollo

Tras su lanzamiento, el sensor Kinect generó gran interés por parte de desarrolladores de aplicaciones que le veían un potencial enorme en diversos ámbitos. Sin embargo, se imposibilitaba en cierta medida la investigación y desarrollo debido al hecho de ser un dispositivo destinado a su uso exclusivo en videojuegos, y que su desarrollador hasta el momento no había liberado ningún tipo de *driver* (controlador de hardware) para su uso en algún entorno de desarrollo.

Un entorno de desarrollo integrado (IDE, por las siglas de *integrated development environment*) es un programa compuesto por un conjunto de herramientas de programación, pudiendo ser capaces dedicarse a diversos lenguajes de programación como: C, C#, C++, JAVA, Visual Basic, Python. Este consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI, *graphic user interface*). Las aplicaciones dentro de los IDE's pueden ser independientes o aplicables a programas existentes (Miramon, 2013).

Con base en esto, los desarrolladores crearon diversas bibliotecas de código y *drivers* que permitían trabajar con el sensor Kinect en el ordenador y que empleaban una serie de marcadores generados en los puntos de flexión del cuerpo. Ejemplo de ellas son las desarrolladas por usuarios a través OpenNI (Open Natural Interaction) o en OpenCV (Open Source Computer Vision, por sus siglas en inglés) (Escenografía Intermedial, 2012).

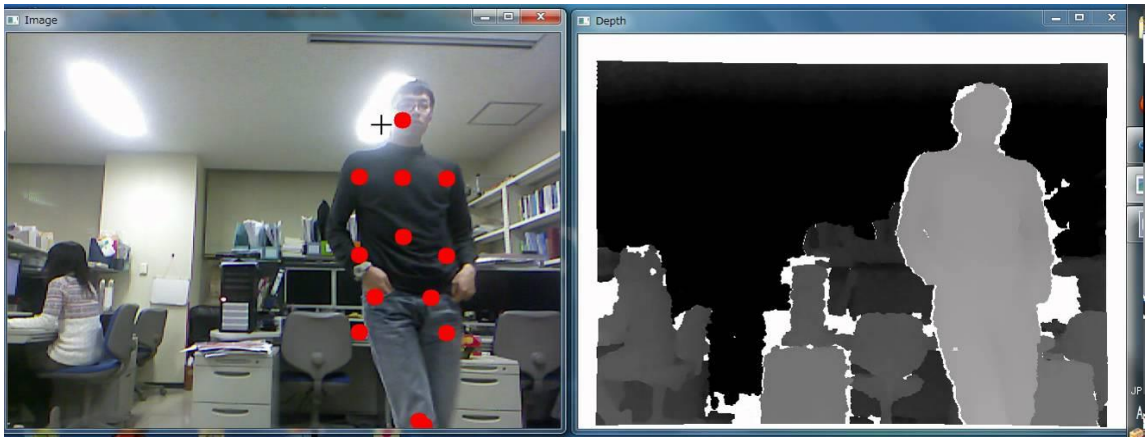

Figura 4.22 Detección de articulaciones con un driver en OpenNI [Canal de YouTube: Nguyễn Văn Vương https://www.youtube.com/watch?v=8Jq2UkPREKg].

OpenNI es un Framework multiplataforma gratuita para escribir aplicaciones donde la interacción humana y un dispositivo basado en uno de los sentidos naturales (vista, oído) actúen entre sí; estos pueden ser: gestos, reconocimiento de voz, o el seguimiento del cuerpo humano (Pfeiffer, 2011).

Su finalidad principal es la de definir una serie de datos como lo son mapas de colores RGB o de profundidad, poses de usuario, etc., y de una interfaz encargada de generar todos estos datos a través de diversas herramientas y métodos (algoritmos, sensores, etc.) para que sea posible trabar con ellos. Permite la captura de movimiento en tiempo real, el reconocimiento de gestos con las manos, el uso de comandos de voz, y la posibilidad de analizar la escena, detectando figuras u objetos situados en la visión del sensor, así como del fondo de la escena (Andreo, 2012).

En el foro oficial de National Instruments, que es un importante punto de encuentro e intercambio de conocimientos entre los usuarios de esta plataforma, el usuario con seudónimo “Naity” implementó una interfaz de seguimiento del cuerpo humano usando Kinect, mediante el desarrollo de una librería generada a partir de ingeniería inversa, combinándolo con el entorno de desarrollo LabView, basado en la interfaz de OpenNI, lo cual puede ser aplicable en el seguimiento del esqueleto humano para la detección y corrección de los ejercicios mediante el desarrollo de otro lenguaje de programación como lo es el lenguaje G ("Naity", 2017).

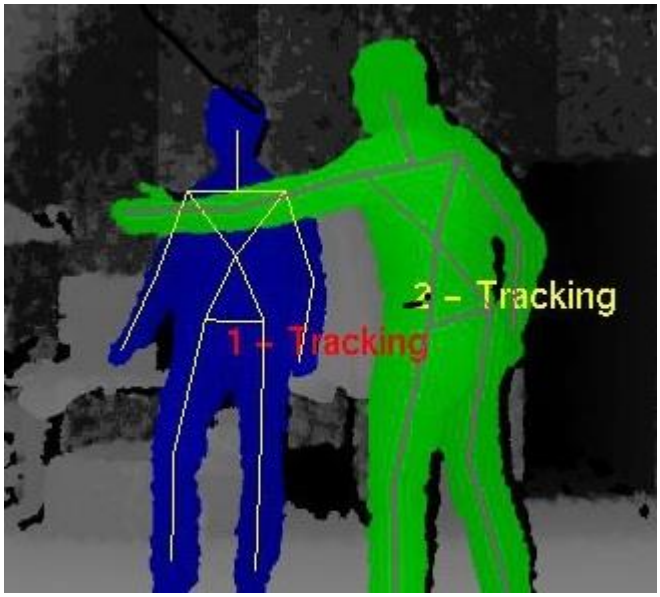

Figura 4.23 Detección de dos esqueletos en OpenNI [http://www.signalprocessingsociety.org/].

Aunque los diversos *drivers* y bibliotecas de OpenNI y OpenCV permiten el trabajo con las cámaras y sensores con los que cuenta Kinect, es imprescindible puntualizar que para asegurar su funcionamiento correcto es necesaria la instalación de los *drivers* oficiales de PrimeSense.

El kit de desarrollo de software (SDK: Software Development Kit) oficial fue liberado en junio de 2011, permitiendo a los desarrolladores la creación de aplicaciones que soporten reconocimiento de gesticulaciones del cuerpo y de sonido, siendo posible programar en los lenguajes de programación C#, C++, o Visual Basic, pudiendo acoplarse con otros entornos de desarrollo como Matlab y Unity 3D. Dicho SDK incluye controladores que permite utilizar Kinect en diversas versiones del sistema operativo Windows (versión 7 u 8, de 32 o 64 bits); además, incluye las API's (*Application Programming Interface*, Interface de Programación de Aplicaciones), las interfaces del dispositivo, así como documentación y una descarga adicional que incluye información de programación y algunos ejemplos ejecutables y su respectivo código fuente del uso de los distintos componentes de Kinect.

A la fecha que se realizó este proyecto de tesis se encontraron diferentes versiones de los SDK liberadas por la propia compañía dueña del dispositivo Kinect: Microsoft. La versión 1.7 del SDK trata de una biblioteca de funciones que facilita la utilización de la primer dispositivo Kinect pensada únicamente para su utilización en la consola XBOX 360; la versión 1.8 fue una actualización para mejorar algunos de sus funcionalidades y para la compatibilidad con el dispositivo Kinect para PC que fue pensando para el desarrollo de aplicaciones, y la versión 2.0 que fue publicada para la compatibilidad con el tercer dispositivo Kinect que fue pensado para su utilización exclusiva con la consola XBOX ONE, el cual incluye mejoras sustanciales en la precisión, sensibilidad y profundidad de las cámaras, sensores y micrófonos, la sensibilidad es 3 veces más sensible, puede reconocer partes pequeñas del cuerpo humano, así como las arrugas de la ropa, el campo de visión es 60º más amplio, el dispositivo es capaz de detectar personas y su movimiento en ambientes oscuros, así como el incremento en el número de articulaciones, pues también es capaz de detectar los músculos de la cara y los dedos de la mano.

En el caso particular del desarrollo del sistema para la corrección de posturas “MuscleKIN” contó con la primera versión del dispositivo: Kinect para XBOX 360, por lo que se procedió a instalar el SDK 1.7 para obtener una compatibilidad óptima con la computadora, permitiendo el desarrollo de la aplicación.

La ventaja de usar el SDK oficial es que es adaptable al entorno de desarrollo Visual Studio, permitiendo conjuntarlo a otros IDEs, haciendo la programación de aplicaciones mucho más sencilla, adecuada y eficaz, permite adquirir ciertas características o ejemplificar la programación de otras debido a sus instrucciones, eventos, instancias y clases, incluso herramientas específicamente implementadas para la programación de aplicaciones y por sus librerías.

Microsoft Visual Studio es un entorno de desarrollo integrado para Windows, y actualmente con soporte para sistema operativo MacOS; soporta lenguajes de programación de tipo C, C#, C++, Visual Basic .NET, Visual J#, F#, JAVA, PHP, y permite crear o desarrollar sitios y aplicaciones, así como servicios de la plataforma .NET (MSN, 2017).

La compatibilidad con Kinect empieza a partir de la versión Visual Studio 2010 y 2012, aunque puede ser usado con versiones posteriores sin ningún tipo de inconveniente. Para el desarrollo del sistema "MuscleKIN" se realizó en el IDE Visual Studio 2017.

Como lenguaje de programación se optó por el desarrollo de la aplicación en C#, debido a las ventajas que este lenguaje posee, C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft y es de los lenguajes de programación más utilizados para el desarrollo de aplicaciones; su sintaxis se derivó de los lenguajes C/C++ y utiliza el mismo modelo de objetos de la plataforma .NET, similar al de Java, aunque con algunas mejoras especialmente comparado con los lenguajes C/C++ como el recolector de basura que libera al programador del manejo de la memoria (Miramon, 2013).

C# está disponible para el desarrollo de aplicaciones eficaces con interfaz gráfica, aplicaciones para internet y aplicaciones para móviles. Además, siendo parte del paquete de desarrollo de Visual Studio, permite una mayor fluidez de programación, pudiendo ser más gráfica en la creación de aplicaciones tradicionales y web, haciendo uso del editor de código avanzado, el depurador, y los diseñadores de interfaces de usuario y otras utilidades con las que cuenta y que facilitan enormemente la programación y desarrollo de las aplicaciones.

4.8 WPF, XAML

WPF (*Windows Presentation Foundation*, por sus siglas en inglés) es la parte más grande del *framework* de desarrollo .NET. Es una serie de ensamblados y herramientas de dicho *framework*, destinada a proporcionar una Interfaz de programación de aplicaciones, API, para crear interfaces de usuarios (UI) para aplicaciones de escritorio. Abarca muchos conceptos de las UI desde XAML (*eXtensible Application Markup Language*, Lenguaje Extensible de Formato de Aplicaciones) hasta patrones de diseño. Combina diferentes plataformas desde el desarrollo de aplicaciones para Windows, aplicaciones de internet enriquecidas (RIA) como las animaciones, recursos multimedia,

y gráficos; y el desarrollo web con la utilización de un lenguaje de meta etiquetas para el desarrollo de interfaces gráficas (Hernández, 2016) (Microsoft, 2017).

Esta plataforma ofrece una infraestructura y potencia gráfica para el desarrollo de aplicaciones de visuales atractivas, con herramientas de fácil interacción. Maneja el lenguaje declarativo XAML que es el lenguaje basado en XML (*eXtensible Markup Languaje*, por sus siglas en inglés) para la creación de aplicaciones gráficas ricas en contenido visual. Es de uso típico por entornos de desarrollo como Visual Studio que ofrecen diseños de este tipo.

Generalmente cuando se usa WPF, XAML es el lenguaje usado para describir interfaces visuales de usuario. Soporta componentes como gráficos, animaciones, audio, video, eventos por medio de código o herramientas de desarrollo gráficas (Miramon, 2013).

5 MÉTODO

El desarrollo del sistema “MuscleKIN” se llevó a cabo mediante la metodología de desarrollo de software en cascada con retroalimentación, ya que, al ser de un problema real, fue necesario seguir todo un proceso completo de análisis y diseño el mismo que indicara de manera concreta qué factores y requerimientos se necesitaban, tanto para su programación y diseño, como de las necesidades ante usuarios reales, y en qué condiciones eran indispensables. Posterior a esto se realizó el proceso de codificación del sistema, la fase de pruebas, su implementación, documentación y de ser necesario su mantenimiento, así justificando todas las fases de dicha metodología.

El método de desarrollo en cascada con retroalimentación es un modelo lineal, en el que el software es dividido en cinco fases que deben procederse de forma secuencial; sin embargo, es posible regresar a la etapa anterior en cualquier momento en caso de requerirse. Por ello cuando se finaliza una de las fases se realiza un a revisión, con el fin de determinar si está en condiciones de avanzar a la siguiente fase (Pantaleo & Lis Rinaudo, 2015).

La Ingeniería de Software propone como a la metodología cascada como la más completa para el desarrollo del Software, teniendo en consideración las siguientes fases generales del proceso (Pantaleo & Lis Rinaudo, 2015):

1. Definición del Software: Corresponde a la visión del producto, a sus aspectos desde el punto de vista final.
2. Análisis de requerimientos: Implica el entendimiento del dominio del producto a ser desarrollado; esto es funciones, comportamiento y relación con sistemas externos.
3. Diseño: Es la forma en que la solución se implementará, esto se basa en análisis previos, esto incluye diagramas, algoritmos.
4. Codificación: Corresponde a la implementación de la solución de acuerdo a cómo se ha estipulado durante el diseño.
5. Pruebas: Se debe asegurar que el producto satisfaga los requerimientos, es decir que cumplan con el comportamiento esperado. Esto se hace con el fin de

asegurar la calidad del software y reducir los riesgos de falla de la aplicación en el entorno final de la implantación.

En la Figura 5.1 se muestra el diagrama general de los procesos llevado a cabo en la metodología de desarrollo de software en cascada.

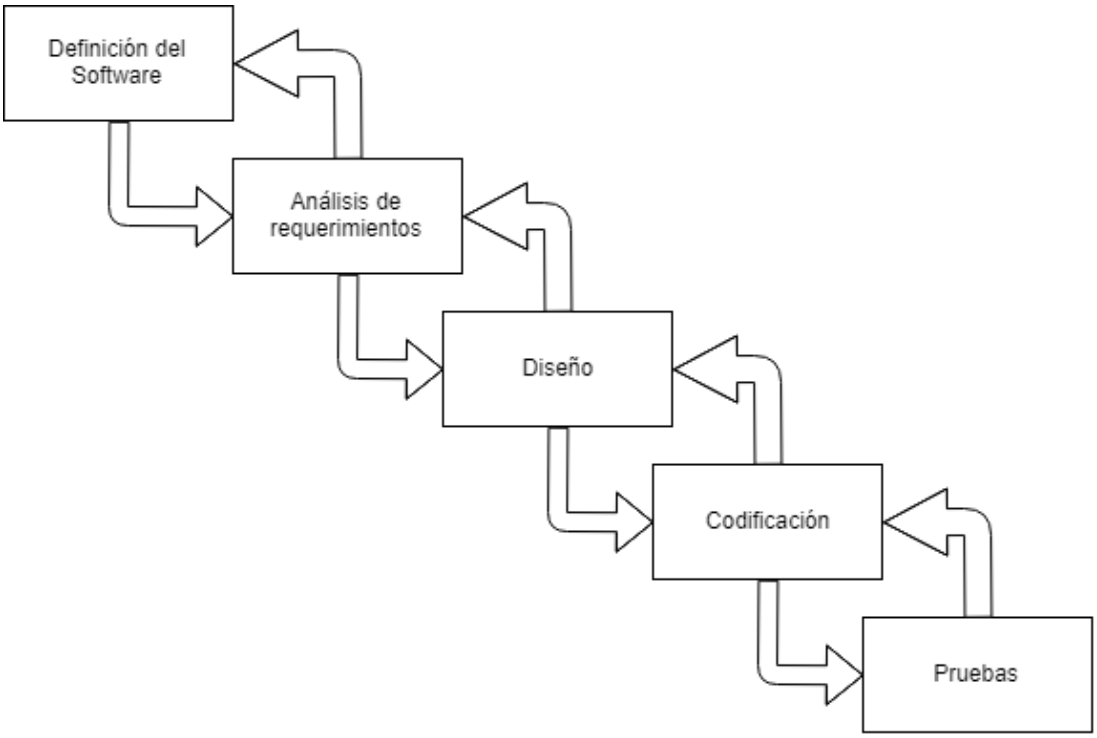

Figura 5.1 Etapas de metodología de desarrollo de software en cascada.

El desarrollo de trabajo empezó la definición del software y con el análisis de todos los antecedentes, requerimientos y documentación del diverso estado del arte. Un análisis pertinente y detallado del problema a resolver, en el caso concreto, el desarrollo de un sistema que sea capaz de detectar la figura humana, procesar la imagen para corregir la postura de la ejecución de ejercicios realizados por los fisicoculturistas en los gimnasios en tiempo real. Así como de las variables que puedan afecten el funcionamiento del dispositivo, y el entorno de desarrollo a utilizar, así como el análisis del dispositivo como el más eficaz y conveniente para la aplicación.

El proceso de diseño, tanto de la Interfaz Natural de Usuario como del sistema en general que contenga apartados de información respecto a los ejercicios, y las respectivas ventanas de corrección de posturas, la generación y diseño de un algoritmo que sea capaz de corregir las mismas en base a la postura de las articulaciones en 3D, así como de las diversas aplicaciones adicionales como contador de ejercicios, y la variación de estas.

Seguido de lo anterior, el tercer aspecto consistente en la conexión del dispositivo con la computadora, además de la programación de las interfaces y algoritmos; por último se considera el testeo del producto final en distintos ambientes, tanto controlados como laboratorios, salones y centros educativos de la Universidad UAEM Plantel Atlacomulco, así como en ambientes reales, concretamente un gimnasio de pesas convencional, la programación para la corrección de errores o "bugs" que pudiesen surgir de la programación principal. Por último, se puntualiza la documentación general del proyecto y los resultados generados durante el proceso de investigación y experimentación, así como la comprobación de éstos, en comparación a la hipótesis.

En la Figura 5.2 se presenta el diagrama de flujo del desarrollo general del sistema siguiendo la metodología usada para su elaboración.

5.1 Requerimientos o especificaciones

5.1.1 Hardware

Para el apartado de Hardware como ya bien se mencionó se optó por hacer uso del dispositivo Kinect para la detección del movimiento del cuerpo humano ya que cuenta con lo esencial para llevar a cabo la programación de una aplicación de éste tipo; cuenta con una cámara RGB (Red, Green, Blue <<rojo, verde, azul>>) que permite la captura del entorno tratando la señal de vídeo por separado de los tres colores primarios, proporcionando mayor calidad y una reproducción más fiel al color, y sensores de profundidad que ayudan a la detección de imágenes en 3D, ésta funciona midiendo las distancias entre la cámara y el objeto que se detecte. Además, se caracteriza por tener en su arquitectura un multiarray de micrófonos y una base motorizada.

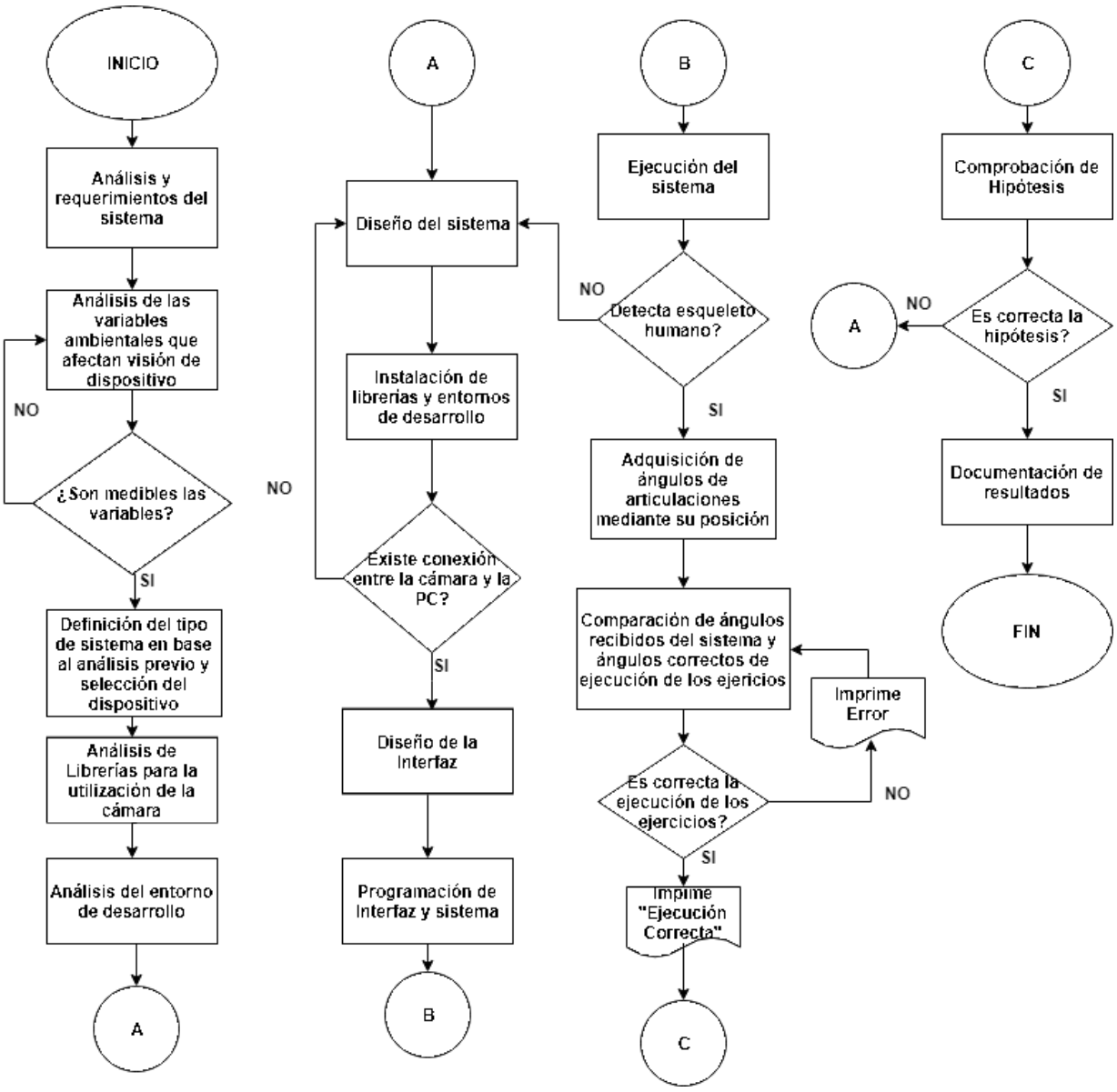

Figura 5.2 Diagrama de Flujo: Metodología del sistema "MuscleKIN".

La cámara RGB con la que cuenta es una cámara digital. La luz atraviesa una lente que la dirige a un filtro encargado de separarla en los colores primarios (rojo, verde y azul), proyectados sobre un sensor fotosensible. El sensor genera una señal eléctrica en función de la intensidad de la señal que incide sobre él. Esta señal es convertida a digital mediante un ADC (Analog Digital Converter) que es analizada y reconstruida para su almacenamiento.

Así mismo se hizo uso de un equipo de computación con los requerimientos mínimos para soportar la tecnología Kinect y los entornos de desarrollo. En el caso del sensor Kinect se necesita:

- RAM 4GB (Recomendado). Mínimo de 2 GB (64 bit) o 1GB (32 bit).
- Procesador de 64 bits CPU I7 3.1 GHz (Recomendado). Mínimo procesador de 1 GHz.
- Controlador USB 3.0.
- Tarjeta gráfica que admita DirectX 9.0 o superior.
- Sistema Operativo Windows7, 8 u 8.1, Windows Embedded 8 o Windows 10.
- Adaptador USB para Kinect (XBOX 360).
- SDK de Kinect (Microsoft: Kinect Hardware setup, 2017).

En el caso concreto de la computadora que se utilizó para la programación y pruebas del sistema, ésta cuenta con las siguientes características:

- Procesador Intel Core i3 2.40 GHz.
- 4 GB memoria RAM.
- Disco Duro 265 GB.
- Sistema operativo Windows 8, 64 bits.

5.1.2 Software

Para la programación de la interfaz y de los algoritmos de captura de movimiento, del Skeleton Tracking y de la comparativa de las posturas se hizo uso de un entorno de desarrollo que fuera viable con respecto a las características y requerimiento que son necesarias para la aplicación, entre los entornos factibles se encontró el distribuido por la compañía Microsoft: Visual Studio, el cual después de hacer investigación en el estado del arte, y tras la pruebas realizadas se encontró como la más viable para la programación ya que la propia compañía desarrolladora liberó los SDK, así como la documentación de los algoritmos que usa el sensor Kinect, y algunos ejemplos de programación, mediante el lenguaje de desarrollo Visual C, C++ y C# convenientemente desarrolladas a través de este entorno de desarrollo y su plena compatibilidad, esto

implica una comunicación entre Kinect y computadora más estable, con instrucciones de programación específicas, generales y de fácil manejo.

Las características o requerimientos mínimos que se necesitan para el entorno de desarrollo Visual Studio:

- Procesador 1.6 GHz.
- 1024 MB de memoria RAM.
- 3 GB de espacio disponible en Disco.
- Tarjeta de video compatible con DirectX 9 con una resolución de pantalla de 1024 x 768 o superior (Microsoft: Visual Studio requeriments, 2017).

5.2 Diseño e implementación

Para el diseño del sistema se trabajó una de las 5 etapas requeridas de la metodología en cascada seleccionada para su desarrollo, el análisis previo arrojó resultados concretos de cada característica del sistema acorde a las necesidades de un usuario fisicoculturista avanzado siendo aplicable a practicantes del fisicoculturismo desde el nivel principiante hasta del nivel del que fue basado. Siendo éstas:

- Correctas posturas de los ejercicios que no incluyan rangos de movimiento que puedan generar resultados contraproducentes a los esperados como lo son las lesiones.
- Considerar movimientos adicionales incorrectos al realizar el ejercicio como el balanceo o la inclusión de otros músculos no propios del ejercicio que se realiza.
- Capacidad del sistema de ignorar accesorios, como mancuernas o barras olímpicas.
- Fácil de usar.
- Interacción completa del usuario.
- Capacidad de variar y contar automáticamente las repeticiones del ejercicio que se esté realizando, dando un mensaje claro que indique que si se cumplió el numero establecido de repeticiones, esto en caso de que su plan de entrenamiento exija un número exacto, para enfocarse únicamente en la realización de este.
- Apartado de información de la ejecución de los ejercicios.

Los requerimientos que componen al sistema han sido mencionados con anterioridad, siendo éstas concretamente el dispositivo Kinect, el entorno de desarrollo Visual Studio que de acuerdo con el análisis fue el más viable la programación por el fácil manejo, y la simplificación de instrucciones que el SDK oficial de Microsoft proporciona al entorno, con la opción de programación en C++, C#, o Visual Basic, siendo C# como lenguaje de programación seleccionado.

Los pasos que seguirá el sistema es el siguiente: Captura de la imagen de la cámara RGB, y de profundidad, definición de la región de interés, es decir la obtención y visualización de información de las articulaciones que serán medidas para su comparación, la inicialización de tolerancias en las posiciones y generar finalmente una salida apropiada para el usuario.

En la Figura 5.3 se maneja el diagrama de interacción del sistema, en el que el usuario se sitúa frente al sensor Kinect, generando el movimiento de los ejercicios, para su posterior envío de la información captada para su procesamiento en una computadora, que además brindará en tiempo real la información e indicativos visuales necesarios para la corrección de las posturas a través de una interfaz mostrados en una pantalla capaz de brindar una imagen clara, de tamaño considerable para ser visible en los puntos de inicio.

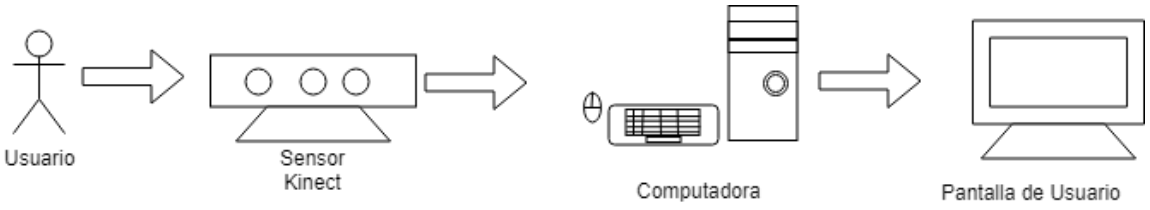

Figura 5.3 Diagrama Interacción Usuario-Sistema.

Uno de los puntos que se consideró como importante por la comunidad de fisicoculturistas era el apartado de fácil interacción, entendimiento y uso, por ello se diseñó la implementación de NUI (Interfaz Natural de Usuario, por sus siglas en español) que, mediante la gesticulación del cuerpo, específicamente de los brazos y las

mano sea capaz de interactuar en función de puntero con el sistema “MuscleKIN”. En la Figura 5.4 se muestra el diagrama de flujo para la programación de la NUI del sistema.

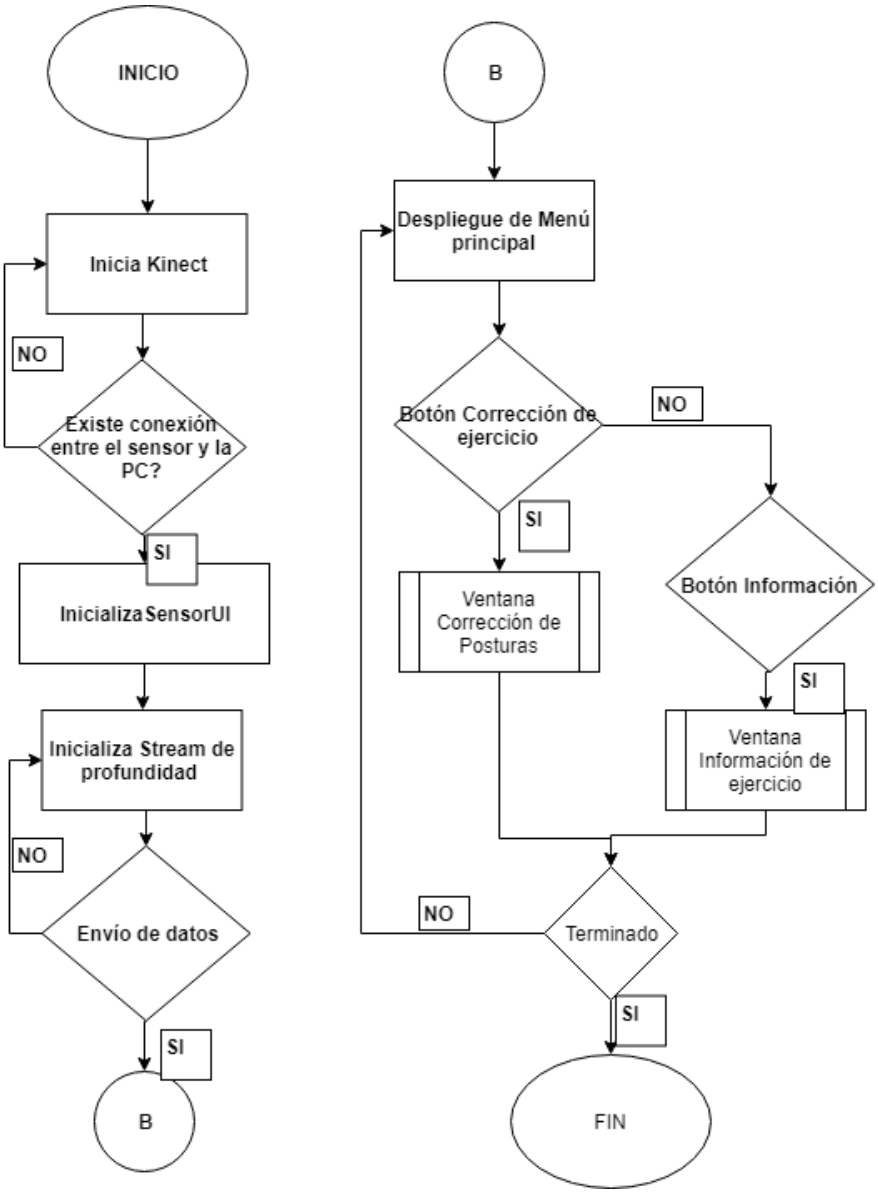

Figura 5.4 Diagrama de Flujo: NUI MuscleKIN.

La parte fundamental del sistema es la corrección de posturas por ello se hizo un diseñó independiente en el que se tuvieron como puntos importantes la utilización de contadores de repeticiones y los 3 distintos *Streams* de Kinect:

- Cámara RGB: Ofrecerá la imagen del usuario en función espejo.
- Cámara de profundidad: Ofrecerá información en 3D del usuario, detectará usuario principal.
- Skeleton Tracking: Ofrecerá información de las articulaciones posicionados en el sistema de ejes de 3 dimensiones que permitirá la corrección de las posturas en los ejercicios, la detección de colisión de bordes, y dibujar en la interfaz el esqueleto en tiempo real.

En la Figura 5.5 se muestra el diagrama de flujo del sistema de corrección de posturas de "MuscleKIN", mientras que en la Figura 5.6, se puede apreciar el diagrama de secuencia de la aplicación, identificando el flujo de información en el sistema.

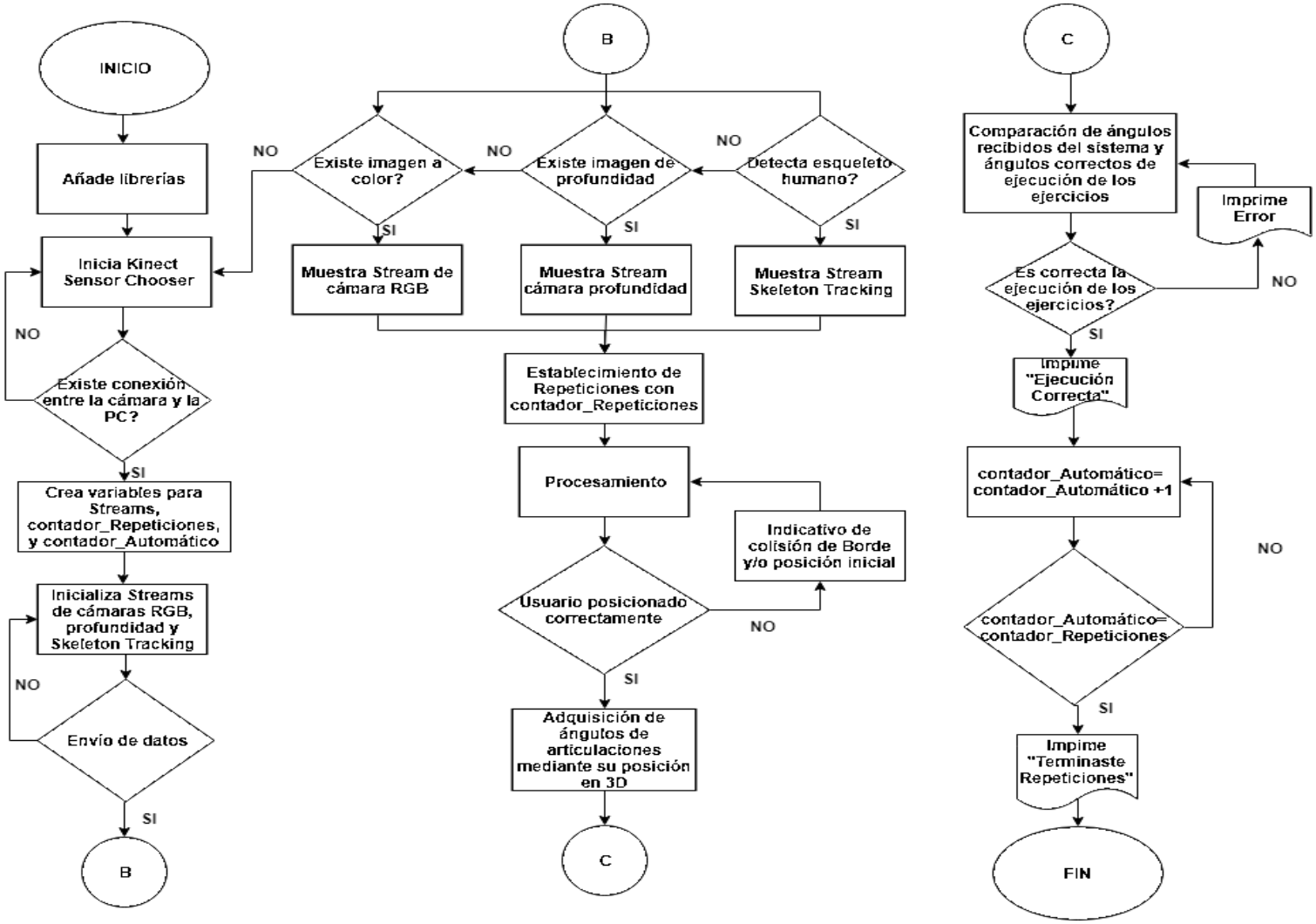

Figura 5.5 Diagrama de Flujo: Sistema para corrección de posturas "MuscleKIN".

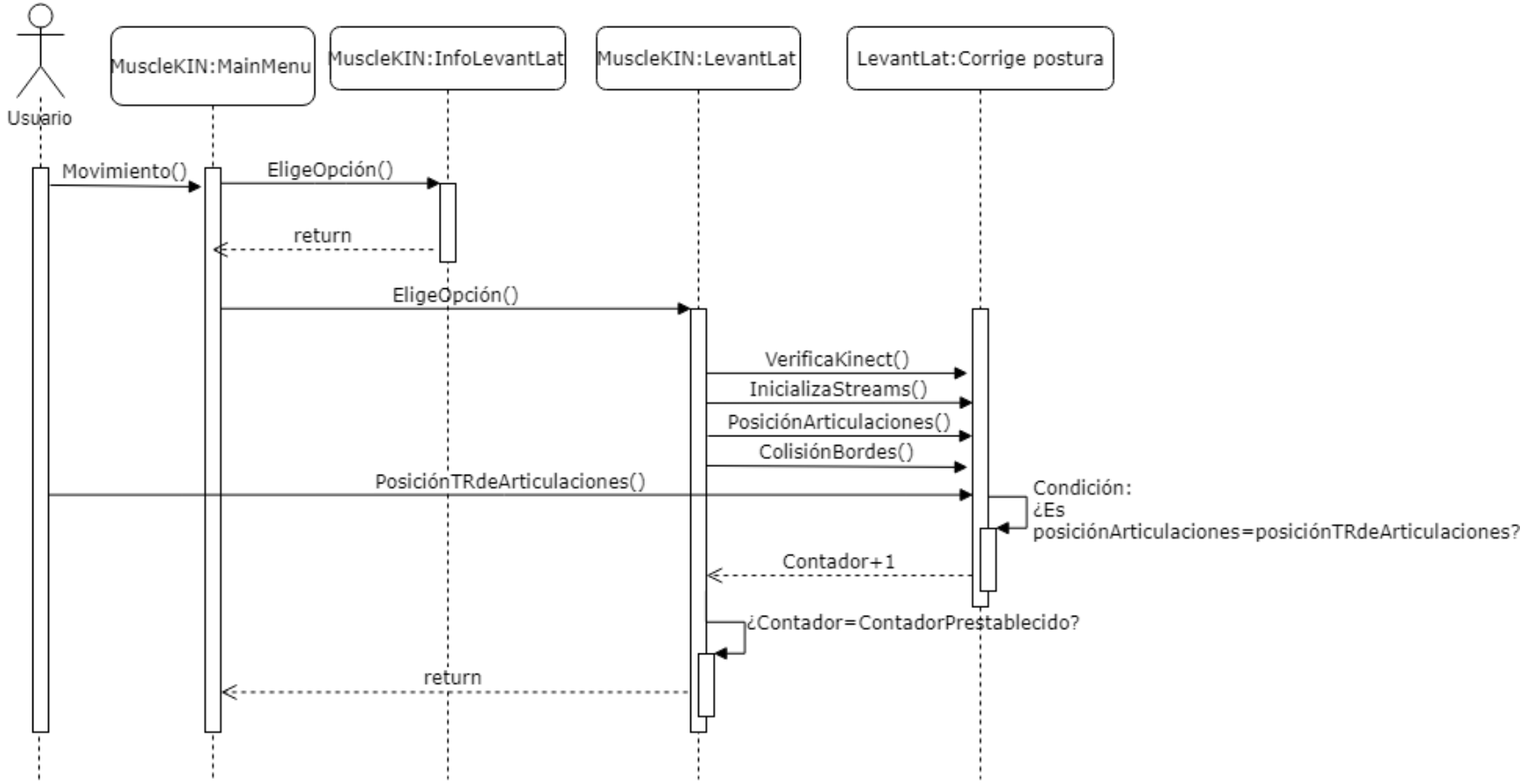

Figura 5.6 Diagrama de Secuencia Software.

5.3 Experimentación

El comienzo del desarrollo del sistema "MuscleKIN" se generó a partir de la búsqueda de un controlador (*driver*) que se adecuara a las necesidades y principios fundamentales establecidos previamente, es decir en el análisis de necesidades.

La programación del sistema para la corrección de posturas en fisicoculturistas fue realizada mediante la utilización de la suite de desarrollo del dispositivo SDK 1.7 KINECT, que incluye Drivers, APIs y herramientas que permiten una adaptabilidad más adecuada frente a OpenNI de desarrollo libre, aunque, ambas permiten el reconocimiento de las articulaciones del cuerpo humano, figuras y de escena, además de la activación de las distintas cámaras o funcionalidades para la operación de éstas; cabe mencionar que siendo este SDK instalado, no habrá compatibilidad con dichos drivers de desarrollo libre (Quinche Curtidor, 2015).

Para la instalación del SDK únicamente se deben seguir los pasos que indica el asistente; también se ofrece la instalación de Kinect Development Kit el cual es una herramienta opcional que contiene documentación del sensor, de su operatividad y de ejemplos de código fuente (véase Figura 5.7); así mismo incluye ejecutables, con la opción de generar sus respectivos archivos y códigos fuente en el entorno de desarrollo Visual Studio de las diversas aplicaciones, que comprende funcionalidades con todos y cada uno de los componentes de hardware con los que cuenta Kinect; así como de la instalación de herramientas adicionales para el desarrollo como lo es Kinect Studio que permite grabación en video para realizar prueba en Kinect sin necesidad de hacerlas en tiempo real, y enlaces a entornos con los que se puede trabajar el sensor Kinect como XNA Game Studio y DirectX.

Se debe tener en cuenta que antes de la instalación, el sensor Kinect se encuentre desconectado, y si el desarrollo de la aplicación será mediante el IDE Visual Studio éste debe estar inactivo.

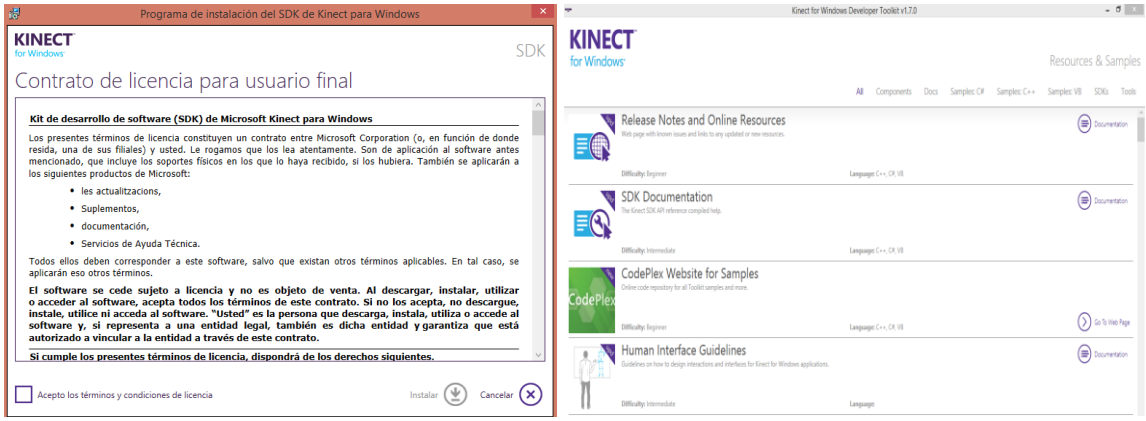

Figura 5.7 Instalador SDK Kinect / Kinect Development Kit.

5.3.1 Imágenes RGB y de profundidad

Como bien se prestableció en el diseño, se requería de la obtención de imágenes con un modelo a color en tiempo real del usuario, para ayudar con un efecto espejo que ayude al practicante a verse y enfocarse en sus posturas. Para ello se hizo de la utilización de las instrucciones incorporadas dentro del SDK de Kinect en Visual Studio, así como el agregado de 3 librerías en el cada uno de los archivos .cs del proyecto en el que se hace uso el sensor.

Para el agregado de las bibliotecas de Kinect es necesario añadir una referencia de ellas dentro del proyecto, por ello se busca el archivo Microsoft.Kinect dentro del apartado de *Agregar referencia*, tal como se indica en la Figura 5.8. Las siguientes referencias, que pueden ser de gran utilidad para el desarrollo de aplicaciones, deberán ser agregadas:

- Microsoft.Kinect.Toolkit.dll
- Microsoft.Kinect.Toolkit.Fusion.dll
- Microsoft.Kinect.Toolkit.Controls.dll
- Microsoft.Kinect.Toolkit.Interaction.dll

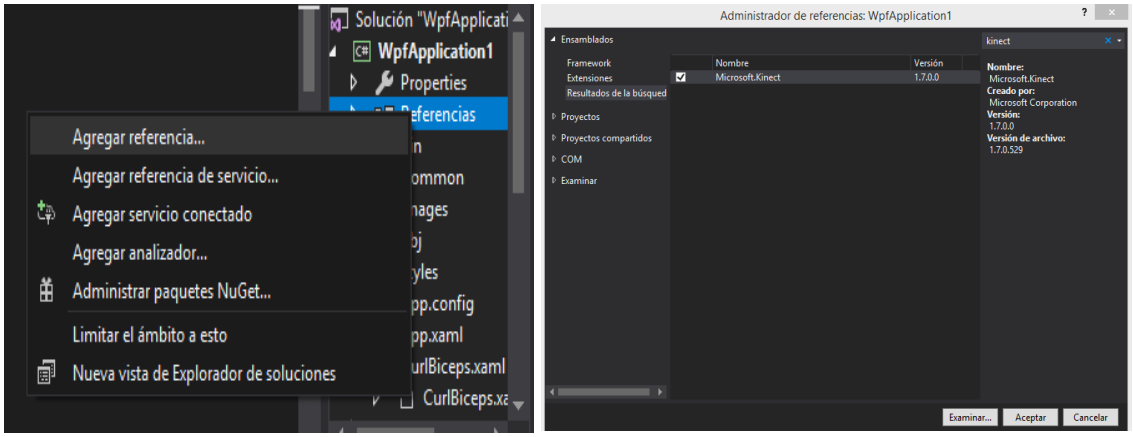

Figura 5.8 Agregado de referencias.

Las bibliotecas del que se hace uso son: Microsoft.Kinect, que permite la detección y uso del sensor, además contiene todas las clases, propiedades y métodos para manejar el sensor; Microsoft.Kinect.Toolkit, que incluye herramientas adicionales para el desarrollo de aplicaciones; mientras que Microsoft..Kinect.Toolkit.Controls implementa métodos para la manipulación de interfaces mediante gesticulación (Figura 5.9).

```
using Microsoft.Kinect;
using Microsoft.Kinect.Toolkit;
using Microsoft.Kinect.Toolkit.Controls;
```

Figura 5.9 Bibliotecas necesarias para el desarrollo con Kinect.

En el diseño gráfico (Figura 5.10), el archivo .xaml de la aplicación se agrega un control de tipo <<image>> dentro de nuestra área de trabajo, en el cual mostrará las imágenes que se recibirá de Kinect de la cámara RGB; otro de tipo <<canvas>> en la misma posición para el stream del esqueleto (Figura 5.11). Una sección para la cámara de profundidad, el cual no necesita de ningún objeto extra para su uso, sólo mediante su invocación del método en código y añadir su posicionamiento. Así como la sección de los botones que incluye la biblioteca de interacción entre ventanas y los contadores de repeticiones.

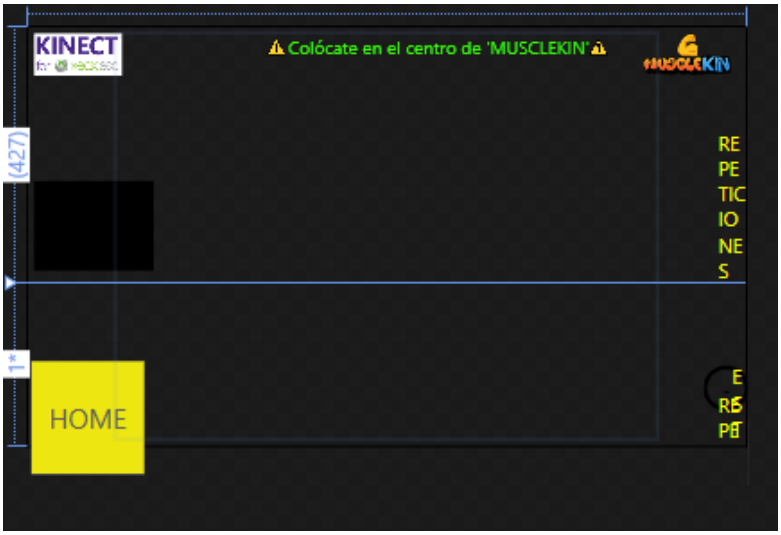

Figura 5.10 Diseño gráfico de proyecto "MuscleKIN".

```
<!--Grid para la imagenes-->
<Grid Grid.Row="0"  Margin="10 0 10 20">
    <!--<Image HorizontalAlignment="Left" VerticalAlignment="Top" Source="Images\kinect_logo.png" Stretch="None" Margin="0 10 0 4"/>-->
    <Image HorizontalAlignment="Right" VerticalAlignment="Top" Source="Images\musclekin.png" Stretch="None" Margin="0 0 0 4"/>
    <k:KinectUserViewer k:KinectRegion.KinectRegion="{Binding ElementName=kinectRegion}" Height="150" HorizontalAlignment="Left" VerticalAlignment="Bottom" ImageBackground
    <k:KinectSensorChooserUI HorizontalAlignment="Center" VerticalAlignment="Top" Name="sensorChooserUi" />
</Grid>
```

```
<!--ViewBox para la camara RGB y Skeleton Tracking-->
<Viewbox Stretch="Uniform" Margin="-1,12,1,13" Grid.RowSpan="2" >
    <Image Name="ventana" k:KinectRegion.KinectRegion="{Binding ElementName=kinectRegion}" Width="640" Height="480"/>
</Viewbox>
<Viewbox Stretch="Uniform" Margin="-1,12,1,13" Grid.RowSpan="2" >
    <Canvas Name="canvasEsqueleto" k:KinectRegion.KinectRegion="{Binding ElementName=kinectRegion}" Width="640" Height="480"></Canvas>
</Viewbox>
```

Figura 5.11 Archivo XAML de interfaz (Elementos para Streams).

Para interactuar con Kinect, se obtiene una instancia de la clase KinectSensor y se define su nombre; éste permitirá manipular las funciones del sensor. Así mismo, se crean variables que contendrán los valores de la cámara RGB y de la cámara de profundidad.

Para la manipulación del dispositivo se debe inicializar y/o detener la variable que contiene el Kinect que se utiliza con la instrucción Kinect.Start() / Kinect.Stop(); además se habilita el Stream.Enable() de ambas cámaras para dar inicio al envío de datos, teniendo en cuenta que son completamente independientes se debe llevar acabo el

mismo proceso para cada caso. Estas instrucciones deben ser incluidas dentro del evento *Loaded* de la aplicación que es necesario agregar al comienzo del proyecto para ejecutar el código al momento de iniciar la ejecución del proyecto (Figura 5.12).

Con el flujo de datos de los streams se permite la obtención de dicha información de manera constante, que es lo que se pretende con aplicaciones de este tipo, en tiempo real. Se puede configurar el formato de imagen que recibiremos, sin embargo, para el desarrollo de "MuscleKIN" se determinó la configuración por default que viene al llamar los métodos de habilitación, con la cámara a color obtiene información RGB con una resolución de 640x480 pixeles y a una velocidad de 30 fps.

```
private void Page_Loaded(object sender, RoutedEventArgs e)
{
    kinect = KinectSensor.KinectSensors.FirstOrDefault();
    kinect.Start();

    //Habilita los sensores o camaras de Kinect
    try
    {
        kinect.DepthStream.Enable();
        kinect.ColorStream.Enable();
        kinect.SkeletonStream.Enable();
    }
    catch
    {
        MessageBox.Show("Fallo al Inicializar Kinect", "Visor de Kinect");
        Application.Current.Shutdown();
    }
    kinect.ColorFrameReady += Kinect_ColorFrameReady;
    kinect.SkeletonFrameReady += Kinect_SkeletonFrameReady;
}
```

Figura 5.12 Código fuente con inicialización de Kinect y habilitación de stream de las cámaras y funcionalidades de "MuscleKIN".

En seguida, se indica qué es lo que tiene que hacer la aplicación con el flujo de datos que se recibe, y esto se hace mediante la creación de un controlador de eventos o EventHandler. Hay 3 métodos que de los que se puede hacer uso en aplicaciones: ColorFramerReady, DepthFrameReady y SkeletonFrameReady para el uso de la cámara de color, la cámara de profundidad y la detección del esqueleto humano respectivamente. Una vez creado estos ya es posible recibir datos de los Streams.

Dentro del EventHandler de cámara, se hace uso de la sintaxis *using*, que elimina el objeto de tipo de tipo ColorImageFrame; es decir, permite la toma constante de los frames enviados por Kinect. Se debe crear un arreglo de datos que recibirá esta información y se crea un mapa de bits que formará la imagen que captura Kinect (Figura 5.13).

```
//Cámara RGB
1 referencia
private void Kinect_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
{
    using (ColorImageFrame frameImagen = e.OpenColorImageFrame())
    {
        if (frameImagen == null) return;
        byte[] datosColor = new byte[frameImagen.PixelDataLength];
        frameImagen.CopyPixelDataTo(datosColor);

        ventana.Source = BitmapSource.Create(
            frameImagen.Width, frameImagen.Height,
            96,
            96,
            PixelFormats.Bgr32,
            null,
            datosColor,
            frameImagen.Width * frameImagen.BytesPerPixel
            );
    }
}
```

Figura 5.13 Código fuente programación de cámara RGB.

5.3.2 Skeleton Tracking

Kinect se desarrolló con una finalidad de adentrar al usuario dentro de ambientes virtuales; para ello se debía hacer la detección del cuerpo humano y cada una de sus articulaciones, para incorporarlo dentro de dichos ambientes, que se basan principalmente en la ubicación en 3 dimensiones. Por ello desarrollaron un algoritmo conocido como “Real-Time Human Pose Recognition in Parts from a Singles Depth Image" (Shotton, et al., 2011).

El algoritmo tiene como objetivo el reconocimiento esquelético de un individuo en tiempo real. Sus fundamentos básicos consisten en la captura de imágenes de profundidad con las cámaras y sensores con los que cuenta, se hace un procesamiento y una estimación de la distancia entre el sensor y los diversos objetos que se encuentran en

su campo de visión, en este caso sería el cuerpo humano, clasifica cada parte extremidad por secciones y en base a esto identifica una de las 20 articulaciones que es capaz de detectar (Figura 5.14).

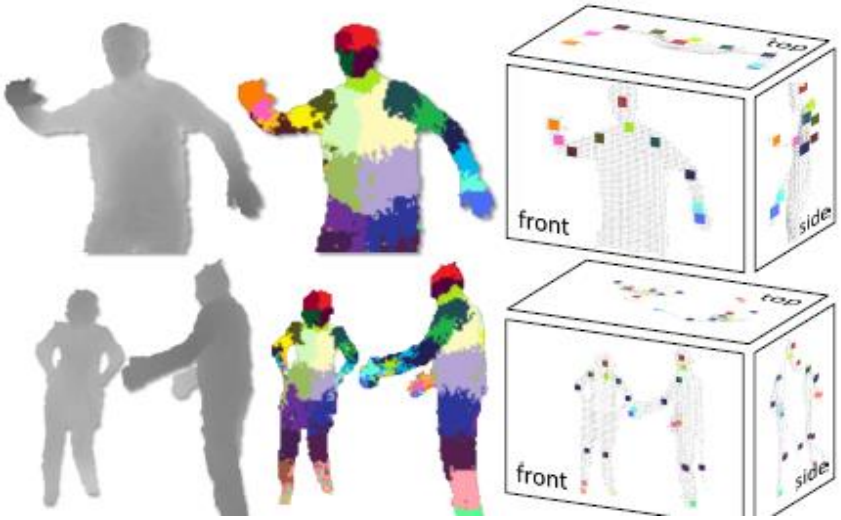

Figura 5.14 Principio de Algoritmo "Real-Time Human Pose Recognition in Parts from a Single Depth Image" (Shotton, et al., 2011).

La parte fundamental de MuscleKIN es la detección y seguimiento del esqueleto humano (también conocido como Skeleton Tracking) para comparar las posturas que presentan los usuarios con las posturas correctas preestablecidas y verificadas. Por ello se hizo uso de las instrucciones integradas en la suite de desarrollo de Kinect bajo la programación un nuevo algoritmo que fuera capaz de detectar la posición en 3 dimensiones de las 20 articulaciones que son posibles detectar con el dispositivo Kinect, mismas que se identifican en la Figura 5.15:

1. Cabeza
2. Hombro central
3. Hombro derecho
4. Hombro izquierdo
5. Codo derecho
6. Codo izquierdo
7. Muñeca derecha
8. Muñeca izquierda
9. Mano derecha
10. Mano izquierda
11. Columna
12. Cadera central
13. Cadera derecha
14. Cadera izquierda

15. Rodilla Derecha
16. Rodilla Izquierda
17. Tobillo Derecho
18. Tobillo Izquierdo
19. Pie derecho
20. Pie Izquierdo

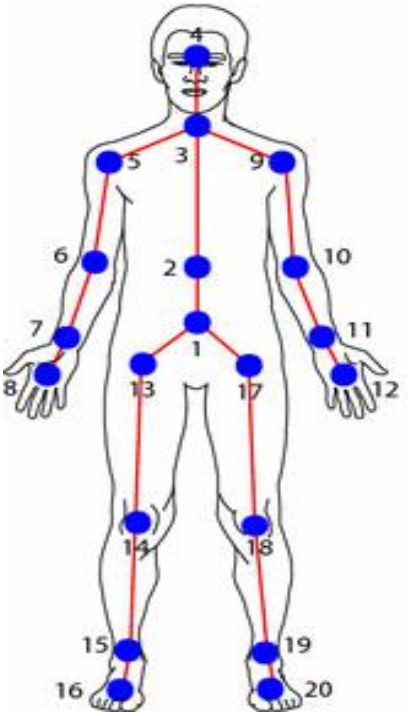

Figura 5.15 Articulaciones detectadas por Kinect.

Al igual que con la cámara RGB y la cámara de profundidad, se habilita e inicializa el Stream respectivo y se crea el controlador de eventos, dentro de ésta asigna una variable de tipo *Skeleton* que recibirá información constantemente por ello se debe asignar un valor de tipo *Array* (Figura 5.16); así mismo utilizando la sintaxis *using* se hace un escaneo de la detección del esqueleto y envía esa información de las articulaciones a una variable que posteriormente serán usadas para el algoritmo de comparación y para el dibujado del esqueleto del usuario dentro de la interfaz. Tiene una resolución de 640x480 pixeles y a una velocidad de 30 fps.

Posterior a esto se obtiene los valores de las articulaciones en el sistema de eje de 3 dimensiones, asignando variables independientes de tipo *Joint* para asignar a cada articulación, variables y obteniendo la información de la posición en el espacio mediante variables SkeletonPoint (Figura 5.17).

```
//Skeleton Stream
1 referencia
private void Kinect_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
    canvasEsqueleto.Children.Clear();
    Skeleton[] esqueletos = null;
    using (SkeletonFrame framesEsqueleto = e.OpenSkeletonFrame())
    {
        if (framesEsqueleto != null)
        {
            esqueletos = new Skeleton[framesEsqueleto.SkeletonArrayLength];
            framesEsqueleto.CopySkeletonDataTo(esqueletos);
        }
    }
    if (esqueletos == null) return;
    foreach (Skeleton esqueleto in esqueletos)
    {
        //Colisión de Bordes
        if (esqueleto.TrackingState == SkeletonTrackingState.Tracked)
        {
```

Figura 5.16 Código fuente: Skeleton Tracking.

```
//Variables tipo Joint para cada articulación
Joint jointCabeza = esqueleto.Joints[JointType.Head];
SkeletonPoint posicionCabeza = jointCabeza.Position;
Joint jointHombroCentro = esqueleto.Joints[JointType.ShoulderCenter];
SkeletonPoint posicionHombroCen = jointHombroCentro.Position;
Joint jointEspina = esqueleto.Joints[JointType.Spine];
SkeletonPoint posicionEspina = jointEspina.Position;
Joint jointCaderaCentro = esqueleto.Joints[JointType.HipCenter];
SkeletonPoint posicionCaderaCen = jointCaderaCentro.Position;
```

```
//Propiedades de Línea
Line huesoCabeza = new Line();
huesoCabeza.Stroke = new SolidColorBrush(Colors.Green);
huesoCabeza.StrokeThickness = 5;
Line huesoEspina = new Line();
huesoEspina.Stroke = new SolidColorBrush(Colors.Green);
huesoEspina.StrokeThickness = 5;
Line huesoEspinaAbajo = new Line();
huesoEspinaAbajo.Stroke = new SolidColorBrush(Colors.Green);
huesoEspinaAbajo.StrokeThickness = 5;
Line huesoHombroCen = new Line();
huesoHombroCen.Stroke = new SolidColorBrush(Colors.Green);
huesoHombroCen.StrokeThickness = 5;
```

Figura 5.17 Código fuente: Variables por articulación / propiedades de dibujado de líneas.

Se lleva a cabo la unión de cada articulación con trazos de líneas entre cada una de ellas y plasmándolas en una región del archivo XAML en un elemento de tipo <<Canvas>> al igual que se hizo con la cámara RGB (véase Figura 5.18), la posición de dichos elementos se encuentran dentro del mismo GRID debido a que supondrá esta parte del sistema la más importante y visual, en donde el usuario final verá su imagen y la de su esqueleto (Figura 5.19).

```
//Dibujado de puntos
//Manos
ColorImagePoint puntoManoDer = kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointManoDerecha.Position, ColorImageFormat.RgbResolution640x480Fps30);
huesoManoDer.X1 = puntoManoDer.X;
huesoManoDer.Y1 = puntoManoDer.Y;
ColorImagePoint puntoMunecaDer = kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointMunecaDerecha.Position, ColorImageFormat.RgbResolution640x480Fps30);
huesoManoDer.X2 = puntoMunecaDer.X;
huesoManoDer.Y2 = puntoMunecaDer.Y;
ColorImagePoint puntoManoIzq = kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointManoIzquierda.Position, ColorImageFormat.RgbResolution640x480Fps30);
huesoManoIzq.X1 = puntoManoIzq.X;
huesoManoIzq.Y1 = puntoManoIzq.Y;
ColorImagePoint puntoMunecaIzq = kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointMunecaIzquierda.Position, ColorImageFormat.RgbResolution640x480Fps30);
huesoManoIzq.X2 = puntoMunecaIzq.X;
huesoManoIzq.Y2 = puntoMunecaIzq.Y;
canvasEsqueleto.Children.Add(huesoManoDer);
canvasEsqueleto.Children.Add(huesoManoIzq);
```

Figura 5.18 Unión de Articulaciones mediante líneas.

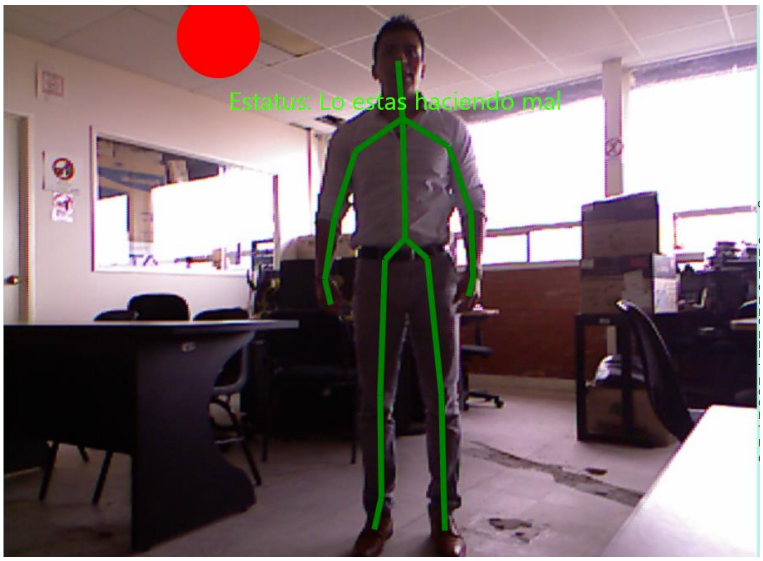

Figura 5.19 Skeleton Tracking.

Cabe mencionar los problemas que se presentan por factores externos en el ambiente o en el usuario que es capaz de afectar la captura de la imagen o el desempeño de Kinect a la hora de detectar el esqueleto humano, así como la limitante de detección de 5 cuerpos en una misma escena y dos esqueletos propiamente sin presentar falla alguna, estos factores externos pueden ser (Figura 5.20):

- La detección de sombras o de ambientes muy poco iluminados.
- Las variaciones en la iluminación de la escena.
- Las perturbaciones agregadas por el dispositivo a usarse.

- La detección de objetos grandes o fuera de su alcance.
- Debe haber una distancia ideal de 2 m hacia el objeto y mínima de 1 m de altura.
- Un fondo no estático o con movimiento, en este caso, el sensor ignora este hecho, puesto que sólo detecta figuras humanas, siendo hasta 5 personas en la misma escena y sólo dos esqueletos completos serán detectados sin presentar fallas.
- Movimientos de la cámara.
- El uso de ropa holgada, puede ser también una dificultad para el seguimiento del cuerpo humano (Body Tracking).

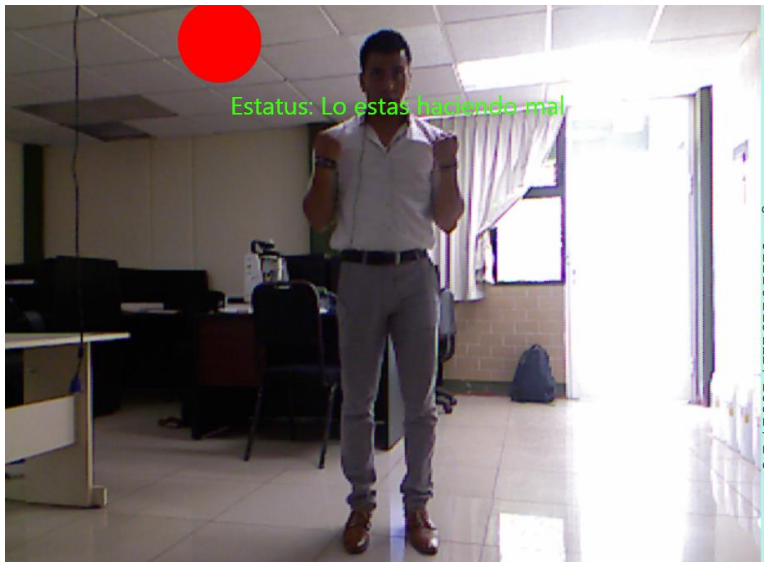

Figura 5.20. Error de detección de esqueleto por iluminación.

Así mismo es preciso mencionar que los elementos que contienen el área donde se muestran las funcionalidades del Skeleton Tracking, así como de la cámara RGB y del sensor de profundidad, y de las otras herramientas finales del sistema como ambos contadores se encuentran dentro una región debido a que la toma constante de información supondrá una saturación de memoria severa que terminará en el colapso del programa como se muestra en la Figura 5.22 el cual muestra la memoria utilizada por el sistema en su 3era y 4ta iteración, por lo cual se ubica dentro de ésta con la finalidad de limpiar la región en cada iteración (Figura 5.21).

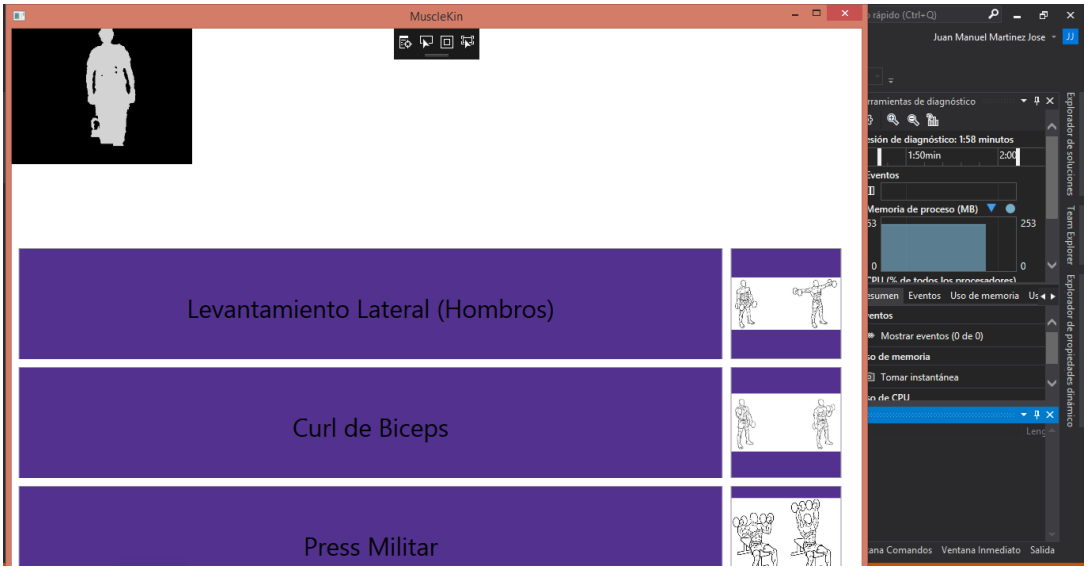

Figura 5.21 Interfaz e información de memoria de proceso inicial.

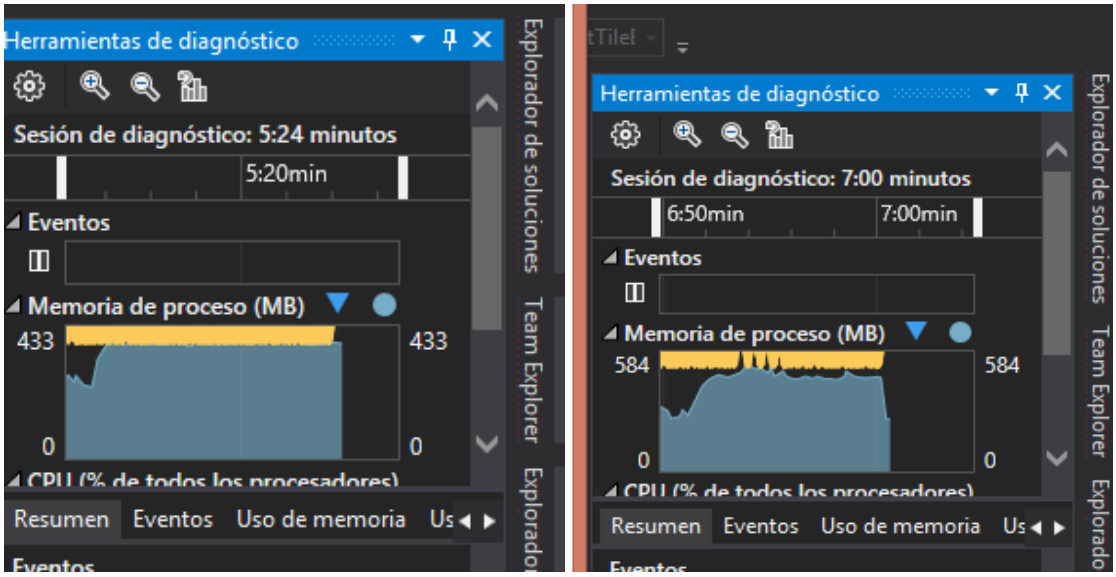

Figura 5.22 Memoria de proceso: 3ra y 4ta iteración.

Lo primero que se realizó paras solucionar dicho problema es mandar a llamar el Colector de Basura: GC (Garbage Collector, por sus siglas en inglés) de manera manual, que interviene cada cierto tiempo para la limpieza de basura parcial y así evitar una colección de basura completa. Su objetivo es proporcionar una capa de abstracción de manejo de memoria para: pérdida de memoria, fragmentación de memoria, o corrupción.

Para posteriormente llamar la clase BindingOperations.ClearBinding() cuya finalidad es quitar cualquier enlace de una propiedad si existe una. Además, añadido el método ClearBinding quita la expresión de enlace y restaura el valor de las propiedades que se encuentren dentro del objeto especificado (Figura 5.23). En este caso de todos los objetos de la región Kinect, el cual contiene todos los objetos que muestra el contenido multimedia de las cámaras, sensores y de la información que se recibe de la acción del usuario (Skeleton Tracking, contadores, imágenes).

```
private void KinectTileButton_Click_3(object sender, RoutedEventArgs e)
{
    GC.Collect();
    BindingOperations.ClearBinding(this.kinectRegion, KinectRegion.KinectSensorProperty);
    this.sensorChooser.KinectChanged -= SensorChooserOnKinectChanged;
    (Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source = new Uri("MainMenu.xaml", UriKind.Relative);
}
```

Figura 5.23 Código fuente para liberación de memoria.

5.3.3 Algoritmo de Comparación de Posturas en Ejercicios

El desarrollo del algoritmo para la comparación de posturas se basó en la información que se obtiene a través de los sensores de profundidad que contiene Kinect, y mediante las variables de tipo *Joint* y *SkeletonPoint* del que previamente se habían establecido para el mapeado del esqueleto en tiempo real (véase Figura 5.24, Figura 5.25). Esta información se obtiene en 3D por lo que se debe tener en cuenta el modelo de gesticulación y de detección del esqueleto humano en cual se mestra en la Figura 5.26.

```
//Variables tipo Joint para cada articulación
Joint jointCabeza = esqueleto.Joints[JointType.Head];
SkeletonPoint posicionCabeza = jointCabeza.Position;
Joint jointHombroCentro = esqueleto.Joints[JointType.ShoulderCenter];
SkeletonPoint posicionHombroCen = jointHombroCentro.Position;
Joint jointEspina = esqueleto.Joints[JointType.Spine];
SkeletonPoint posicionEspina = jointEspina.Position;
Joint jointCaderaCentro = esqueleto.Joints[JointType.HipCenter];
SkeletonPoint posicionCaderaCen = jointCaderaCentro.Position;
```

Figura 5.24 Código fuente para la obtención de posición en 3D de las articulaciones.

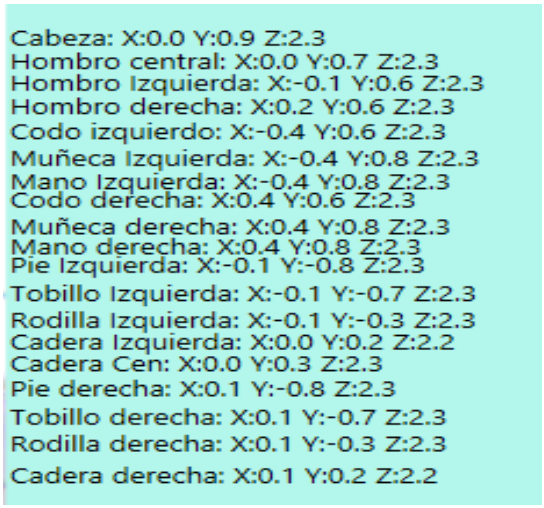

Figura 5.25 Valores de posición de las articulaciones detectadas por Kinect, obtenidas en coordenadas de 3 dimensiones.

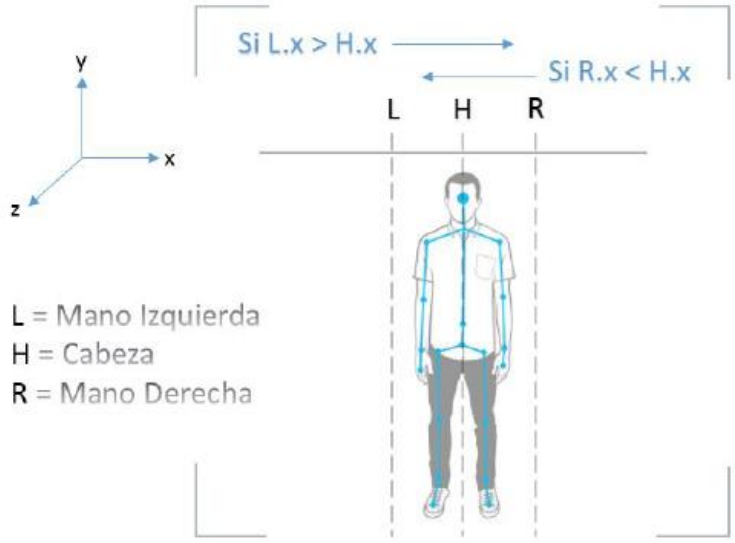

Figura 5.26 Modelo de gesticulación y detección de esqueleto Kinect.

Se procedió a realizar una sesión de captura de movimiento y posición con el apoyo del instructor certificado en el área de pesas Daniel Lovera Martínez, mediante las instrucciones mencionadas con anterioridad y la información proporcionada tanto del instructor como de la literatura especializada, se tomó la captura en dos posiciones principales, una para la postura inicial e ideal para la realización de los ejercicios,

incluyendo medidas precisas de cada parte y articulación del cuerpo, así como de la postura correcta que debe alcanzar el usuario final para realizar el ejercicio en cuestión de manera correcta. Además de una serie de posiciones secundarias las cuales indican las posturas incorrectas del usuario; la finalidad de la toma de estas posturas es bajar la tasa de error por parte del dispositivo Kinect ante los problemas de entorno que ya fueron mencionados y una mejor precisión a la hora de enviar el indicativo visual que precise si se realizó el ejercicio correctamente o no.

Toda esta información de la posición de las articulaciones en 3 dimensiones es incluida dentro de 4 condicionales de tipo *if*, los cuales contienen dentro las condiciones la postura inicial, la postura correcta, las posturas incorrectas con sus respectivos indicativos visuales que proveen al sistema "MuscleKIN" un fácil entendimiento al momento de uso final; y una última condicional que provee información de las repeticiones del ejercicio que realiza el fisicoculturista (contador automático) e indica si se ha llegado al número de repeticiones prestablecidas (Figura 5.27).

```
//Boolean bandera = true;
if (posicionHombroDer.X <= 0.4 && posicionHombroDer.X >= 0.0 && posicionHombroDer.Y <= 0.7 && posicionHombroDer.Y >= 0.3 &&
    posicionMunecaDer.X <= 0.7 && posicionMunecaDer.X >= 0.5 && posicionMunecaDer.Y <= 0.7 && posicionMunecaDer.Y >= 0.5 &&
    posicionHombroIzq.X <= 0.1 && posicionHombroIzq.X >= -0.3 && posicionHombroIzq.Y <= 0.8 && posicionHombroIzq.Y >= 0.4 &&
    posicionMunecaIzq.X <= -0.3 && posicionMunecaIzq.X >= -0.6 && posicionMunecaIzq.Y <= 0.7 && posicionMunecaIzq.Y >= 0.5)
{
    contador = int.Parse(textBlockContador.Text);
    mensajeContador = contador.ToString();
    bien.Visibility = Visibility.Visible;
    mal.Visibility = Visibility.Hidden;
    inicial.Visibility = Visibility.Hidden;

    if (bien.Visibility == Visibility.Visible)
    {
        i++;
        int milliseconds = 700;
        mensajeContadorAutomatico = i.ToString();
        textBlockContadorAutomatico.Text = mensajeContadorAutomatico;
        Thread.Sleep(milliseconds);
        if (mensajeContadorAutomatico == mensajeContador)
        {
            BindingOperations.ClearBinding(this.kinectRegion, KinectRegion.KinectSensorProperty);
            (Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source = new Uri("Terminado.xaml", UriKind.Relative);
        }
    }
}
else
{
    mal.Visibility = Visibility.Visible;
    bien.Visibility = Visibility.Hidden;
    inicial.Visibility = Visibility.Hidden;
}
```

Figura 5.27 Código fuente Algoritmo de comparación de posturas.

Todo se realizó manteniendo una tolerancia de 5 grados de flexión (físicamente hablando) para considerar una repetición como correcta en cada condicional de postura correcta, esta tolerancia se tomó en base a la información recabada en estado del arte y por la recomendada por el instructor que apoyó en la captura de movimiento (Figura 5.28).

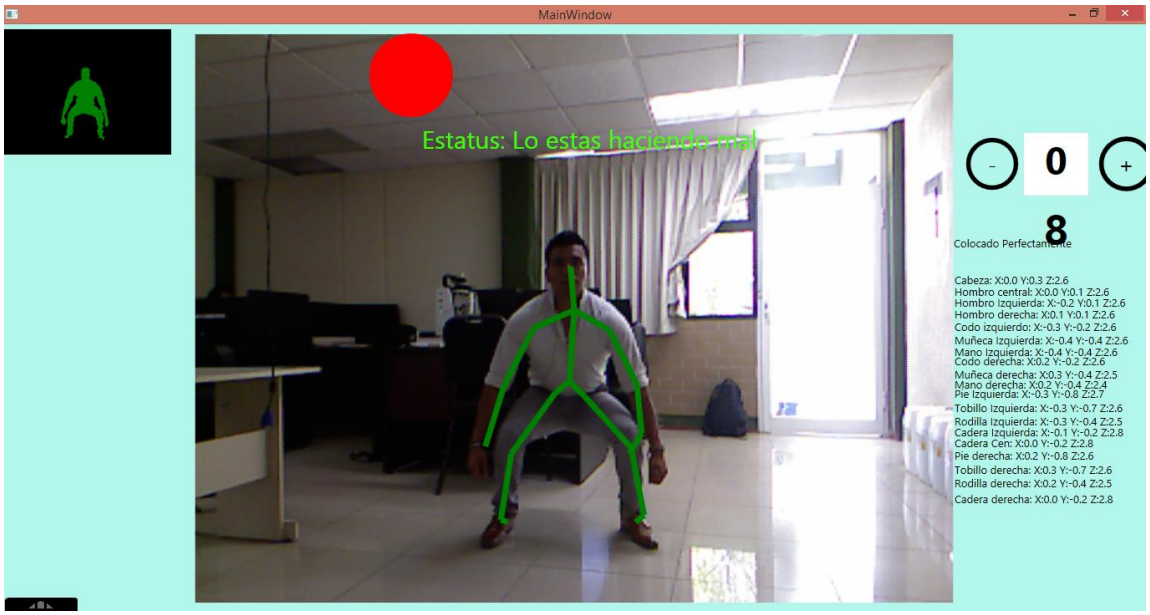

Figura 5.28 Ventana de usuario para la corrección de posturas (Postura incorrecta).

5.3.4 Interfaz Natural de Usuario

El planteamiento inicial del sistema "MuscleKIN" es la autonomía del mismo en diversos sentidos, la no intervención de factores externos, primordialmente los que tienen que ver con el contacto físico entre el usuario final y cualquier componente (monitor, sensor Kinect, computadora y periféricos), además de la calibración automática ante las diversas condiciones del ambiente (Dentro del rango permitido: Iluminación natural y/o iluminación artificial), así como de los grados requeridos para su posición inicial (0º la posición del motor de Kinect y a una altura de 1.20 m en ubicación). Por eso se desarrolló una Interfaz Natural de Usuario (NUI), el cual es un término usado para referirse a una interfaz de usuario que es efectivamente invisible con las interacciones que realiza el usuario.

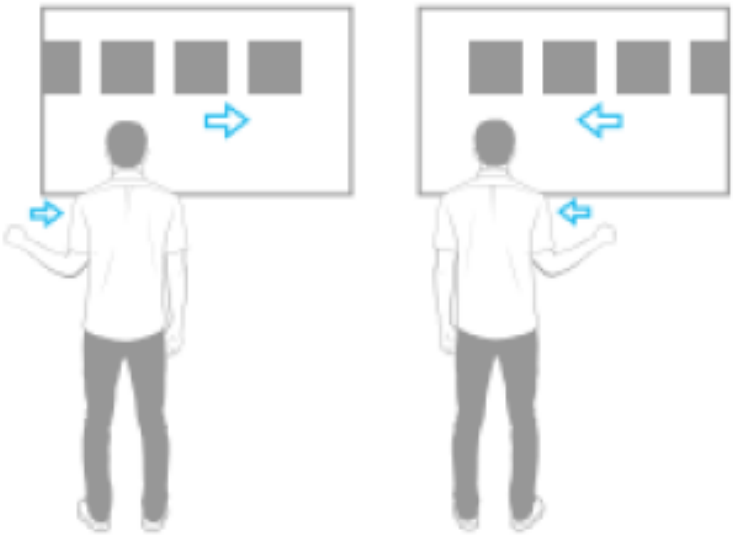

Figura 5.29 Control de Interfaz mediante gesticulación.

Se diseñó la interfaz con la que contaría el sistema, el cual en primera instancia incluye un menú principal en donde se alojan el catálogo de ejercicios, cada uno con dos botones independientes, pero con un desplazamiento en conjunto cuando se hace navegación a través de la interfaz, los cuales tendrían la función de abrir las nuevas ventanas: La ventana de ejecución del ejercicio y la ventana de apartado de información para la realización (Figura 5.30).

La primera de estas tendría en su contenido la ventana en la cual se encuentra el objetivo principal de este proyecto, es decir el apartado de corrección de posturas. Dentro de estas se incluye los streams de la cámara RGB, de profundidad, y del Skelton Tracking, así como de las herramientas adicionales como el establecimiento de repeticiones y el contador automático (Figura 5.31).

Por otro lado, la segunda ventana contiene información en texto de la realización del ejercicio y de forma adicional la imagen descriptiva de ejecución (Figura 5.32).

Figura 5.30 UI Menú principal.

Figura 5.31 UI Ventana de Correción de posturas

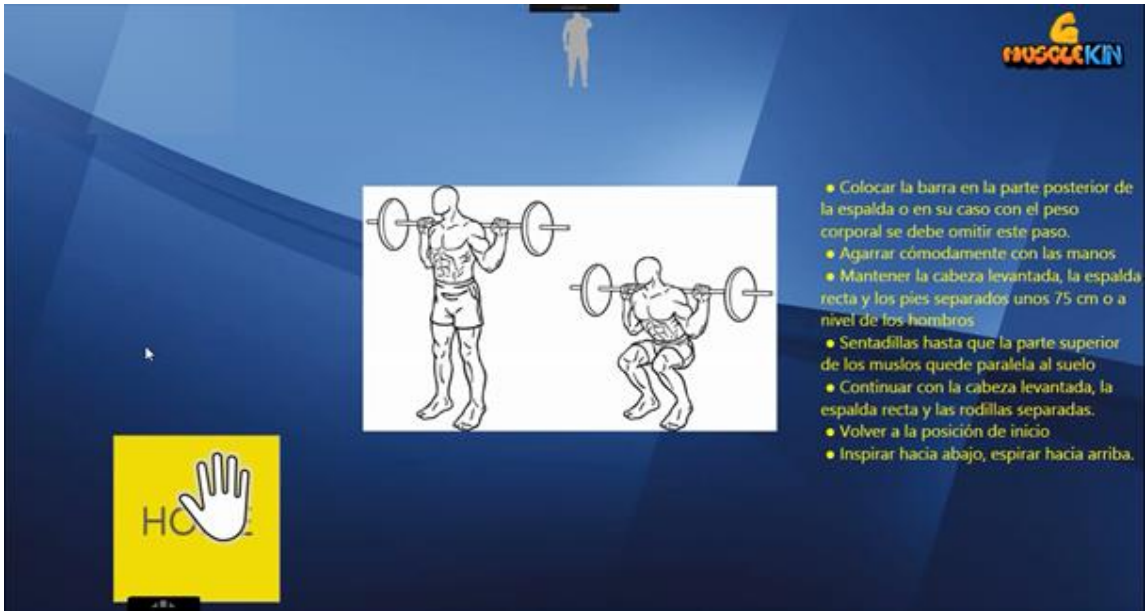

Figura 5.32 UI Ventana de información.

Para este apartado que comprende el sistema se agregó el namesapace de los controles externos del Toolkit de Kinect al proyecto (veáse Figura 5.33), que permite hacer uso de elementos ya prediseñados como:

- <KinectSensorViewer>: Permite verificar si hay algún sensor Kinect conectado y lo verifica en tiempo real.
- <KinectSensorChooserUI>: Permite definir una zona para usar la gesticulación de las manos como cursor.
- <KinectUserViewer>: No permite observar los usuarios que están en la visión del sensor, así como el usuario que está haciendo uso de la función de cursor.
- <KinectTileButton>: Botón cuadrado para uso especial de la función cursor.
- <KinectCircleButton>: Botón redondo para uso especial de la function cursor .
- <KinectScrollViewer>: Crear una zona para poder navegar con el cursor, es un scroll específicamente diseñada para cuando se tienen más elementos de los que caben una la ventana, pudiendo ser manipulada mediante la gesticulación.

```
<Page x:Class="WpfApplication1.LevantLat"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
            xmlns:local="clr-namespace:Microsoft.Samples.Kinect.ControlsBasics"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:k="http://schemas.microsoft.com/kinect/2013"
        xmlns:tk="clr-namespace:Microsoft.Kinect.Toolkit;assembly=Microsoft.Kinect.Toolkit"
```

Figura 5.33 Añadimiento de librería para control mediante gesticulación.

5.3.5 Colisión en Bordes

Una parte muy esencial del sistema es de la colisión del esqueleto humano con los bordes del <<canvas>> donde se ubica la información que proporciona Kinect, se comprendió este apartado debido a la utilización incorrecta que pueda suscitarse en un usuario final, al ubicarse dentro de los rangos incorrectos de visión del sensor Kinect. Por ello de desarrollo un algoritmo que hace el seguimiento del esqueleto constantemente, y mediante condicionales, así como de las propiedades que incluyen las librerías de Kinect permite dar indicativos visuales de colisión (Figura 5.34 y Figura 5.35).

```
foreach (Skeleton esqueleto in esqueletos)
{
    //Colisión de Bordes
    if (esqueleto.TrackingState == SkeletonTrackingState.Tracked)
    {

        status1.Visibility = Visibility.Hidden;
        Advertencia.Visibility = Visibility.Hidden;
        Advertencia2.Visibility = Visibility.Hidden;
        Abajo.Visibility = Visibility.Hidden;
        Arriba.Visibility = Visibility.Hidden;
        Izquierda.Visibility = Visibility.Hidden;
        Derecha.Visibility = Visibility.Hidden;

        if ((esqueleto.ClippedEdges & FrameEdges.Bottom) != 0)
            Abajo.Visibility = Visibility.Visible;
        if ((esqueleto.ClippedEdges & FrameEdges.Top) != 0)
            Arriba.Visibility = Visibility.Visible;
        if ((esqueleto.ClippedEdges & FrameEdges.Right) != 0)
            Derecha.Visibility = Visibility.Visible;
        if ((esqueleto.ClippedEdges & FrameEdges.Left) != 0)
            Izquierda.Visibility = Visibility.Visible;
```

Figura 5.34 código fuente colisión de bordes.

Figura 5.35 Interfaz indicando la colisión de esqueleto con borde superior.

6 RESULTADOS

De cara a realizar la valoración y funcionamiento del sistema se llevaron a cabo 8 pruebas a lo largo de todo el proceso de desarrollo en diversos entornos ambientales.

En el Centro Universitario UAEM Atlacomulco:

- Centro de Autoacceso
- Laboratorio de Maestría en Ciencias de la Computación

En el gimnasio público “Total GYM:

- Salón de usos múltiples
- Área de pesas

Las pruebas se realizaron con distintas condiciones ambientales con especial énfasis en la iluminación, además de la ropa que se utilizó, distancias y altura en la posición del sensor Kinect.

Por otra parte, la prueba final del sistema se llevó a cabo en un ambiente real (área de pesas del gimnasio “Total GYM”) con la participación del instructor certificado por la Federación Mexicana de Fisicoconstructivismo y Fitness A.C. Daniel Lovera Martínez, además de algunos socios de los diversos niveles de fisicoculturistas (principiantes, intermedios y avanzados). Cabe mencionar que para dicha prueba se hizo una calibración del sensor en altura, posicionamiento e iluminación (artificial).

Antes de la instalación de “MuscleKIN” es necesario precisar que se debe contar con un sistema operativo Windows de 64 bits; para iniciar el proceso únicamente se debe ejecutar el archivo .msi, el cual abrirá el asistente automático de instalación (véase Figura 6.1).

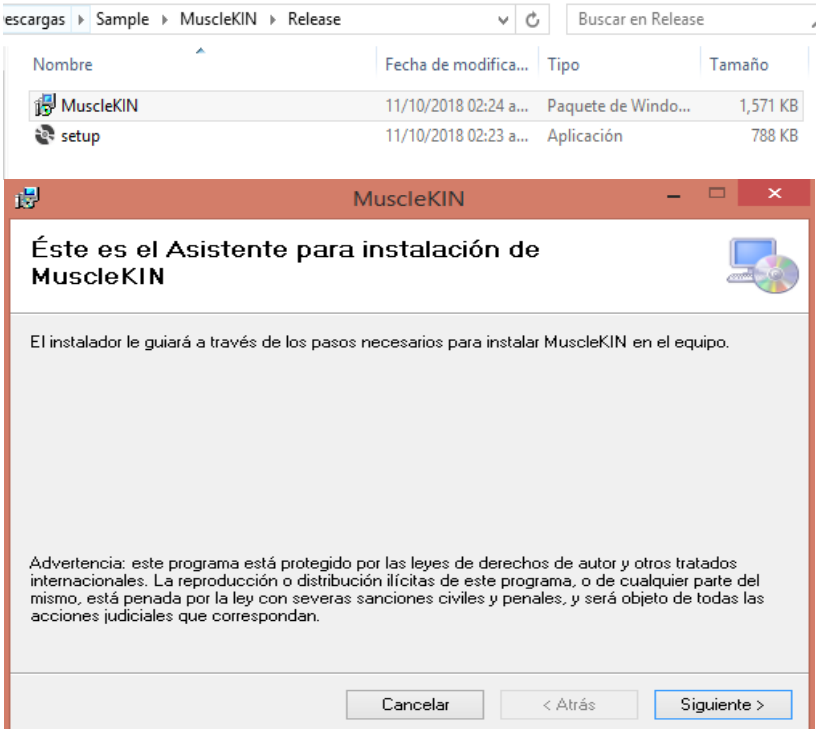

Figura 6.1 Ejecutable/ Instalador automático de "MuscleKIN".

Una vez dado autorización para su instalación, éste solicita la ruta de acceso de la carpeta en la que se guardaran los archivos como se muestra en la Figura 6.2, seguido de esto la confirmación del proceso por parte del usuario (véase Figura 6.3).

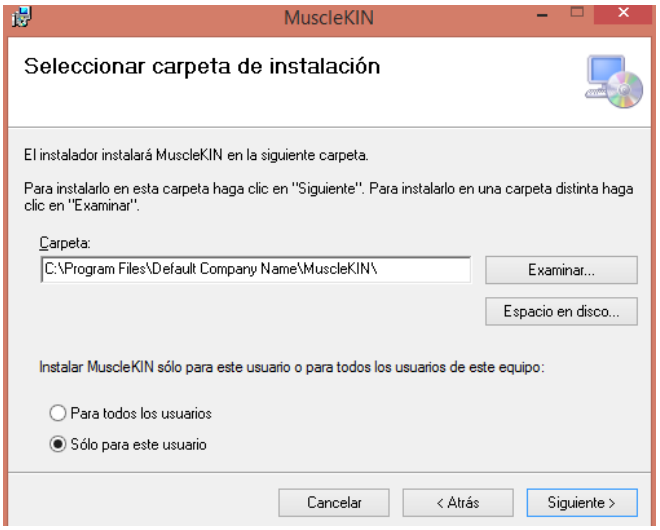

Figura 6.2 Establecimiento de Carpeta de Instalación.

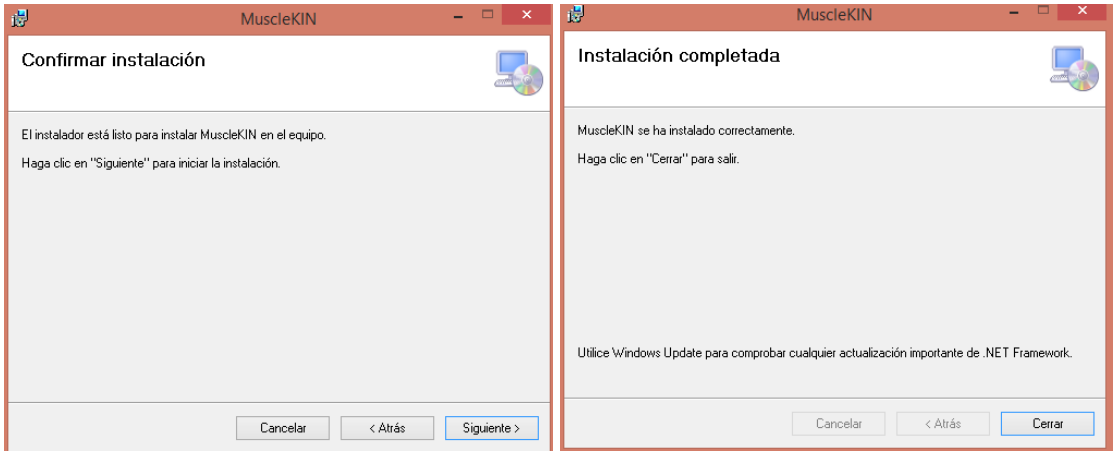

Figura 6.3 Confirmación de Instalación.

Para el inicio del sistema únicamente se debe ejecutar el acceso directo creado después de la instalación tal como se muestra en la Figura 6.4. o bien ejecutarlo mediante el buscador.

Figura 6.4 Acceso directo de "MuscleKIN".

Al momento de la ejecución de "MuscleKIN", éste indica si el equipo cuenta con las características específicas en resolución, caso contrario éste proporcionara una opción para su ejecución óptima. En la Figura 6.5 se muestra la ventana principal del sistema que se visualiza el usuario en primera instancia, el cual consiste en un menú desplegable

con los distintos ejercicios incluidos para la corrección de posturas, además de una sección secundaria de información para la realización de los mismos (véase Figura 6.6).

Figura 6.5 Ventana principal "MuscleKIN" (Menú).

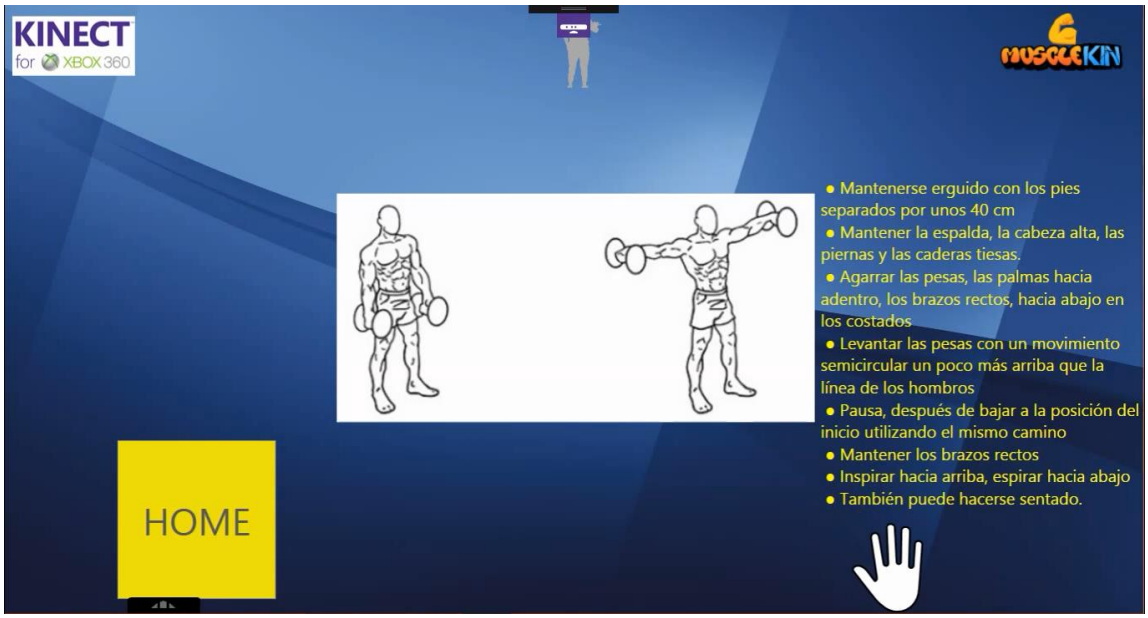

Figura 6.6 Ventana de información para la realización de ejercicio: Levantamientos laterales.

Como primer punto se observó que el sistema “MuscleKIN”, mediante su Interfaz Natural de Usuario, resultó efectivamente intuitivo como se tenía previsto, el usuario siempre tuvo presente sin necesidad de instrucciones que sus manos eran el control del sistema, prescindiendo el uso de mandos o algún tipo de periférico (véase Figura 6.7). Sin embargo, se obtuvieron variaciones en la detección de gestos dependiendo del ambiente en que se realizaron las pruebas: mayor iluminación era equivalente a una mejor detección de las gesticulaciones por parte del sensor Kinect.

Figura 6.7 Interacción USUARIO-NUI.

En cuanto a la corrección de las posturas se hizo una comparación de la ejecución acertada de cada ejercicio previamente establecidas en la sesión de captura de movimiento y contenidas dentro del sistema, con las presentadas por los usuarios (siendo un total de 10 participantes) de los distintos niveles de fisicoculturistas: principiantes, intermedios y avanzados; es preciso mencionar que dichos usuarios presentan una estatura variada entre el 140-175 cm., ubicados a 2 m. de distancia con el sistema que tiene al sensor Kinect ubicado a 120 cm de altura.

En las demostraciones previas a la final, que fueron realizadas en el Centro Universitario UAEM Atlacomulco y en el área de usos múltiples del gimnasio “Total GYM” la variación en la efectividad del sistema varió, específicamente por la iluminación precaria o excesiva, así como el tipo de ropa con el que se realizaban las pruebas (chamarras o ropa muy holgada).

En la Figura 6.8 y Figura 6.9 se muestra la comparativa de las posturas en el ejercicio de levantamiento lateral. En la primera imagen se muestra la postura inicial correcta prestablecida en “MuscleKIN”, y en la segunda imagen la realizada por el usuario, el cual tiene que obtener el incentivo visual por parte del sistema que demuestra que se mantiene en esa posición (Color amarillo).

Figura 6.8 Pruebas finales. Ejercicio: Levantamiento lateral (posición de referencia).

Figura 6.9 Pruebas finales. Ejercicio: Levantamiento lateral (Posición inicial correcta).

En segunda instancia se muestra la imagen de la interfaz con el recorrido o postura incorrecta en la ejecución del ejercicio, además de su respectivo indicativo visual que demuestra dicho estatus (Color rojo), tal como se indica en la Figura 6.10. En este caso, el sistema indica que el ejercicio parte de una posición inicial incorrecta, por lo que las repeticiones no se incrementan, y el usuario deberá comenzar nuevamente la repetición en curso, hasta aprender el punto de origen del ejercicio., así como el recorrido de los brazos para realizarlo adecuadamente.

De manera semejante, en la Figura 6.11 se aprecia la posición correcta de finalización del ejercicio, donde la postura que logre el usuario (Figura 6.12) debe coincidir con la posición correcta sugerida por el sistema. Si el usuario no completa satisfactoriamente el recorrido del movimiento de los brazos, entonces el sistema indicará un error, y la repetición no se contabilizará.

Figura 6.10 Pruebas finales. Ejercicio: Levantamiento lateral (Posición incorrecta).

Figura 6.11 Pruebas finales. Ejercicio: Levantamiento lateral (Posición correcta de finalización de referencia).

Figura 6.12 Pruebas finales. Ejercicio: Levantamiento lateral (Posición correcta de finalización).

De manera semejante, se realiza para el resto de los ejercicios especificados en este documento, por ejemplo, para la sentadilla, que se muestra en la Figura 6.13 y Figura 6.14, que ilustran la posición inicial correcta y la incorrecta, respectivamente.

Figura 6.13 Pruebas finales. Ejercicio: Sentadilla (Posición inicial / Posición correcta).

Figura 6.14 Pruebas finales. Ejercicio: Sentadilla (Posición inicial / Posición incorrecta).

Finalmente, se muestra la interfaz con la realización de la correcta practica de uno de los ejercicios, manteniendo los grados de flexión permitidas dentro del rango especificado dentro de los requisitos del sistema (5º, ±2º arriba y abajo de la posición en cada articulación involucrada con respecto a las posturas obtenidas en la captura de movimiento). Esto se realiza en la pantalla con un indicativo visual en verde que demuestra al usuario, en tiempo real, que realiza de manera precisa el ejercicio, siendo acompañada con el aumento en el contador automático de repeticiones.

Por ejemplo, en la Figura 6.15 se muestra un ejercicio donde el usuario parte de la posición inicial correcta. Sin embargo, durante la trayectoria, si el movimiento que detecta el sistema queda fuera de la tolerancia establecida. Entonces se indicará una alerta indicando que debe corregir el ejercicio (Figura 6.16), hasta que el usuario vuelva a posicionarse dentro de las especificaciones propias de cada ejercicio (Figura 6.17).

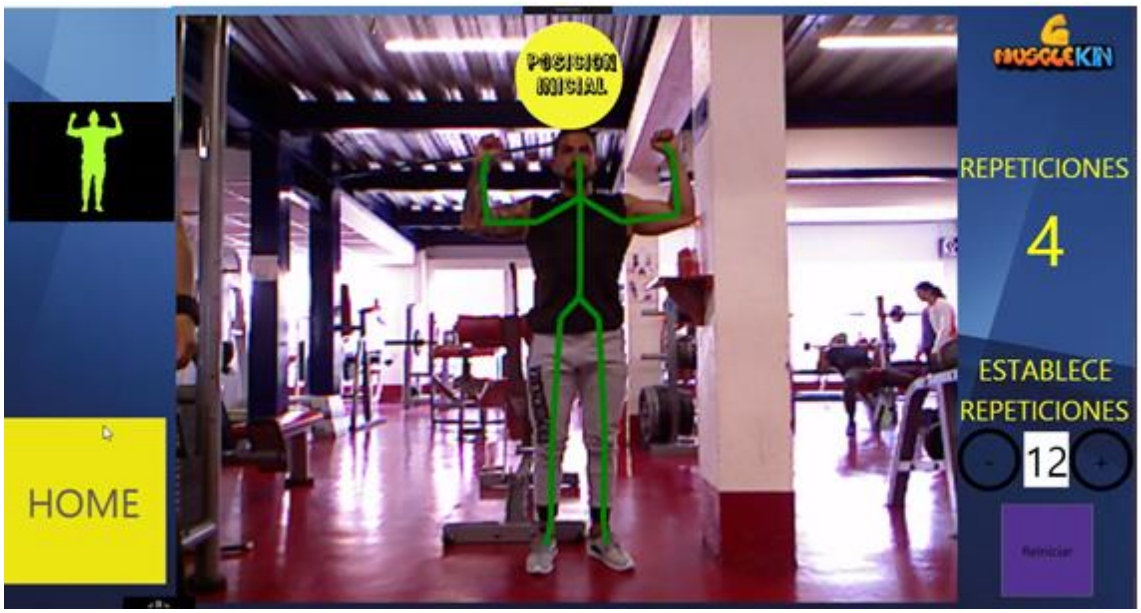

Figura 6.15 Ventana de usuario para la corrección de posturas (postura inicial correcta).

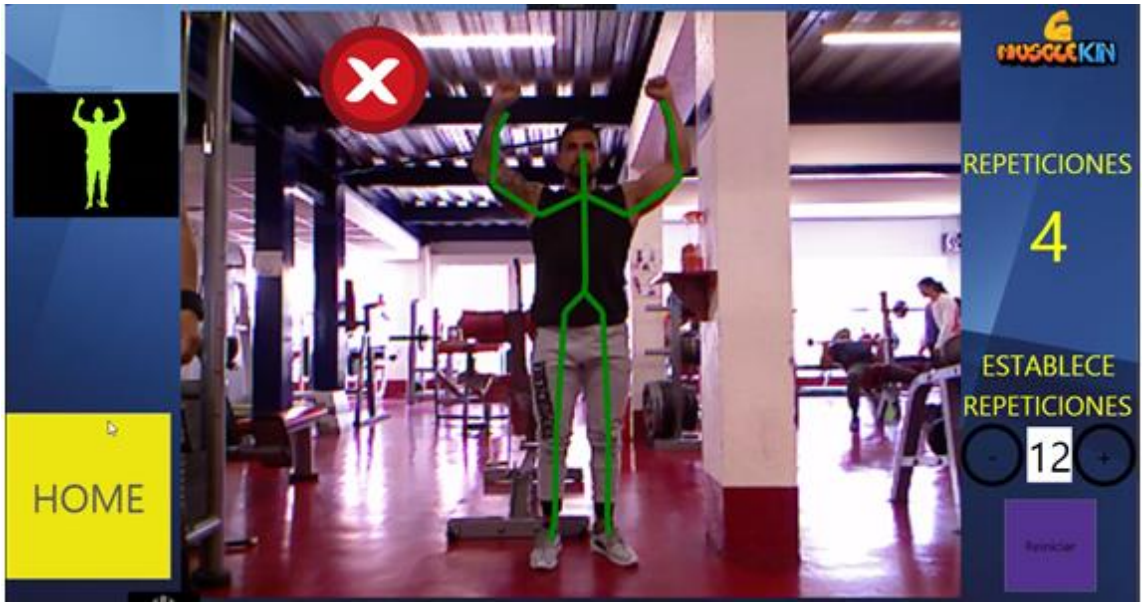

Figura 6.16 Ventana de usuario para la corrección de posturas (Postura incorrecta).

Figura 6.17 Ventana de usuario para la corrección de posturas (Postura correcta).

Con la finalidad de evaluar el desempeño del sistema en distintos escenarios, se presenta el gráfico de la Figura 6.18, donde se indican los porcentajes de eficacia en la detección de posturas correctas para todos los ejercicios contenidos en el sistema que se presentaron a lo largo de las 8 pruebas que se realizaron para la comprobación de "MuscleKIN".

Las pruebas, como ya se mencionó, se realizaron en 4 ambientes diferentes, a distintos tipos de iluminación los cuales se contienen en la gráfica; abarcando unos de los objetivos específicos del proyecto de tesis, el cual precisaba obtener la calidad del sistema ante distintos entornos.

- Ambiente poco iluminado: Sala de usos múltiples del gimnasio "Total GYM".
- Ambiente poco iluminado artificialmente: Laboratorio de Maestría en Ciencias de la computación del CU UAEM Atlacomulco.
- Ambiente con iluminación natural: Centro de Auto acceso CU UAEM Atlacomulco (CAA).
- Ambiente bien iluminado natural y artificialmente: Área de pesas del gimnasio "Total GYM".

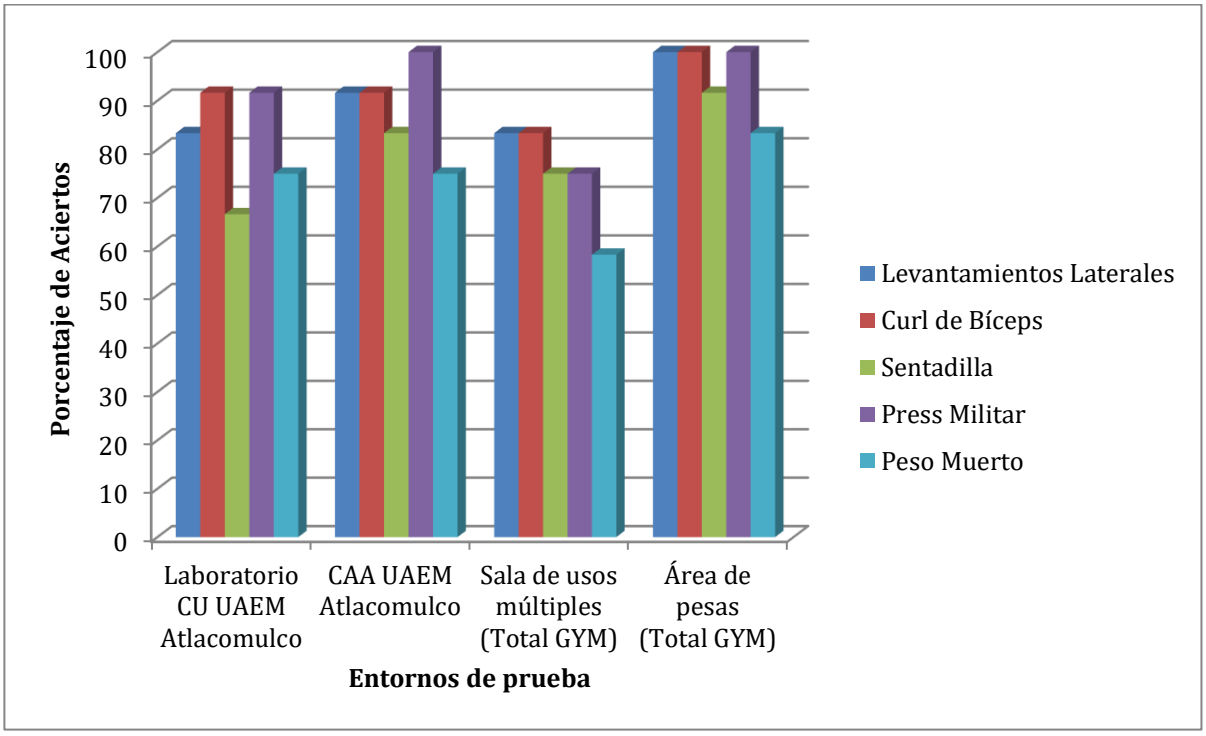

Figura 6.18 Porcentaje de aciertos en pruebas de MuscleKIN.

Se llevaron a cabo pruebas de cuatro series por cada ejercicio comprendido por 12 repeticiones, tomando como muestra principal dos series de las cuatro que se efectuaron en total.

Cabe mencionar que las verificaciones del sistema que se efectuaron dentro de los espacios del Centro Universitario UAEM Atlacomulco fueron con la ayuda de un usuario fisicoculturista a nivel principiante o amateur, mientras que las muestras de las pruebas realizadas en las instalaciones del gimnasio "Total GYM" fueron bajo el manejo del instructor certificado, fisicoculturista a nivel avanzado. Los datos obtenidos se muestran en la Tabla 6.1.

Tabla 6.1 Datos (Repeticiones y porcentaje) de repeticiones correctas en pruebas del sistema "MuscleKIN".

	Levantamientos Laterales	Curl de Bíceps	Sentadilla	Press Militar	Peso Muerto
Laboratorio CU UAEM Atlacomulco	20 (83.33%)	22 (91.63%)	16 (66.64%)	22 (91.63%)	18 (74.97%)
CAA UAEM Atlacomulco	22 (91.63%)	22 (91.63%)	20 (83.33%)	24 (100%)	18 (74.97%)
Sala de usos múltiples (Total GYM)	20 (83.33%)	20 (83.33%)	18 (74.97%)	18 (74.97%)	14 (58.31%)
Área de pesas (Total GYM)	24 (100%)	24 (100%)	22 (91.63%)	24 (100%)	20 (83.33%)

Teniendo la información anterior en cuenta se pudo observar que los resultados variaron acorde al tipo de iluminación del entorno, adaptándose de una forma satisfactoria al ambiente que cuenta tanto con iluminación natural como artificial en este caso es el área de pesas del gimnasio "Total GYM", así mismo se observó que con un ambiente de iluminación natural como lo es el Centro de Autoacceso del CU UAEM Atlacomulco su funcionamiento se encuentra en un estado correcto, variando en precisión de la detección del esqueleto en un 0-8.36% respecto al ambiente óptimo; dejando a los ambientes poco iluminados natural o artificialmente como no óptimos para el uso de "MuscleKIN" con una variación de error entre 17.66-25.03% respecto al ambiente más óptimo.

Lo anterior es explicado debido a las especificaciones tanto de uso, como de hardware del sensor Kinect v1 para XBOX 360 el cual indica que para el uso del sensor siempre debe haber un ambiente iluminado adecuadamente, caso contrario, deriva en fallas en el skeleton tracking, y en la cámara de profundidad. Esto puedo ser arreglado con la adaptación del sistema a un sensor Kinect v2 el cuál es capaz de adaptarse a ambiente en todo tipo de iluminación incluyendo ambientes completamente oscuros (Véase Figura 6.19).

Figura 6.19 Detección de imagen en ambiente oscuro con Kinect V2 [Canal de YouTube: Microsoft Research, https://www.youtube.com/watch?v=JaOlUa57BWs].

Por otra parte, se observó que los ejercicios que comprenden un menor número de articulaciones para su realización y que para su ejecución se requiere encontrarse de pie frente al sistema, caso en concreto del curl de bíceps, el press militar o los levantamientos laterales tuvieron un menor porcentaje de falla, debido a que no se presenta ocultamiento o sobreposicionamiento en la detección de las articulaciones. Siendo sentadilla y peso muerto los que presentan un porcentaje mayor de falla debido a la calidad en la detección y colisión de articulaciones por el sensor Kinect, el cual hace el intento de predecir la posición en coordenadas de tres dimensiones, pero obviando la presencia de fallas debido a ello.

Finalmente se observó que gracias a la inclusión dentro de una región de todos los elementos que obtienen información constante de Kinect y su posterior liberación de memoria mediante su limpieza en instrucción y el llamado del Garbage Collector, permitieron un mejor manejo de la memoria del que se hace uso. En la Figura 6.20 se muestra la memoria usada tras las 6ta iteración del sistema, comprobando lo anterior.

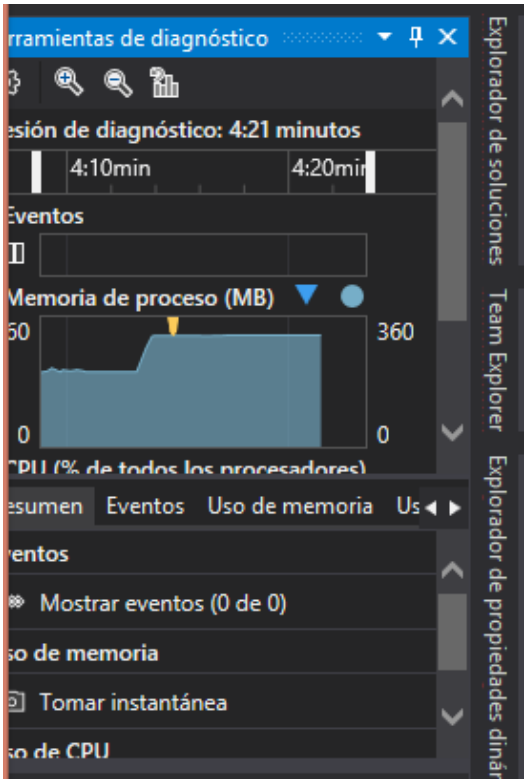

Figura 6.20 Memoria de proceso 6ta Iteración

CONCLUSIONES

El objetivo principal del proyecto de tesis es la realización de un sistema de visión artificial para realizar la corrección de posturas en fisicoculturistas mediante la utilización del sensor Kinect y el desarrollo de un algoritmo capaz de realizar dicha función a través de la posición de sus articulaciones en un ambiente en 3D. Se puede concluir que se ha cumplido con este objetivo de manera satisfactoria, manteniendo los grados de flexión en los rangos correctos y establecidos en los criterios específicos de un instructor certificado de pesas, concretamente 5º grados de flexión.

Además, las herramientas adicionales como el contador automático, el establecimiento de repeticiones y el apartado de información sirvieron para la correcta realización de los ejercicios en un ambiente real de un gimnasio, ya que proporcionó de manera concisa una explicación de las posturas de estos y permitió el enfoque primordial a la ejecución.

Durante la realización del proyecto se detectó un problema con la saturación de memoria, el cual radicaba en la programación del sistema y con el manejo de archivos WPF, ya que este tipo de programación requiere de una liberación a través de instrucciones. En este caso lo que se hizo fue administrar todos los recursos que adquirían constante toma de memoria dentro de una región para que en cada traspaso de ventana esta se limpiara evitando así saturación excesiva e innecesaria.

Finalmente, con base en el trabajo de investigación realizado, éste da un indicativo que pueda dar paso para futuras investigaciones en la realización de la actividad física, por ejemplo, suplir de manera definitiva la participación de un instructor; que pueda, además de corregir las posturas de los ejercicios en el área libre, corregir las posturas en máquinas o en las distintas poses que se realizan. Además se puede dotar al sistema con una aplicación que pueda incluir la detección de complexión física del usuario, que basándose en dicha información sea capaz de proporcionar rutinas y dietas personalizadas, llevando a cabo su seguimiento a través de una base de datos, con información estadística de porcentaje de grasa corporal, estatura, índice de masa muscular, etc., para su progresión constante en base a sus objetivos, metas, experiencia, tipo de cuerpo y de entrenamiento; con un seguimiento personal.

REFERENCIAS

"Naity", 2017. *User Tracking with LabVIEW and Kinect based on the OpenNI Interface.* [Online]
Available at: http://forums.ni.com/t5/Example-Program-Drafts/User-Tracking-with-LabVIEW-and-Kinect-based-on-the-OpenNI/ta-p/3491209
[Accessed 2017 Febrero 15].

Andreo, E. T., 2012. *Tesis de Licenciatura: Desarrollo de Interfaces de usuario naturales con Kinect.* Valencia, España: Universidad Politécnica de Valencia.

A. V., 2016. *Admira Visión.* [Online]
Available at: http://www.admiravision.es/es/articulos/divulgacion/articulo/anatomia-ocular#.WjcEUVXibIU
[Accessed 15 Agosto 2017].

Cherebetiu, G., Anca, C. & Galeana, M., 1988. *Entrenamiento deportivo.* México: Pax.

C. I. P. E. T., 2017. *Visión Artificial.* [Online]
Available at: http://www.etitudela.com/celula/downloads/visionartificial.pdf
[Accessed 22 Octubre 2017].

Colegio de Ópticos, C. y. O. d. C., 2017. *Fisiología del Ojo humano.* [Online]
Available at: http://www.colegiodeopticos.cl/fisiologia-del-ojo-humano-2/
[Accessed 13 Noviembre 2017].

E. Hannula, D., J. Simons, D. & J. C., N., 2005. Imaging implicit perception: promise and pitfalls. *Nature Reviews Neuroscience,* Issue 6, p. 247–255.

Edison Muñoz, J., Felipe Villada, J. & Giraldo Trujillo, J. C., 2013. Exergames: una herramienta tecnológica para la actividad física. *Revista Médica de Risaralda*, 19(2), pp. 126-130.

E. I., 2012. *Kinect y Processing: SimpleOpenNI.* [Online]
Available at: https://escenografiaaumentada.wordpress.com/2012/05/07/kinect-y-

processing-simpleopenni/
[Accessed 02 Noviembre 2017].

Geijo Vegas, N., 2016. *El ojo y la cámara fotográfica. Semejanzas y Diferencias.* [Online]
Available at: http://uvadoc.uva.es/bitstream/10324/19108/1/TFG-G%201799.pdf
[Accessed 20 Agosto 2017].

González Rosquete, V., 2013. *Advant y Advant-ed: plataforma para el entrenamiento cognitivo y físico con Kinect.* [Online]
Available at: http://diversidad.murciaeduca.es/publicaciones/dea2012/docs/vgonzalez.pdf
[Accessed 17 Marzo 2017].

Hernández, L., 2014. *DecaBlogs Running.* [Online]
Available at: http://blog.running.decathlon.es/mejora-tu-carrera-con-pliometricos/
[Accessed 07 Marzo 2018].

Hernández, L. d. V., 2016. *¿ Qué es WPF ? un repaso por sus características.* [Online]
Available at: https://programarfacil.com/blog/programacion-net-blog/que-es-wpf/
[Accessed 08 Junio 2018].

INEGI, 2014. *Módulo de Práctica Deportiva y Ejercicio Físico: Resultados de mayo 2014*, s.l.: INEGI.

INEGI, 2016. *Módulo de Práctica Deportiva y Ejercicio Físico: Resultados de mayo 2016*, s.l.: INEGI.

LabView, 2017. *Labview.* [Online]
Available at: http://digital.ni.com/public.nsf/allkb/0F48AC4190C2EB4E8625756E007C7922
[Accessed 10 Agosto 2017].

M., 2011. *XBOX.* [Online]
Available at: https://marketplace.xbox.com/es-MX/Product/UFC-Personal-

Trainer/66acd000-77fe-1000-9115-d8025451085b#
[Accessed 12 Febrero 2018].

M., 2012. *XBOX*. [Online]
Available at: http://marketplace.xbox.com/en-US/Product/Nike-Kinect-Training/66acd000-77fe-1000-9115-d8024d53090f
[Accessed 12 Febrero 2018].

Martinez Zarzuela, M. et al., 2011. *Monitorización del cuerpo humano en 3D mediante tecnología Kinect.* [Online]
Available at: http://www.saaei.org/edicion2011/papers/SE1/SE106.pdf.
[Accessed 02 Febrero 2017].

Martínez, M. M., 2010. *Proyecto Fin de Máster en Ingeniería Informática para la Industria: Técnicas de visión estereoscópica para determinar la estructura tridimensional de la escena.* Madrid, España: Universidad Complutense de Madrid (UCM).

M. E., 2011. *XBOX*. [Online]
Available at: http://marketplace.xbox.com/es-MX/Product/Your-Shape-Fitness-Evolved/66acd000-77fe-1000-9115-d8025553084f
[Accessed 12 Febrero 2018].

Microsoft, 2017. *XAML overview (WPF).* [Online]
Available at: https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf
[Accessed 08 Junio 2018].

Miramon, J. F., 2013. *Tesis: Implementación de reconocimiento facial y gesticulaciones para el manejo y control de proyecto RESA.* Tehuacán, Puebla: Instituto Tcenológico de Tehuacán.

M. K. h., 2017. *Kinect hardware.* [Online]
Available at: https://developer.microsoft.com/en-us/windows/kinect/hardware
[Accessed 2017 Marzo 16].

M. K. H. s., 2017. *Microsoft.* [Online] Available at: https://developer.microsoft.com/es-es/windows/kinect/hardware-setup [Accessed 10 Agosto 2017].

M. p. p., 2017. *Musculación para principiantes.* [Online] Available at: https://www.musculacionparaprincipiantes.com/ [Accessed 20 Noviembre 2017].

MSN, 2017. *¿Qué es y para qué sirve Visual Studio 2017?.* [Online] Available at: https://www.msn.com/es-es/noticias/microsoftstore/%C2%BFqu%C3%A9-es-y-para-qu%C3%A9-sirve-visual-studio-2017/ar-AAnLUC9
[Accessed 14 Mayo 20].

Muños Cardona, J. E., Henao Gallo, O. A. & López Herrera, J. F., 2013. *Sistema de Rehabilitación basado en el Uso de Análisis Biomecánico y Videojuegos mediante el Sensor Kinect.* [Online] Available at: http://itmojs.itm.edu.co/index.php/tecnologicas/article/view/454/465 [Accessed 02 Febrero 2017].

Murillo, A., 2017. *Kinect for developers.* [Online] Available at: http://www.kinectfordevelopers.com/es/2012/11/06/que-es-el-dispositivo-kinect/
[Accessed 10 Septiembre 2017].

M. V. S. r., 2017. *Microsoft.* [Online] Available at: http://www.microsoft.com/es-es/download/details.aspx?id=12752 [Accessed 2017 Agosto 10].

N. Platanov, V. & M. Bulatova, M., 2001. *La preparación Física.* Barcelona, España: Paidotribo.

Navarrete, G., 2010. *LevelUP.* [Online] Available at: https://www.levelup.com/Xbox-360/juegos/34760/Your-Shape-Fitness-Evolved/review
[Accessed 12 Febrero 2018].

OMS, 2017. *Organización Mundial de la Salud.* [Online] Available at: http://www.who.int/dietphysicalactivity/pa/es/ [Accessed 16 Agosto 2017].

Osuna, D. L., 2018. *IES "Leonardo Da Vinci": Luz y Visión.* [Online] Available at: http://intercentres.edu.gva.es/iesleonardodavinci/Fisica/Vision/Luz-vision08.htm
[Accessed 23 Marzo 2018].

país, P. d. E., 2017. ¿Qué es el Culturismo?. *Periódico digital. El país*.

Pajares Martinsanz, G. & de la Cruz García, J. M., 2008. *Visión por Computador: Imágenes digitales y aplicaciones.* Madrid: RA-MA.

Pantaleo, G. & Lis Rinaudo, L., 2015. *Ingeniería de Software.* Buenos Aires, Argentina: Alfaomega.

Pearl, B. & Moran, G. T., 1990. *Enciclopedia general del ejercicio: Musculación.* Barcelona, España: Paidotribo.

Pfeiffer, S., 2011. *Guiado gestual de un robot humanoide.* [Online] Available at: http://upcommons.upc.edu/handle/2099.1/12454 [Accessed 2017 Marzo 15].

Quinche Curtidor, J. C., 2015. *Dispositivos de captura de movimiento (Kinect), para la navegación de experiencias formativas en ambientes virtuales 3D,* s.l.: Universidad Autónoma de México.

Rodríguez Mier, F. A., 2017. *Oftalmologia Figueres.* [Online] Available at: http://www.oftalmologiafigueres.com/pdf/EL%20OJO%20Y%20LA%20CAMARA%20FOTOGRAFICA%20ok.pdf
[Accessed 20 Octubre 2017].

Ruíz Olaya, A. F., 2008. *Sistema Robótico Multimodal para Análisis y Estudios en Biomécanica, Movimiento Humano y Control Neuromotor,* Leganés: UNIVERSIDAD CARLOS III DE MADRID.

Sardi, L. D., 2012. *Algoritmo de Segmentación Online de Imágenes en Secuencias de Video.* [Online]
Available at: http://materias.fi.uba.ar/7500/Sardi.pdf
[Accessed 15 Marzo 2017].

Shapiro, L. & Stockman, G., 2000. *Computer Vision.* s.l.:Prentice Hill.

Shotton, J. et al., 2011. *Microsoft Research.* [Online]
Available at: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/BodyPartRecognition.pdf
[Accessed 17 Agosto 2017].

SmartSpot, 2017. *SmartSpot.* [Online]
Available at: https://www.smartspot.io/
[Accessed 24 Agosto 2017].

UAEMEX, 2018. *Perfil de Egreso Ingeniería en Computación.* [Online]
Available at: http://fingenieria.uaemex.mx/portal/coordinacion/ICO/perfilEgresoICO.php
[Accessed 24 Marzo 2018].

V. A., 2012. *Aplicación práctica de la visión artificial en el control de procesos industriales.* [Online]
Available at: http://visionartificial.fpcat.cat/wp-content/uploads/UD_1_didac_Conceptos_previos.pdf
[Accessed 24 Octubre 2017].

Webb, J. & Ashley, J., 2011. *Beginning Kinect Programming with te Microsoft Kinect SDK.* 1era. ed. s.l.:Apress.

ANEXOS: Código fuente

Clase MainWindow.cs

Clase principal del programa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{

    //Inicialización y Propiedades de la ventana MainWindow

    public partial class MainWindow : Window
    {
        // Variables para propiedades de pantalla
        private const int MinimumScreenWidth = 1920;
        private const int MinimumScreenHeight = 1080;


        public MainWindow()
        {
            InitializeComponent();

            Loaded += MainWindow_Loaded;

        }

        void MainWindow_Loaded(object sender, RoutedEventArgs e)
        {
            // Obtiene el tamaño maximo de la pantalla en uso
            double height = SystemParameters.PrimaryScreenHeight;
            double width = SystemParameters.PrimaryScreenWidth;

            // Condición si la pantalla es menor a 1920 x 1080 entonces
el usuario no obtendrá la mejor experiencia y establece las propiedades
máximas
            if ((width < MinimumScreenWidth) || (height <
MinimumScreenHeight))
            {
                MessageBoxResult continueResult =
MessageBox.Show(Properties.Resources.SuboptimalScreenResolutionMessage,
```

```
Properties.Resources.SuboptimalScreenResolutionTitle,
MessageBoxButton.YesNo);
                if (continueResult == MessageBoxResult.No)
                {
                    this.Close();
                }
            }
            this.WindowState = System.Windows.WindowState.Maximized;
            this.WindowStyle = System.Windows.WindowStyle.None;

            // Condición para reedireccionamiento a MainMenu
            if (Generics.loadingStatus == 0)
            {
                _mainFrame.Source = new Uri("MainMenu.xaml",
UriKind.Relative);
                Generics.loadingStatus = 1;
            }
        }

    }
}
```

Clase MainMenu.cs

Contiene la interfaz natural de usuario principal, un menú deslizable que contiene enlaces a las ventanas de corrección y de información de los 5 ejercicios en el sistema.

```
using Microsoft.Kinect;
using Microsoft.Kinect.Toolkit;
using Microsoft.Kinect.Toolkit.Controls;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    public partial class MainMenu : Page
    {
        //Región Kinect; variable sensorChooser
        #region "Kinect"
```

```
        private readonly KinectSensorChooser sensorChooser;


        #endregion
        public MainMenu()
        {

            this.InitializeComponent();
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            if (Generics.GlobalKinectSensorChooser == null)
            {
                // inicializa el sensor chooser and UI
                this.sensorChooser = new KinectSensorChooser();
                this.sensorChooser.KinectChanged +=
SensorChooserOnKinectChanged;
                this.sensorChooserUi.KinectSensorChooser =
this.sensorChooser;
                this.sensorChooser.Start();
                Generics.GlobalKinectSensorChooser =
this.sensorChooser;
            }
            else
            {
                this.sensorChooser = new KinectSensorChooser();
                this.sensorChooser =
Generics.GlobalKinectSensorChooser;
                this.sensorChooser.KinectChanged +=
SensorChooserOnKinectChanged;
                this.sensorChooserUi.KinectSensorChooser =
sensorChooser;
            }

            // Enlaza el sensor chooser del sensor actual a la
KinectRegion
            var regionSensorBinding = new Binding("Kinect") { Source =
this.sensorChooser };

            //Borra datos de los element contenidos en la región
BindingOperations.SetBinding(this.kinectRegion,KinectRegion.KinectSenso
rProperty, regionSensorBinding);
        }



        #region "Nuevo Kinect"


        // Deteccion e inicializacion de kinect y streams
        private static void SensorChooserOnKinectChanged(object sender,
KinectChangedEventArgs args)
        {
            // Condicional para habilitar/deshabilitar
      NewSensor/Oldsensor en caso de contar con más de un Kinect y su
      desconexión abrupta
            if (args.OldSensor != null)
            {
```

```
                try
                {
                    args.OldSensor.DepthStream.Range = DepthRange.Near;
args.OldSensor.SkeletonStream.EnableTrackingInNearRange = false;
                    args.OldSensor.DepthStream.Disable();
                    args.OldSensor.SkeletonStream.Disable();
                }
                catch (InvalidOperationException)
                {
                }
            }

            if (args.NewSensor != null)
            {
                try
                {
                    args.NewSensor.ElevationAngle = 0;
args.NewSensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps
30);
                    args.NewSensor.SkeletonStream.Enable();

                    try
                    {
                        args.NewSensor.DepthStream.Range =
DepthRange.Default;
args.NewSensor.SkeletonStream.EnableTrackingInNearRange = true;
                    }
                    catch (InvalidOperationException)
                    {
                        args.NewSensor.DepthStream.Range =
DepthRange.Default;
args.NewSensor.SkeletonStream.EnableTrackingInNearRange = false;
                    }
                }
                catch (InvalidOperationException)
                {
                }
            }
        }

        #endregion


        //Botones del menu...
        //Enlaces a Ventanas de correción
        //Enlaces a ventanas de Información
        private void KinectTileButton_Click(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
```

```
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("LevantLat.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_1(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("Info_Levantamiento.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_2(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("CurlBiceps.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_3(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("Info_Curl.xaml", UriKind.Relative);

        }

        private void KinectTileButton_Click_4(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("PressMilitar.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_5(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
```

```
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("Info_Press.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_6(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("Sentadilla.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_7(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("Info_Sentadilla.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_8(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("PesoMuerto.xaml", UriKind.Relative);
        }

        private void KinectTileButton_Click_9(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("Info_PesoMuerto.xaml", UriKind.Relative);
        }
    }
}
```

Clase LevantLat.cs

Clase para la correción de postura del ejercicio levantamiento lateral. Nota: Las clases de los distintos ejercicios varían en las condiconales de posicionamiento de las articulaciones, de igual manera incluyen o excluyen articulaciones acorde al ejercicio

```
using Microsoft.Kinect;
using Microsoft.Kinect.Toolkit;
using Microsoft.Kinect.Toolkit.Controls;
using System;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Collections.Generic;
using System.Text;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Threading;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para LevantLat.xaml
    /// </summary>
    public partial class LevantLat : Page
    {
        #region "Kinect"
        KinectSensor kinect;
        private readonly KinectSensorChooser sensorChooser;
        int i = 0;
        String mensajeContador = "";
        String mensajeContadorAutomatico = "";
        int contador = 0;
        #endregion

        public LevantLat()
        {
            this.InitializeComponent();
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            // Inicializa sensor chooser and UI
            this.sensorChooser = new KinectSensorChooser();
            this.sensorChooser = Generics.GlobalKinectSensorChooser;
            this.sensorChooser.KinectChanged +=
SensorChooserOnKinectChanged;
            this.sensorChooserUi.KinectSensorChooser = sensorChooser;
            // Enlaza el sensor chooser del sensor actual a la
KinectRegion
```

```
            var regionSensorBinding = new Binding("Kinect") { Source =
this.sensorChooser };
            BindingOperations.SetBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty, regionSensorBinding);
        }

        #region "Nuevo Kinect"

        //Inicializa Kinect y Streams
        private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            kinect = KinectSensor.KinectSensors.FirstOrDefault();
            kinect.Start();

            //Habilita los sensores o camaras de Kinect
            try
            {
                kinect.DepthStream.Enable();
                kinect.ColorStream.Enable();
                kinect.SkeletonStream.Enable();
            }
            catch
            {
                MessageBox.Show("Fallo al Inicializar Kinect", "Visor
de Kinect");
                Application.Current.Shutdown();
            }
            kinect.ColorFrameReady += Kinect_ColorFrameReady;
            kinect.SkeletonFrameReady += Kinect_SkeletonFrameReady;
        }



        //Código en caso de cierre de ventana, stop sensor
        private void WindowClosing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.Stop();
        }


        //Código en caso de errores, desconexión y conexión de sensor
        void SensorChooserOnKinectChanged(object sender,
KinectChangedEventArgs args)
        {
            if (args.OldSensor != null)
            {
                try
                {
                    args.OldSensor.DepthStream.Range = DepthRange.Near;

args.OldSensor.SkeletonStream.EnableTrackingInNearRange = false;
                    args.OldSensor.DepthStream.Disable();
                    args.OldSensor.SkeletonStream.Disable();
                }
```

```
                catch (InvalidOperationException)
                {
                }
            }

            if (args.NewSensor != null)
            {
                try
                {

args.NewSensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps
30);
                    args.NewSensor.ColorStream.Enable();
                    args.NewSensor.SkeletonStream.Enable();

                    try
                    {
                        args.NewSensor.DepthStream.Range =
DepthRange.Default;

args.NewSensor.SkeletonStream.EnableTrackingInNearRange = true;
                    }
                    catch (InvalidOperationException)
                    {
                        args.NewSensor.DepthStream.Range =
DepthRange.Default;

args.NewSensor.SkeletonStream.EnableTrackingInNearRange = false;
                    }
                }
                catch (InvalidOperationException)
                {
                }
            }
        }

        //Cámara RGB
        private void Kinect_ColorFrameReady(object sender,
ColorImageFrameReadyEventArgs e)
        {
            //Using para la recolección constante de frames
            using (ColorImageFrame frameImagen =
e.OpenColorImageFrame())
            {
                //Condicionales en caso de recolección de Frames
                if (frameImagen == null) return;
                byte[] datosColor = new
byte[frameImagen.PixelDataLength];
                frameImagen.CopyPixelDataTo(datosColor);

                ventana.Source = BitmapSource.Create(
                    frameImagen.Width, frameImagen.Height,
                    96,
                    96,
                    PixelFormats.Bgr32,
                    null,
                    datosColor,
```

```
                    frameImagen.Width * frameImagen.BytesPerPixel
                    );
            }
        }

        //Skeleton Stream
        private void Kinect_SkeletonFrameReady(object sender,
SkeletonFrameReadyEventArgs e)
        {
            canvasEsqueleto.Children.Clear();
            Skeleton[] esqueletos = null;
            using (SkeletonFrame framesEsqueleto =
e.OpenSkeletonFrame())
            {
                if (framesEsqueleto != null)
                {
                    esqueletos = new
Skeleton[framesEsqueleto.SkeletonArrayLength];
                    framesEsqueleto.CopySkeletonDataTo(esqueletos);
                }
            }
            if (esqueletos == null) return;
            foreach (Skeleton esqueleto in esqueletos)
            {
                //Colisión de Bordes
                if (esqueleto.TrackingState ==
SkeletonTrackingState.Tracked)
                {
                  // indicativos visuales deshabilitados
                    status1.Visibility = Visibility.Hidden;
                    Advertencia.Visibility = Visibility.Hidden;
                    Advertencia2.Visibility = Visibility.Hidden;
                    Abajo.Visibility = Visibility.Hidden;
                    Arriba.Visibility = Visibility.Hidden;
                    Izquierda.Visibility = Visibility.Hidden;
                    Derecha.Visibility = Visibility.Hidden;

                  //Condicionales de bordes
                    if ((esqueleto.ClippedEdges & FrameEdges.Bottom) !=
0)
                        Abajo.Visibility = Visibility.Visible;
                    if ((esqueleto.ClippedEdges & FrameEdges.Top) != 0)
                        Arriba.Visibility = Visibility.Visible;
                    if ((esqueleto.ClippedEdges & FrameEdges.Right) !=
0)
                        Derecha.Visibility = Visibility.Visible;
                    if ((esqueleto.ClippedEdges & FrameEdges.Left) !=
0)
                        Izquierda.Visibility = Visibility.Visible;

                    //Variables tipo Koint para cada articulación
                    Joint jointCabeza =
esqueleto.Joints[JointType.Head];
                    SkeletonPoint posicionCabeza =
jointCabeza.Position;
                    Joint jointHombroCentro =
esqueleto.Joints[JointType.ShoulderCenter];
```

```
                    SkeletonPoint posicionHombroCen =
jointHombroCentro.Position;
                    Joint jointEspina =
esqueleto.Joints[JointType.Spine];
                    SkeletonPoint posicionEspina =
jointEspina.Position;
                    Joint jointCaderaCentro =
esqueleto.Joints[JointType.HipCenter];
                    SkeletonPoint posicionCaderaCen =
jointCaderaCentro.Position;

                    Joint jointManoDerecha =
esqueleto.Joints[JointType.HandRight];
                    SkeletonPoint posicionManoDer =
jointManoDerecha.Position;
                    Joint jointCodoDerecho =
esqueleto.Joints[JointType.ElbowRight];
                    SkeletonPoint posicionCodoDer =
jointCodoDerecho.Position;
                    Joint jointHombroDerecho =
esqueleto.Joints[JointType.ShoulderRight];
                    SkeletonPoint posicionHombroDer =
jointHombroDerecho.Position;
                    Joint jointMunecaDerecha =
esqueleto.Joints[JointType.WristRight];
                    SkeletonPoint posicionMunecaDer =
jointMunecaDerecha.Position;
                    Joint jointPieDerecho =
esqueleto.Joints[JointType.FootRight];
                    SkeletonPoint posicionPieDer =
jointPieDerecho.Position;
                    Joint jointRodillaDerecha =
esqueleto.Joints[JointType.KneeRight];
                    SkeletonPoint posicionRodillaDer =
jointRodillaDerecha.Position;
                    Joint jointTobilloDerecho =
esqueleto.Joints[JointType.AnkleRight];
                    SkeletonPoint posicionTobilloDer =
jointTobilloDerecho.Position;
                    Joint jointCaderaDerecha =
esqueleto.Joints[JointType.HipRight];
                    SkeletonPoint posicionCaderaDer =
jointCaderaDerecha.Position;

                    Joint jointCodoIzquierdo =
esqueleto.Joints[JointType.ElbowLeft];
                    SkeletonPoint posicionCodoIzq =
jointCodoIzquierdo.Position;
                    Joint jointManoIzquierda =
esqueleto.Joints[JointType.HandLeft];
                    SkeletonPoint posicionManoIzq =
jointManoIzquierda.Position;
                    Joint jointHombroIzquierdo =
esqueleto.Joints[JointType.ShoulderLeft];
                    SkeletonPoint posicionHombroIzq =
jointHombroIzquierdo.Position;
```

```
                    Joint jointMunecaIzquierda =
esqueleto.Joints[JointType.WristLeft];
                    SkeletonPoint posicionMunecaIzq =
jointMunecaIzquierda.Position;
                    Joint jointPieIzquierdo =
esqueleto.Joints[JointType.FootLeft];
                    SkeletonPoint posicionPieIzq =
jointPieIzquierdo.Position;
                    Joint jointRodillaIzquierda =
esqueleto.Joints[JointType.KneeLeft];
                    SkeletonPoint posicionRodillaIzq =
jointRodillaIzquierda.Position;
                    Joint jointTobilloIzquierdo =
esqueleto.Joints[JointType.AnkleLeft];
                    SkeletonPoint posicionTobilloIzq =
jointTobilloIzquierdo.Position;
                    Joint jointCaderaIzquierda =
esqueleto.Joints[JointType.HipLeft];
                    SkeletonPoint posicionCaderaIzq =
jointCaderaIzquierda.Position;


                    //Propiedades de Línea
                    Line huesoCabeza = new Line();
                    huesoCabeza.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoCabeza.StrokeThickness = 5;
                    Line huesoEspina = new Line();
                    huesoEspina.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoEspina.StrokeThickness = 5;
                    Line huesoEspinaAbajo = new Line();
                    huesoEspinaAbajo.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoEspinaAbajo.StrokeThickness = 5;
                    Line huesoHombroCen = new Line();
                    huesoHombroCen.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoHombroCen.StrokeThickness = 5;

                    Line huesoBrazoDer = new Line();
                    huesoBrazoDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoBrazoDer.StrokeThickness = 5;
                    Line huesoManoDer = new Line();
                    huesoManoDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoManoDer.StrokeThickness = 5;
                    Line huesoBrazoMayDer = new Line();
                    huesoBrazoMayDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoBrazoMayDer.StrokeThickness = 5;
                    Line huesoPieDer = new Line();
                    huesoPieDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoPieDer.StrokeThickness = 5;
                    Line huesoPantorrillaDer = new Line();
```

```
                    huesoPantorrillaDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoPantorrillaDer.StrokeThickness = 5;
                    Line huesoPiernaDer = new Line();
                    huesoPiernaDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoPiernaDer.StrokeThickness = 5;
                    Line huesoCaderaDer = new Line();
                    huesoCaderaDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoCaderaDer.StrokeThickness = 5;
                    Line huesoHombroDer = new Line();
                    huesoHombroDer.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoHombroDer.StrokeThickness = 5;

                    Line huesoBrazoIzq = new Line();
                    huesoBrazoIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoBrazoIzq.StrokeThickness = 5;
                    Line huesoManoIzq = new Line();
                    huesoManoIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoManoIzq.StrokeThickness = 5;
                    Line huesoBrazoMayIzq = new Line();
                    huesoBrazoMayIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoBrazoMayIzq.StrokeThickness = 5;
                    Line huesoPieIzq = new Line();
                    huesoPieIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoPieIzq.StrokeThickness = 5;
                    Line huesoPantorrillaIzq = new Line();
                    huesoPantorrillaIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoPantorrillaIzq.StrokeThickness = 5;
                    Line huesoPiernaIzq = new Line();
                    huesoPiernaIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoPiernaIzq.StrokeThickness = 5;
                    Line huesoCaderaIzq = new Line();
                    huesoCaderaIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoCaderaIzq.StrokeThickness = 5;
                    Line huesoHombroIzq = new Line();
                    huesoHombroIzq.Stroke = new
SolidColorBrush(Colors.Green);
                    huesoHombroIzq.StrokeThickness = 5;


                    //Dibujado de puntos
                    //Manos
                    ColorImagePoint puntoManoDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointManoDerecha.P
osition, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoManoDer.X1 = puntoManoDer.X;
                    huesoManoDer.Y1 = puntoManoDer.Y;
```

```
                    ColorImagePoint puntoMunecaDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointMunecaDerecha
.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoManoDer.X2 = puntoMunecaDer.X;
                    huesoManoDer.Y2 = puntoMunecaDer.Y;
                    ColorImagePoint puntoManoIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointManoIzquierda
.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoManoIzq.X1 = puntoManoIzq.X;
                    huesoManoIzq.Y1 = puntoManoIzq.Y;
                    ColorImagePoint puntoMunecaIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointMunecaIzquier
da.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoManoIzq.X2 = puntoMunecaIzq.X;
                    huesoManoIzq.Y2 = puntoMunecaIzq.Y;
                    canvasEsqueleto.Children.Add(huesoManoDer);
                    canvasEsqueleto.Children.Add(huesoManoIzq);

                    //Antebrazos
                    ColorImagePoint puntoCodoDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointCodoDerecho.P
osition, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoBrazoDer.X1 = puntoCodoDer.X;
                    huesoBrazoDer.Y1 = puntoCodoDer.Y;
                    huesoBrazoDer.X2 = puntoMunecaDer.X;
                    huesoBrazoDer.Y2 = puntoMunecaDer.Y;
                    ColorImagePoint puntoCodoIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointCodoIzquierdo
.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoBrazoIzq.X1 = puntoCodoIzq.X;
                    huesoBrazoIzq.Y1 = puntoCodoIzq.Y;
                    huesoBrazoIzq.X2 = puntoMunecaIzq.X;
                    huesoBrazoIzq.Y2 = puntoMunecaIzq.Y;
                    canvasEsqueleto.Children.Add(huesoBrazoDer);
                    canvasEsqueleto.Children.Add(huesoBrazoIzq);

                    //Brazos
                    ColorImagePoint puntoHombroDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointHombroDerecho
.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoBrazoMayDer.X1 = puntoHombroDer.X;
                    huesoBrazoMayDer.Y1 = puntoHombroDer.Y;
                    huesoBrazoMayDer.X2 = puntoCodoDer.X;
                    huesoBrazoMayDer.Y2 = puntoCodoDer.Y;
                    ColorImagePoint puntoHombroIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointHombroIzquier
do.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoBrazoMayIzq.X1 = puntoHombroIzq.X;
                    huesoBrazoMayIzq.Y1 = puntoHombroIzq.Y;
                    huesoBrazoMayIzq.X2 = puntoCodoIzq.X;
                    huesoBrazoMayIzq.Y2 = puntoCodoIzq.Y;
                    canvasEsqueleto.Children.Add(huesoBrazoMayDer);
                    canvasEsqueleto.Children.Add(huesoBrazoMayIzq);

                    //Cabeza
```

```
                    ColorImagePoint puntoCabeza =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointCabeza.Positi
on, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoCabeza.X1 = puntoCabeza.X;
                    huesoCabeza.Y1 = puntoCabeza.Y;
                    ColorImagePoint puntoHombroCen =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointHombroCentro.
Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoCabeza.X2 = puntoHombroCen.X;
                    huesoCabeza.Y2 = puntoHombroCen.Y;
                    canvasEsqueleto.Children.Add(huesoCabeza);

                    //Hombros
                    huesoHombroDer.X1 = puntoHombroDer.X;
                    huesoHombroDer.Y1 = puntoHombroDer.Y;
                    huesoHombroDer.X2 = puntoHombroCen.X;
                    huesoHombroDer.Y2 = puntoHombroCen.Y;
                    huesoHombroIzq.X1 = puntoHombroIzq.X;
                    huesoHombroIzq.Y1 = puntoHombroIzq.Y;
                    huesoHombroIzq.X2 = puntoHombroCen.X;
                    huesoHombroIzq.Y2 = puntoHombroCen.Y;
                    canvasEsqueleto.Children.Add(huesoHombroDer);
                    canvasEsqueleto.Children.Add(huesoHombroIzq);

                    //Espina
                    huesoEspina.X1 = puntoHombroCen.X;
                    huesoEspina.Y1 = puntoHombroCen.Y;
                    ColorImagePoint puntoEspina =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointEspina.Positi
on, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoEspina.X2 = puntoEspina.X;
                    huesoEspina.Y2 = puntoEspina.Y;
                    huesoEspinaAbajo.X1 = puntoEspina.X;
                    huesoEspinaAbajo.Y1 = puntoEspina.Y;
                    ColorImagePoint puntoCaderaCen =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointCaderaCentro.
Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoEspinaAbajo.X2 = puntoCaderaCen.X;
                    huesoEspinaAbajo.Y2 = puntoCaderaCen.Y;
                    canvasEsqueleto.Children.Add(huesoEspina);
                    canvasEsqueleto.Children.Add(huesoEspinaAbajo);

                    //Piernas
                    ColorImagePoint puntoRodillaDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointRodillaDerech
a.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPiernaDer.X1 = puntoRodillaDer.X;
                    huesoPiernaDer.Y1 = puntoRodillaDer.Y;
                    ColorImagePoint puntoCaderaDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointCaderaDerecha
.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPiernaDer.X2 = puntoCaderaDer.X;
                    huesoPiernaDer.Y2 = puntoCaderaDer.Y;
                    ColorImagePoint puntoRodillaIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointRodillaIzquie
rda.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPiernaIzq.X1 = puntoRodillaIzq.X;
```

```
                    huesoPiernaIzq.Y1 = puntoRodillaIzq.Y;
                    ColorImagePoint puntoCaderaIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointCaderaIzquier
da.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPiernaIzq.X2 = puntoCaderaIzq.X;
                    huesoPiernaIzq.Y2 = puntoCaderaIzq.Y;
                    canvasEsqueleto.Children.Add(huesoPiernaDer);
                    canvasEsqueleto.Children.Add(huesoPiernaIzq);

                    //Pies
                    ColorImagePoint puntoPieDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointPieDerecho.Po
sition, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPieDer.X1 = puntoPieDer.X;
                    huesoPieDer.Y1 = puntoPieDer.Y;
                    ColorImagePoint puntoTobilloDer =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointTobilloDerech
o.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPieDer.X2 = puntoTobilloDer.X;
                    huesoPieDer.Y2 = puntoTobilloDer.Y;
                    ColorImagePoint puntoPieIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointPieIzquierdo.
Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPieIzq.X1 = puntoPieIzq.X;
                    huesoPieIzq.Y1 = puntoPieIzq.Y;
                    ColorImagePoint puntoTobilloIzq =
kinect.CoordinateMapper.MapSkeletonPointToColorPoint(jointTobilloIzquie
rdo.Position, ColorImageFormat.RgbResolution640x480Fps30);
                    huesoPieIzq.X2 = puntoTobilloIzq.X;
                    huesoPieIzq.Y2 = puntoTobilloIzq.Y;
                    canvasEsqueleto.Children.Add(huesoPieDer);
                    canvasEsqueleto.Children.Add(huesoPieIzq);

                    //Pantorrilla
                    huesoPantorrillaDer.X1 = puntoRodillaDer.X;
                    huesoPantorrillaDer.Y1 = puntoRodillaDer.Y;
                    huesoPantorrillaDer.X2 = puntoTobilloDer.X;
                    huesoPantorrillaDer.Y2 = puntoTobilloDer.Y;
                    huesoPantorrillaIzq.X1 = puntoRodillaIzq.X;
                    huesoPantorrillaIzq.Y1 = puntoRodillaIzq.Y;
                    huesoPantorrillaIzq.X2 = puntoTobilloIzq.X;
                    huesoPantorrillaIzq.Y2 = puntoTobilloIzq.Y;
                    canvasEsqueleto.Children.Add(huesoPantorrillaDer);
                    canvasEsqueleto.Children.Add(huesoPantorrillaIzq);

                    //Caderas
                    huesoCaderaDer.X1 = puntoCaderaDer.X;
                    huesoCaderaDer.Y1 = puntoCaderaDer.Y;
                    huesoCaderaDer.X2 = puntoCaderaCen.X;
                    huesoCaderaDer.Y2 = puntoCaderaCen.Y;
                    huesoCaderaIzq.X1 = puntoCaderaIzq.X;
                    huesoCaderaIzq.Y1 = puntoCaderaIzq.Y;
                    huesoCaderaIzq.X2 = puntoCaderaCen.X;
                    huesoCaderaIzq.Y2 = puntoCaderaCen.Y;
                    canvasEsqueleto.Children.Add(huesoCaderaDer);
                    canvasEsqueleto.Children.Add(huesoCaderaIzq);
```

```
                    //Algoritmo de corección (Condiciones)
                    if (posicionHombroCen.X >= -0.1 &&
posicionHombroCen.Y >= 0.1 && posicionHombroCen.X <= 0.3 &&
posicionHombroCen.Y <= 0.7 &&
                        posicionHombroDer.X >= 0.0 &&
posicionHombroDer.Y >= 0.4 && posicionHombroDer.X <= 0.4 &&
posicionHombroDer.Y <= 0.8 &&
                        posicionCabeza.X >= -0.1 && posicionCabeza.Y >=
0.6 && posicionCabeza.X <= 0.3 && posicionCabeza.Y <= 1.0 &&
                        posicionCaderaCen.X >= -0.1 &&
posicionCaderaCen.Y >= 0.0 && posicionCaderaCen.X <= 0.3 &&
posicionCaderaCen.Y <= 0.4 &&
                        posicionMunecaDer.X >= 0.2 &&
posicionMunecaDer.Y >= 0.0 && posicionMunecaDer.X <= 0.4 &&
posicionMunecaDer.Y <= 0.2 &&
                        posicionMunecaIzq.X >= -0.3 &&
posicionMunecaIzq.Y >= 0.0 && posicionMunecaIzq.X <= -0.1 &&
posicionMunecaIzq.Y <= 0.2 &&
                        posicionTobilloDer.X >= 0.0 &&
posicionTobilloDer.Y >= -0.9 && posicionTobilloDer.X <= 0.4 &&
posicionTobilloDer.Y <= -0.5 &&
                        posicionTobilloIzq.X >= -0.2 &&
posicionTobilloIzq.Y >= -0.9 && posicionTobilloIzq.X <= 0.2 &&
posicionTobilloIzq.Y <= -0.5)
                    {
                        //Indicativos habilitados
                        inicial.Visibility = Visibility.Visible;
                        bien.Visibility = Visibility.Hidden;
                        mal.Visibility = Visibility.Hidden;
                    }
                    else
                    {

                        //Boolean bandera = true, condición verdadera;
                        if (posicionHombroDer.X <= 0.4 &&
posicionHombroDer.X >= 0.0 && posicionHombroDer.Y <= 0.7 &&
posicionHombroDer.Y >= 0.3 &&
                            posicionMunecaDer.X <= 0.7 &&
posicionMunecaDer.X >= 0.5 && posicionMunecaDer.Y <= 0.7 &&
posicionMunecaDer.Y >= 0.5 &&
                            posicionHombroIzq.X <= 0.1 &&
posicionHombroIzq.X >= -0.3 && posicionHombroIzq.Y <= 0.8 &&
posicionHombroIzq.Y >= 0.4 &&
                            posicionMunecaIzq.X <= -0.3 &&
posicionMunecaIzq.X >= -0.6 && posicionMunecaIzq.Y <= 0.7 &&
posicionMunecaIzq.Y >= 0.5)
                            {
                                //indicativos habiltados, inicialización de
contador
                                contador =
int.Parse(textBlockContador.Text);
                                mensajeContador = contador.ToString();
                                bien.Visibility = Visibility.Visible;
                                mal.Visibility = Visibility.Hidden;
                                inicial.Visibility = Visibility.Hidden;
```

```
                            //Condicional para incremento de contador
automático
                            if (bien.Visibility == Visibility.Visible)
                            {
                                i++;
                              //Duerme el programa durante 700
milisegundos tras detectar un ejercicio correcta para evitar errores
por la toma de frames constantes
                                int milliseconds = 700;
                                mensajeContadorAutomatico =
i.ToString();
                                textBlockContadorAutomatico.Text =
mensajeContadorAutomatico;
                                Thread.Sleep(milliseconds);
                                if (mensajeContadorAutomatico ==
mensajeContador)
                                {
//Borrado de memoria de proceso
BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);

//Envía ventana con mensaje "Terminado" tras finalización de
repeticiones
(Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source
= new Uri("Terminado.xaml", UriKind.Relative);
                                }
                            }
                        }
                        else
                        {
                            mal.Visibility = Visibility.Visible;
                            bien.Visibility = Visibility.Hidden;
                            inicial.Visibility = Visibility.Hidden;
                        }
                    }
                }
            }
        }

        #endregion

        //Boton de regreso a ventana principal
        private void KinectTileButton_Click_3(object sender,
RoutedEventArgs e)
        {
            GC.Collect();
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("MainMenu.xaml", UriKind.Relative);
        }

        //Bontón de suma +1 al contador predeterminado
        private void KinectCircleButton_Click_ContadorMas(object
sender, RoutedEventArgs e)
```

```
        {
            contador = int.Parse(textBlockContador.Text);
            contador = contador + 1;
            mensajeContador = contador.ToString();
            textBlockContador.Text = mensajeContador;
        }

        //Bontón de suma -1 al contador predeterminado
        private void KinectCircleButton_Click_ContadorMenos(object
sender, RoutedEventArgs e)
        {
            contador = int.Parse(textBlockContador.Text);
            contador = contador - 1;
            mensajeContador = contador.ToString();
            textBlockContador.Text = mensajeContador;
        }

       //Bontón de reinicio de contador
        private void Reiniciar_Click(object sender, RoutedEventArgs e)
        {
            //Reiniciamos el contador de SET DE REPETICIONES a 0 para
indicar un nuevo numero de repeticiones
            contador = 0;
            textBlockContador.Text = contador.ToString();
            int cont = 0;
            textBlockContadorAutomatico.Text = cont.ToString();
        }
    }
}
```

Clase CurlBiceps.cs

```
//Condicionales de corrección de posturas curl de biceps

if (posicionHombroCen.X >= -0.1 &&
                        posicionHombroCen.Y >= 0.1 &&
                        posicionHombroCen.X <= 0.3 &&
                        posicionHombroCen.Y <= 0.7 &&

                        posicionHombroDer.X >= 0.0 &&
                        posicionHombroDer.Y >= 0.4 &&
                        posicionHombroDer.X <= 0.4 &&
                        posicionHombroDer.Y <= 0.8 &&

                        posicionCabeza.X >= -0.1 &&
                        posicionCabeza.Y >= 0.6 &&
                        posicionCabeza.X <= 0.3 &&
                        posicionCabeza.Y <= 1.0 &&

                        posicionCaderaCen.X >= -0.1 &&
                        posicionCaderaCen.Y >= 0.0 &&
                        posicionCaderaCen.X <= 0.3 &&
                        posicionCaderaCen.Y <= 0.4 &&

                        posicionMunecaDer.X >= 0.2 &&
```

```
                        posicionMunecaDer.Y >= 0.0 &&
                        posicionMunecaDer.X <= 0.4 &&
                        posicionMunecaDer.Y <= 0.2 &&

                        posicionMunecaIzq.X >= -0.3 &&
                        posicionMunecaIzq.Y >= 0.0 &&
                        posicionMunecaIzq.X <= -0.1 &&
                        posicionMunecaIzq.Y <= 0.2 &&

                        posicionTobilloDer.X >= 0.0 &&
                        posicionTobilloDer.Y >= -0.9 &&
                        posicionTobilloDer.X <= 0.4 &&
                        posicionTobilloDer.Y <= -0.5 &&

                        posicionTobilloIzq.X >= -0.2 &&
                        posicionTobilloIzq.Y >= -0.9 &&
                        posicionTobilloIzq.X <= 0.2 &&
                        posicionTobilloIzq.Y <= -0.5)
                    {
                        inicial.Visibility = Visibility.Visible;
                        bien.Visibility = Visibility.Hidden;
                        mal.Visibility = Visibility.Hidden;
                    }
                    else
                    {
                        //Ambas manos
                        if (posicionCodoDer.X >= 0.1 &&
posicionCodoDer.X <= 0.5 && posicionCodoDer.Y >= 0.2 &&
posicionCodoDer.Y <= 0.6 &&
                            posicionHombroDer.X >= 0.0 &&
posicionHombroDer.X <= 0.4 && posicionHombroDer.Y >= 0.4 &&
posicionHombroDer.Y <= 0.8 &&
                            posicionMunecaDer.X >= 0.1 &&
posicionMunecaDer.X <= 0.5 && posicionMunecaDer.Y >= 0.3 &&
posicionMunecaDer.Y <= 0.7 &&

                            posicionCodoIzq.X >= -0.4 &&
posicionCodoIzq.X <= 0.0 && posicionCodoIzq.Y >= 0.2 &&
posicionCodoIzq.Y <= 0.6 &&
                            posicionHombroIzq.X >= -0.3 &&
posicionHombroIzq.X <= 0.1 && posicionHombroIzq.Y >= 0.4 &&
posicionHombroIzq.Y <= 0.8 &&
                            posicionMunecaIzq.X >= -0.3 &&
posicionMunecaIzq.X <= 0.1 && posicionMunecaIzq.Y >= 0.3 &&
posicionMunecaIzq.Y <= 0.7)
                            {
                                contador =
int.Parse(textBlockContador.Text);
                                mensajeContador = contador.ToString();
                                bien.Visibility = Visibility.Visible;
                                mal.Visibility = Visibility.Hidden;
                                inicial.Visibility = Visibility.Hidden;
                                if (bien.Visibility == Visibility.Visible)
                                {
                                    i++;
                                    int milliseconds = 1000;
```

```
                                mensajeContadorAutomatico =
i.ToString();
                                textBlockContadorAutomatico.Text =
mensajeContadorAutomatico;
                                Thread.Sleep(milliseconds);
                                if (mensajeContadorAutomatico ==
mensajeContador)
                                {

BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
                                    //this.sensorChooser.KinectChanged
-= SensorChooserOnKinectChanged;

(Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source
= new Uri("Terminado.xaml", UriKind.Relative);
                                }
                            }
                        }
                        else
                        {

                            //Mano Derecha
                            if (posicionCodoDer.X >= 0.1 &&
posicionCodoDer.X <= 0.5 && posicionCodoDer.Y >= 0.2 &&
posicionCodoDer.Y <= 0.6 &&
                                posicionHombroDer.X >= 0.0 &&
posicionHombroDer.X <= 0.4 && posicionHombroDer.Y >= 0.4 &&
posicionHombroDer.Y <= 0.8 &&
                                posicionMunecaDer.X >= 0.1 &&
posicionMunecaDer.X <= 0.5 && posicionMunecaDer.Y >= 0.3 &&
posicionMunecaDer.Y <= 0.7)
                            {
                                contador =
int.Parse(textBlockContador.Text);
                                mensajeContador = contador.ToString();
                                bien.Visibility = Visibility.Visible;
                                mal.Visibility = Visibility.Hidden;
                                inicial.Visibility = Visibility.Hidden;
                                if (bien.Visibility ==
Visibility.Visible)
                                {
                                    i++;
                                    int milliseconds = 1000;
                                    mensajeContadorAutomatico =
i.ToString();
                                    textBlockContadorAutomatico.Text =
mensajeContadorAutomatico;
                                    Thread.Sleep(milliseconds);
                                    if (mensajeContadorAutomatico ==
mensajeContador)
                                    {

BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);

//this.sensorChooser.KinectChanged -= SensorChooserOnKinectChanged;
```

```
(Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source
= new Uri("Terminado.xaml", UriKind.Relative);
                                    }
                                    //}
                                }
                                else
                                {

                                    //Mano Izquierda
                                    if (posicionCodoIzq.X >= -0.4 &&
posicionCodoIzq.X <= 0.0 && posicionCodoIzq.Y >= 0.2 &&
posicionCodoIzq.Y <= 0.6 &&
                                        posicionHombroIzq.X >= -0.3 &&
posicionHombroIzq.X <= 0.1 && posicionHombroIzq.Y >= 0.4 &&
posicionHombroIzq.Y <= 0.8 &&
                                        posicionMunecaIzq.X >= -0.3 &&
posicionMunecaIzq.X <= 0.1 && posicionMunecaIzq.Y >= 0.3 &&
posicionMunecaIzq.Y <= 0.7)
                                    {
                                        contador =
int.Parse(textBlockContador.Text);
                                        mensajeContador =
contador.ToString();
                                        bien.Visibility =
Visibility.Visible;
                                        mal.Visibility =
Visibility.Hidden;
                                        inicial.Visibility =
Visibility.Hidden;
                                        if (bien.Visibility ==
Visibility.Visible)
                                        {
                                            i++;
                                            int milliseconds = 1000;
                                            mensajeContadorAutomatico =
i.ToString();

textBlockContadorAutomatico.Text = mensajeContadorAutomatico;
                                            Thread.Sleep(milliseconds);
                                            if
(mensajeContadorAutomatico == mensajeContador)
                                            {

BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);

//this.sensorChooser.KinectChanged -= SensorChooserOnKinectChanged;

(Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source
= new Uri("Terminado.xaml", UriKind.Relative);
                                            }
                                        }
                                    }
                                    else
                                    {
```

```
                                        mal.Visibility =
Visibility.Visible;
                                        bien.Visibility =
Visibility.Hidden;
                                        inicial.Visibility =
Visibility.Hidden;
                                    }
                                }
                            }
                        }
                    }
```

Clase PressMilitar.cs

```
//Condicionales de corrección de posturas de press militar

if (posicionHombroCen.X >= -0.2 &&
                        posicionHombroCen.Y >= 0.5 &&
                        posicionHombroCen.X <= 0.2 &&
                        posicionHombroCen.Y <= 0.9 &&

                        posicionHombroDer.X >= 0.0 &&
                        posicionHombroDer.Y >= 0.4 &&
                        posicionHombroDer.X <= 0.4 &&
                        posicionHombroDer.Y <= 0.8 &&

                        posicionHombroIzq.X >= -0.4 &&
                        posicionHombroIzq.Y >= 0.4 &&
                        posicionHombroIzq.X <= 0.0 &&
                        posicionHombroIzq.Y <= 0.8 &&

                        posicionMunecaIzq.X >= -0.7 &&
                        posicionMunecaIzq.Y >= 0.6 &&
                        posicionMunecaIzq.X <= -0.3 &&
                        posicionMunecaIzq.Y <= 0.8 &&

                        posicionCodoDer.X >= 0.2 &&
                        posicionCodoDer.Y >= 0.4 &&
                        posicionCodoDer.X <= 0.6 &&
                        posicionCodoDer.Y <= 0.6 &&

                        posicionCodoIzq.X >= -0.6 &&
                        posicionCodoIzq.Y >= 0.4 &&
                        posicionCodoIzq.X <= -0.2 &&
                        posicionCodoIzq.Y <= 0.6)
                    {
                        inicial.Visibility = Visibility.Visible;
                        bien.Visibility = Visibility.Hidden;
                        mal.Visibility = Visibility.Hidden;
                    }
                    else
                    {

                        //Boolean bandera = true;
```

```
                        if (posicionCodoDer.X >= 0.0 &&
posicionCodoDer.X <= 0.4 && posicionCodoDer.Y >= 0.6 &&
posicionCodoDer.Y <= 1.0 &&
                            posicionHombroDer.X >= 0.0 &&
posicionHombroDer.X <= 0.4 && posicionHombroDer.Y >= 0.4 &&
posicionHombroDer.Y <= 0.9 &&
                            // posicionMunecaDer.X >= 0.0 &&
posicionMunecaDer.X <= 0.4 && posicionMunecaDer.Y >= 0.9 &&
posicionMunecaDer.Y <= 1.3 &&

                            posicionCodoIzq.X >= -0.4 &&
posicionCodoIzq.X <= 0.0 && posicionCodoIzq.Y >= 0.7 &&
posicionCodoIzq.Y <= 1.1 &&
                            posicionHombroIzq.X >= -0.3 &&
posicionHombroIzq.X <= 0.1 && posicionHombroIzq.Y >= 0.5 &&
posicionHombroIzq.Y <= 0.9 &&
                            posicionMunecaIzq.X >= -0.3 &&
posicionMunecaIzq.X <= 0.1 && posicionMunecaIzq.Y >= 0.8 &&
posicionMunecaIzq.Y <= 1.2)
                        {
                            contador =
int.Parse(textBlockContador.Text);
                            mensajeContador = contador.ToString();
                            bien.Visibility = Visibility.Visible;
                            mal.Visibility = Visibility.Hidden;
                            inicial.Visibility = Visibility.Hidden;

                            if (bien.Visibility == Visibility.Visible)
                            {
                                i++;
                                int milliseconds = 1000;
                                mensajeContadorAutomatico =
i.ToString();
                                textBlockContadorAutomatico.Text =
mensajeContadorAutomatico;
                                Thread.Sleep(milliseconds);
                                if (mensajeContadorAutomatico ==
mensajeContador)
                                {

BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
                                    //this.sensorChooser.KinectChanged
-= SensorChooserOnKinectChanged;

(Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source
= new Uri("Terminado.xaml", UriKind.Relative);
                                }
                            }
                        }
                        else
                        {
                            mal.Visibility = Visibility.Visible;
                            bien.Visibility = Visibility.Hidden;
                            inicial.Visibility = Visibility.Hidden;
                        }
                    }
```

Clase PesoMuerto.cs

```
//Condicionales de corrección de posturas de ejercicio peso muerto

if (posicionHombroCen.X >= -0.2 &&
                        posicionHombroCen.Y >= 0.5 &&
                        posicionHombroCen.X <= 0.4 &&
                        posicionHombroCen.Y <= 0.9 &&

                        posicionCaderaCen.X >= -0.2 &&
                        posicionCaderaCen.Y >= 0.1 &&
                        posicionCaderaCen.X <= 0.2 &&
                        posicionCaderaCen.Y <= 0.5 &&
                        posicionTobilloDer.X >= 0.0 &&
                        posicionTobilloDer.Y >= -0.9 &&
                        posicionTobilloDer.X <= 0.4 &&
                        posicionTobilloDer.Y <= -0.5 &&

                        posicionTobilloIzq.X >= -0.4 &&
                        posicionTobilloIzq.Y >= -0.9 &&
                        posicionTobilloIzq.X <= 0.0 &&
                        posicionTobilloIzq.Y <= -0.5 &&

                        posicionRodillaDer.X >= 0.0 &&
                        posicionRodillaDer.Y >= -0.7 &&
                        posicionRodillaDer.X <= 0.4 &&
                        posicionRodillaDer.Y <= -0.3 &&

                        posicionRodillaIzq.X >= -0.3 &&
                        posicionRodillaIzq.Y >= -0.7 &&
                        posicionRodillaIzq.X <= 0.1 &&
                        posicionRodillaIzq.Y <= -0.3)
                    {
                        inicial.Visibility = Visibility.Visible;
                        bien.Visibility = Visibility.Hidden;
                        mal.Visibility = Visibility.Hidden;
                    }
                    else
                    {

                        //Boolean bandera = true;
                        if (posicionCaderaCen.X >= -0.2 &&
posicionCaderaCen.X <= 0.2 && posicionCaderaCen.Y >= -0.3 &&
posicionCaderaCen.Y <= 0.1 &&
                            posicionHombroCen.X >= -0.2 &&
posicionHombroCen.X <= 0.2 && posicionHombroCen.Y >= -0.1 &&
posicionHombroCen.Y <= 0.3 &&

                            posicionTobilloDer.X >= 0.0 &&
posicionTobilloDer.X <= 0.4 && posicionTobilloDer.Y >= -0.9 &&
posicionTobilloDer.Y <= -0.5 &&
                            posicionTobilloIzq.X >= -0.4 &&
posicionTobilloIzq.X <= 0.0 && posicionTobilloIzq.Y >= -0.9 &&
posicionTobilloIzq.Y <= -0.5 &&
```

```
                            posicionHombroDer.X >= 0.0 &&
posicionHombroDer.X <= 0.4 && posicionHombroDer.Y >= -0.1 &&
posicionHombroDer.Y <= 0.3 &&
                            posicionMunecaDer.X >= 0.1 &&
posicionMunecaDer.X <= 0.5 && posicionMunecaDer.Y >= -0.6 &&
posicionMunecaDer.Y <= -0.2 &&
                            posicionCaderaDer.X >= -0.1 &&
posicionCaderaDer.X <= 0.3 && posicionCaderaDer.Y >= -0.3 &&
posicionCaderaDer.Y <= 0.1 &&
                            posicionRodillaDer.X >= 0.0 &&
posicionRodillaDer.X <= 0.4 && posicionRodillaDer.Y >= -0.7 &&
posicionRodillaDer.Y <= -0.3 &&

                            posicionHombroIzq.X >= -0.4 &&
posicionHombroIzq.X <= 0.0 && posicionHombroIzq.Y >= -0.1 &&
posicionHombroIzq.Y <= 0.3 &&
                            posicionMunecaIzq.X >= -0.5 &&
posicionMunecaIzq.X <= -0.1 && posicionMunecaIzq.Y >= -0.6 &&
posicionMunecaIzq.Y <= -0.2 &&
                            posicionCaderaIzq.X >= -0.2 &&
posicionCaderaIzq.X <= 0.2 && posicionCaderaIzq.Y >= -0.3 &&
posicionCaderaIzq.Y <= 0.1 &&
                            posicionRodillaIzq.X >= -0.3 &&
posicionRodillaIzq.X <= 0.1 && posicionRodillaIzq.Y >= -0.7 &&
posicionRodillaIzq.Y <= -0.3)
                        {
                            contador =
int.Parse(textBlockContador.Text);
                            mensajeContador = contador.ToString();
                            bien.Visibility = Visibility.Visible;
                            mal.Visibility = Visibility.Hidden;
                            inicial.Visibility = Visibility.Hidden;

                            if (bien.Visibility == Visibility.Visible)
                            {
                                i++;
                                int milliseconds = 1500;
                                mensajeContadorAutomatico =
i.ToString();
                                textBlockContadorAutomatico.Text =
mensajeContadorAutomatico;
                                Thread.Sleep(milliseconds);
                                if (mensajeContadorAutomatico ==
mensajeContador)
                                {

BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
                                    //this.sensorChooser.KinectChanged
-= SensorChooserOnKinectChanged;

(Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source
= new Uri("Terminado.xaml", UriKind.Relative);
                                }
                            }
                        }
```

```
                    else
                    {
                        mal.Visibility = Visibility.Visible;
                        bien.Visibility = Visibility.Hidden;
                        inicial.Visibility = Visibility.Hidden;
                    }
                }
```

Clase Sentadilla.cs

```
//Condicionales de corrección de posturas de sentadilla

if (posicionHombroCen.X >= -0.2 && posicionHombroCen.Y >= 0.5 &&
posicionHombroCen.X <= 0.2 &&posicionHombroCen.Y <= 0.9 &&
                        posicionCaderaCen.X >= -0.1 &&
                        posicionCaderaCen.Y >= 0.0 &&
                        posicionCaderaCen.X <= 0.3 &&
                        posicionCaderaCen.Y <= 0.4 &&

                        posicionTobilloDer.X >= 0.0 &&
                        posicionTobilloDer.Y >= -0.9 &&
                        posicionTobilloDer.X <= 0.4 &&
                        posicionTobilloDer.Y <= -0.5 &&

                        posicionTobilloIzq.X >= -0.5 &&
                        posicionTobilloIzq.Y >= -0.9 &&
                        posicionTobilloIzq.X <= -0.1 &&
                        posicionTobilloIzq.Y <= -0.5 &&

                        posicionRodillaDer.X >= 0.0 &&
                        posicionRodillaDer.Y >= -0.5 &&
                        posicionRodillaDer.X <= 0.4 &&
                        posicionRodillaDer.Y <= -0.1 &&

                        posicionRodillaIzq.X >= -0.4 &&
                        posicionRodillaIzq.Y >= -0.5 &&
                        posicionRodillaIzq.X <= 0.0 &&
                        posicionRodillaIzq.Y <= -0.1)
                {
                    inicial.Visibility = Visibility.Visible;
                    bien.Visibility = Visibility.Hidden;
                    mal.Visibility = Visibility.Hidden;
                }
                else
                {

                    //Boolean bandera = true;
                    if (posicionHombroCen.X >= -0.2 &&
                    posicionHombroCen.Y >= -0.3 &&
                    posicionHombroCen.X <= 0.2 &&
                    posicionHombroCen.Y <= 0.1 &&
```

```
                        posicionCaderaCen.X >= -0.2 &&
                        posicionCaderaCen.Y >= -0.6 &&
                        posicionCaderaCen.X <= 0.2 &&
                        posicionCaderaCen.Y <= -0.2 &&

                        posicionTobilloDer.X >= 0.0 &&
                        posicionTobilloDer.Y >= -0.9 &&
                        posicionTobilloDer.X <= 0.4 &&
                        posicionTobilloDer.Y <= -0.5 &&

                        posicionTobilloIzq.X >= -0.5 &&
                        posicionTobilloIzq.Y >= -0.9 &&
                        posicionTobilloIzq.X <= -0.1 &&
                        posicionTobilloIzq.Y <= -0.5 &&

                        posicionRodillaDer.X >= 0.0 &&
                        posicionRodillaDer.Y >= -0.6 &&
                        posicionRodillaDer.X <= 0.4 &&
                        posicionRodillaDer.Y <= -0.2 &&

                        posicionRodillaIzq.X >= -0.5 &&
                        posicionRodillaIzq.Y >= -0.6 &&
                        posicionRodillaIzq.X <= -0.1 &&
                        posicionRodillaIzq.Y <= -0.2)
                        {
                            contador =
int.Parse(textBlockContador.Text);
                            mensajeContador = contador.ToString();
                            bien.Visibility = Visibility.Visible;
                            mal.Visibility = Visibility.Hidden;
                            inicial.Visibility = Visibility.Hidden;

                            if (bien.Visibility == Visibility.Visible)
                            {
                                i++;
                                int milliseconds = 1500;
                                mensajeContadorAutomatico =
i.ToString();
                                textBlockContadorAutomatico.Text =
mensajeContadorAutomatico;
                                Thread.Sleep(milliseconds);
                                if (mensajeContadorAutomatico ==
mensajeContador)
                                {

BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
                                    //this.sensorChooser.KinectChanged
-= SensorChooserOnKinectChanged;

(Application.Current.MainWindow.FindName("_mainFrame") as Frame).Source
= new Uri("Terminado.xaml", UriKind.Relative);
                                }
                            }
                        }
                        else
                        {
```

```
                mal.Visibility = Visibility.Visible;
                bien.Visibility = Visibility.Hidden;
                inicial.Visibility = Visibility.Hidden;
            }
        }
```

Clase InfoLevantLat.cs

Nota: En esta clase en general lo único que cambia es la información del ejercicio y del enlace de regreso a la clase MainMenu.xaml, siendo aplicables para los 5 ejercicios que incluye "MuscleKIN" por lo que no se considera importante la inclusión de cada clase de manera individual

```
using System;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using Microsoft.Kinect;
using Microsoft.Kinect.Toolkit;
using Microsoft.Kinect.Toolkit.Controls;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para Info_Levantamiento.xaml
    /// </summary>
    public partial class Info_Levantamiento : Page
    {

        #region "Kinect"
        private readonly KinectSensorChooser sensorChooser;

        #endregion

        public Info_Levantamiento()
        {
            this.InitializeComponent();
            // Inicializa sensor chooser and UI
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser = new KinectSensorChooser();
            this.sensorChooser = Generics.GlobalKinectSensorChooser;
            this.sensorChooser.KinectChanged +=
SensorChooserOnKinectChanged;
            this.sensorChooserUi.KinectSensorChooser = sensorChooser;
            var regionSensorBinding = new Binding("Kinect") { Source =
this.sensorChooser };
```

```
            BindingOperations.SetBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty, regionSensorBinding);
        }

        #region "Nuevo Kinect"

       //En caso de cierre de ventana= KinectStop
        private void WindowClosing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            this.sensorChooser.Stop();
        }


       //Inicialización de Kinect y Streams
        private static void SensorChooserOnKinectChanged(object sender,
KinectChangedEventArgs args)
        {
            if (args.OldSensor != null)
            {
                try
                {
                    args.OldSensor.DepthStream.Range = DepthRange.Near;

args.OldSensor.SkeletonStream.EnableTrackingInNearRange = false;
                    args.OldSensor.DepthStream.Disable();
                    args.OldSensor.SkeletonStream.Disable();
                }
                catch (InvalidOperationException)
                {
                }
            }

            if (args.NewSensor != null)
            {
                try
                {

args.NewSensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps
30);
                    args.NewSensor.SkeletonStream.Enable();

                    try
                    {
                        args.NewSensor.DepthStream.Range =
DepthRange.Default;

args.NewSensor.SkeletonStream.EnableTrackingInNearRange = true;
                    }
                    catch (InvalidOperationException)
                    {
                        args.NewSensor.DepthStream.Range =
DepthRange.Default;

args.NewSensor.SkeletonStream.EnableTrackingInNearRange = false;
                    }
                }
```

```
                catch (InvalidOperationException)
                {
                }
            }
        }

        #endregion

        //Botón regresar
        private void KinectTileButton_Click_3(object sender,
RoutedEventArgs e)
        {
            BindingOperations.ClearBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty);
            this.sensorChooser.KinectChanged -=
SensorChooserOnKinectChanged;
            (Application.Current.MainWindow.FindName("_mainFrame") as
Frame).Source = new Uri("MainMenu.xaml", UriKind.Relative);

        }
    }
}
```