Universidad Autónoma de Querétaro
Facultad de Ingeniería
Campus San Juan del Río

Desarrollo de interfaz gráfica de usuario para análisis postural mediante sensor TOF Kinect

TESIS

Que como parte de los requisitos para obtener el título de
Ingeniero Electromecánico linea terminal Mecatrónica

Presenta

Daniel Jaramillo Quintanar

No. de expediente: 218387

Director: M. en C. Marco Antonio Garduño Ramón

Co-asesor: Dr. Luis Alberto Morales Hernández

San Juan del Río, Qro., Mayo de 2016.

Resumen

La confiabilidad de los análisis de postura generalmente recae en la experiencia de los especialistas del área de fisioterapia, los cuales emplean un método tradicional basado en observaciones, así como, el uso de ciertos instrumentos como son la plomada y el posturómetro. Existen herramientas que emplean diferentes sensores para complementar dichos diagnósticos y volverlos algo más cuantitativo, sin embargo, sus costos son elevados. Existe la necesidad de desarrollar una herramienta propia la cual sirva de auxiliar para realizar análisis posturales a un bajo costo, que sea además fácilmente transportable y que su implementación en zonas marginadas no implique grandes dificultades. En este trabajo se presenta la metodología para desarrollar un sistema mediante un sensor tiempo de vuelo (TOF por sus siglas en inglés) Kinect, de bajo costo, el cual es un dispositivo que incorpora algoritmos para detectar automáticamente hasta 20 articulaciones del cuerpo humano. Dicha información permite implementar algoritmos matemáticos para realizar análisis de postura basado en criterios angulares establecidos en manuales de fisioterapia. Actualmente, el sistema se especializa en la detección de rodillas en valgo y en varo, clasificando el nivel de problema como ligero o crítico. El sistema permite generar un reporte de diagnóstico el cual incluye: información general del paciente, así como, de los problemas detectados denotados mediante información numérica, imágenes donde se evidencia el problema y, finalmente, anotaciones realizadas por el examinador al momento del estudio.

(Palabras clave: Análisis Postural, Interfaz Gráfica de Usuario, Time-Of-Flight, Kinect.)

Dedicatorias

A mis padres Joel Jaramillo Zamudio y Gloria Quintanar Olvera,

por ser un símbolo de esfuerzo y trabajo duro,

así como por otorgarme todo su cariño y

apoyo a lo largo de mi vida y

carrera estudiantil.

Agradecimientos

A mi familia por brindarme su paciencia, cariño, y constante apoyo durante mi desarrollo como persona, además de haber compartido día a día conmigo su sabiduría y experiencias de la vida.

A todos mis amigos que durante el transcurso de la carrera me apoyaron, y compartieron conmigo parte de sus vidas, convirtiéndose en parte importante de la mía.

A mis asesores el Dr. Luis Alberto Morales Hernández y el M. en C. Marco Antonio Garduño Ramón, por haber compartido parte de su conocimiento conmigo, apoyándome en todo momento durante este proyecto, al grado de crear también un vínculo de mutua amistad y respeto.

Al Dr. Juan Primo Benítez y al Dr. Jesús Rooney Rivera Guillen por haberse mostrado tan pacientes y haberme brindado su apoyo durante el desarrollo del proyecto.

A la Universidad Autónoma de Querétaro por haberme brindado tanto los materiales, como la instrucción apropiada durante el transcurso de mi carrera.

Agradezco por la beca y financiamiento de este trabajo a CONACYT (134481) y proyectos FOPER de la Universidad.

A todas aquellas personas que haya omitido dentro de esta lista, pero que sin los momentos que he compartido con ellas no hubiera podido llegar a ser quién soy hoy en día.

Índice general

Índice de figuras

Capítulo 1

Introducción

En la actualidad, para lograr resultados óptimos, la sociedad ha optado por utilizar las herramientas proporcionadas por la tecnología. Algunas áreas beneficiadas por los avances tecnológicos han sido: la industria, la biología, la astronomía, e inclusive dentro del área social como lo son la arqueología, entre muchas otras. Algunas de las áreas las cuales han mostrado mayor avance gracias a la tecnología son las ramas de la medicina y enfermería. En ellas, gracias a herramientas, en su mayoría automatizadas, como rayos X, electrocardiogramas, ultrasonidos, aparatos de rehabilitación mediante choques eléctricos, etc., han logrado facilitar los diagnósticos y procedimientos, así como poder realizarlos de una manera más precisa y confiable. Una de las herramientas de análisis y diagnóstico más populares es el uso de imágenes digitales y, más aún, el uso de técnicas de procesamiento de imágenes para evidenciar algún problema de salud. Una de las aplicaciones dadas al procesamiento de imágenes es para realizar los llamados análisis de postura o posturales. Estos análisis, anteriormente, se realizaban por medio de una simple inspección visual llevada a cabo por un especialista, el cual se basaba principalmente en su visión y experiencia. No se contaba con una demostración cuantitativa. Sin embargo, con el apoyo de sensores ópticos y aplicaciones de software se ha logrado cuantificar desequilibrios y fallas en la postura. En el presente trabajo se lleva a cabo el desarrollo de una interfaz de usuario para realizar análisis de postura mediante el uso de un sensor TOF (Time of Flight) Kinect.

1.1. Antecedentes

El procesamiento de imágenes obtenidas por medio de un sensor Kinect ha sido ampliamente utilizado en la actualidad debido a las ventajas de trabajar con un sensor de costo accesible, con herramientas de desarrollo libres y disponibles en la red, así como con características técnicas lo suficientemente buenas para obtener resultados de gran calidad. Es posible encontrar tanto a nivel internacional como nacional una gran cantidad de trabajos relacionados con el uso de las herramientas del Kinect, por ejemplo: en robótica móvil, interfaces sin mando físico, sistemas de procesamiento general de imágenes, etc. De manera local dentro de la Universidad Autónoma de Querétaro (UAQ) también se han realizado trabajos relacionados con este sensor.

Yeh et al. (2012), desarrollaron un software en el cual se apoya la rehabilitación de pacientes con problemas tanto físicos como emocionales, para lograr esto utilizaron las herramientas brindadas por el sensor Kinect en el cual hacen un análisis de los movimientos realizados por el paciente al seguir una serie de instrucciones, así como al interactuar con el sensor por medio de comandos de voz. Diego et al. (2012), realizaron una comparación de la esqueletización realizada por un sensor Kinect con respecto a otros sistemas utilizados en el medio, analizando su exactitud y precisión, así como su repetibilidad, demostrando que entrega mediciones muy acertadas por lo cual dijeron puede ser utilizado para diversas aplicaciones relacionadas con la postura de un sujeto. Borghese et al. (2012), realizaron un sistema basado en el sensor Kinect el cual apoya a la rehabilitación de pacientes, este es visto más bien como una aplicación ya que solo se necesita de un sensor Kinect de venta libre, un Xbox 360 y un monitor o televisor por lo que esta al alcance de todos y cumple con su principal objetivo que es el que el paciente cuente con una herramienta para su rehabilitación en casa. Garrido Navarro. et al. (2013), desarrollaron un sistema de ayuda a las personas con desorden en balance de su cuerpo, en este sistema se utiliza a el sensor Kinect para capturar la imagen y procesarla dándole al usuario una serie de movimientos a seguir y entregándole a la salida su mejora con respecto a las anteriores mediciones, también desplegando la medida del desorden de postura en el que se encuentra. Fernández et al. (2012), demostraron que, aunque el sensor Kinect muestra una menor precisión que otros sistemas ópticos de captura de movimientos, presenta

muchas ventajas como lo son su fácil traslado y programación, así como su bajo costo, además de que, aunque la precisión es menor, es suficientemente buena para realizar trabajos que tengan que ver con posturas de cuerpos físicos. Lange et al. (2011), en este trabajo se realizaron algunas pruebas del uso del sensor Kinect como apoyo en la rehabilitación de pacientes, se observaron algunas ligeras desventajas como interrupciones por el ambiente y también concluyeron con que el sensor podía ser de gran ayuda ya que muestra una esqueletización, así como la facilidad de ser comandado por voz y a su vez mostrar imágenes y/o pasos a seguir por el usuario. Gama et al. (2012), realizaron un software guía de apoyo para la buena ejecución de movimientos en una rehabilitación terapéutica, en la cual se basa con las lecturas realizadas por el sensor Kinect, con el que determina si los movimientos se realizan correctamente, si no es así, el sistema da consejos al usuario para una mejor ejecución, con esto permite una rehabilitación óptima y más rápida, así como también previene posibles lesiones por malas ejecuciones. Lange et al. (2012), en este trabajo se llevo a cabo un estudio de la manera en que se comporta el sensor Kinect y lo eficaces que son sus mediciones para poder determinar si se podía utilizar para fines relacionados con la rehabilitación de pacientes, determinaron que gracias a su facilidad de realizar diversos tipos de aplicaciones con una gran variedad de herramientas resulta ser de mucha utilidad en este medio. Chien-Yen et al. (2012), realizaron una comparación entre la eficacia de las medidas realizadas por el sensor de bajo costo de Microsoft Kinect contra Optitrack, que es un sistema reconocido con una fiabilidad elevada, encontrándose con resultados sumamente agradables ya que se demostró que el sensor Kinect entrega medidas con muy buena resolución.

Díaz et al. (2012) realizaron, dentro de la UAQ, una comparativa entre dos aplicaciones enfocadas al reconocimiento de patrones en vídeo para el control virtual de un mouse, una desarrollada bajo la aplicación de software libre Processing y la otra mediante C# con el SDK (Software Development Kit) de Microsoft para Kinect. Implementaron un sistema para la detección del usuario y asignan funciones del mouse a los gestos obtenidos mediante el rastreo de sus manos. Mencionaron que debido a que el SDK de Microsoft es de uso específico para el Kinect, éste puede sacar todo el provecho de dicho hardware para desarrollar una gran cantidad de aplicaciones, aunque esto no es una limitante total para no hacer uso de software libre. De León et al. (2012), llevaron a cabo una comparativa para determinar el potencial del sensor Kinect contra medios tradicionales para el

control de un vehículo móvil. El control de la dirección del vehículo se realiza mediante un volante virtual que son las manos del usuario simulando a éste. Mediante el Kinect se detecta la posición de estas y se determina la dirección y la velocidad del vehículo. Si bien acostumbrarse a esta nueva forma de control es difícil al principio, basta un rato de pruebas para lograr resultados similares a los obtenidos contra un sistema de control físico tradicional. Garduño et al (2014), desarrollaron una investigación acerca de la manera en la que se obtienen las coordenadas de las articulaciones adquiridas por el sensor Kinect y las plasmó en una aplicación sencilla sin interfaz gráfica, en su sistema no llegaron a realizar la estimación de los ángulos formados entre articulaciones.

Como se puede observar el sensor Kinect ha sido ampliamente utilizado en el área de fisioterapia y rehabilitación, pero es a menudo solo aplicado como guía de pasos a seguir para el usuario, mostrando imágenes de lo que debe de hacer y haciéndole saber si su ejecución es la correcta. Sin embargo, en su mayoría no realizan un análisis detallado de las articulaciones, o bien no toman en cuenta los ángulos formados entre estas, lo cual podría servir para detectar otro tipo de problemáticas.

1.2. Descripción del problema

Los expertos del área de fisioterapia realizan su análisis en base a su experiencia y preparación personal. Tradicionalmente las observaciones realizadas son registradas en ciertos formatos establecidos en los cuales generalmente sólo se observa el dibujo de la silueta del cuerpo humano sobre el que se hacen las anotaciones pertinentes de los problemas detectados. Ambos aspectos generan un cierto grado de incertidumbre en el correcto diagnóstico de algún problema postural, donde además no se cuenta con un registro tangible y mucho menos información cuantitativa del problema detectado. Un análisis postural estático permite analizar el cuerpo humano ante los efectos de la gravedad y detectar aquellas des alineaciones articulares que puedan mermar la calidad de vida de los pacientes. Consiste en colocar a la persona por delante de una superficie cuadriculada llamada posturómetro que permita notar desequilibrio corporales para posteriormente registrar cada observación hecha en un formato específico. Sin embargo, el proceso se basa en demasía en la expe-

riencia del evaluador y llegan a existir diferencias en el diagnóstico de un mismo grupo de pacientes examinado por dos expertos diferentes. Además tradicionalmente este análisis no contempla la adquisición de imágenes que pudieran servir como auxiliar en el diagnóstico de problemas posturales al compartir dichas fotos con un experto con una mayor experiencia en caso de existir duda. Se han desarrollado sistemas y aplicaciones auxiliares para la realización de esta tarea que permiten llevarla a cabo de una manera semi-automática o completamente automática, algunas de ellas utilizan proyección de líneas láser como referencias sobre los miembros articulares de interés, mientras que otras hacen uso de uno o varios dispositivos de captura de imágenes y pueden o no requerir la colocación de referencias físicas como pelotas de icopor directamente sobre el cuerpo humano. Este último grupo de aplicaciones generalmente forman parte de un sistema más complejo que integra un sistema de visión (hardware) junto con un software propio que incluye ciertas rutinas de procesamiento de imágenes. Lamentablemente la implementación de estas herramientas auxiliares no siempre es posible dados sus requerimientos específicos de instalación, como la necesidad de acondicionamiento de áreas destinadas para la colocación de los sistemas de visión o la necesidad de una alta capacitación del personal para su correcto uso. Otro aspecto importante a considerar es la nula posibilidad que tiene las aplicaciones de software de estos sistemas comerciales a ser modificados o actualizados por el usuario mismo para adaptarlos a sus necesidades o para implementar algoritmos de análisis propios. Finalmente, los altos costos que tienen es algo que influye para su utilización en zonas con cierto nivel de marginación.

1.3. Objetivos e Hipótesis

1.3.1. Hipótesis

Es posible desarrollar un sistema automático para análisis de postura mediante el uso de procesamiento de imágenes y sistemas de visión TOF de bajo costo.

1.3.2. Objetivo general

Desarrollar una interfaz gráfica de usuario que cumpla con los requerimientos solicitados por la gente del área de fisioterapia mediante herramientas de programación para desplegar la información de detección de articulaciones que entregan el sensor TOF Kinect y los algoritmos de análisis postural que sirva como herramienta auxiliar para lograr diagnósticos basados en información cuantitativa.

1.3.3. Objetivos particulares

- Establecer en conjunto con los investigadores de fisioterapia de la facultad de enfermería los requerimientos necesarios que debe llevar una interfaz de análisis postural.
- Definir las posturas estáticas a analizar.
- Implementar los algoritmos de análisis dimensional entre miembros articulares.
- Implementar los algoritmos de análisis de desequilibrio o desnivel entre miembros articulares.
- Implementar los algoritmos de análisis de orientaciones entre miembros articulares.
- Desarrollar un software con interfaz amigable para los usuarios.
- Documentar el desarrollo y modo de operación de la aplicación para generar un manual de usuario.

1.4. Justificación

La implementación de los sistemas de visión como herramienta auxiliar en este tipo de estudios de la postura mediante aplicaciones comerciales ha sido importante donde, además del respaldo fotográfico, se cuentan con algoritmos los cuales de manera semi-automática o automática llevan

a cabo el análisis postural entregando información numérica específica del problema detectado, todo esto de una manera cerrada y con pocas o nulas posibilidades de actualización o mejora por parte del mismo usuario. Hoy día es posible encontrar dichos sistemas en los grandes centros de salud, hospitales especializados y centros deportivos de alto rendimiento que pueden sortear sus altos costos. Sin embargo, socialmente aún hay un cierto grado de marginación y existe gente que nunca ha sido sometida a un análisis de la postura el cual permita, con los debidos cuidados y ejercicios de rehabilitación, mejorar su calidad de vida. El sensor Kinect ha significado una revolución tecnológica fomentando la creación de aplicaciones de toda índole gracias a las libertades de desarrollo dadas por Microsoft y ha venido siendo utilizado como herramienta auxiliar el en área de rehabilitación y fisioterapia para el diagnóstico de problemas relacionados con la postura y para tareas de rehabilitación debido a la función de detección automática de articulaciones que posee pero, sobre todo, por su bajo costo. Localmente, el sensor Kinect ha sido utilizado en aplicaciones generales de procesamiento de imágenes, de control de dispositivos y de interfaces gráficas, incluso ha habido un acercamiento puntual abordando su utilización para estimación de dimensiones entre miembros articulares del cuerpo humano, sin embargo, es necesario el desarrollo de una aplicación formal propia que conjunte e integre una mayor serie de herramientas encaminadas al análisis de la postura, la cual este bien documentada, programada estructuralmente para facilitar su actualización y escalamiento y la cual pueda ser implementada fácilmente.

1.5. Planteamiento general

En la Fig. 1.1 se muestra un diagrama a bloque del planteamiento general del proyecto.

Sensor Kinect: La primera parte de este proyecto se encuentra en obtener las imágenes desde el sensor Kinect, las cuales a continuación serán utilizadas para la medición y el análisis.

Conexión USB: Una de las ventajas del sensor Kinect es la facilidad de obtener su lectura mediante el protocolo de comunicación USB, el cual se comunicara a una PC en la que se procesaran las imágenes.

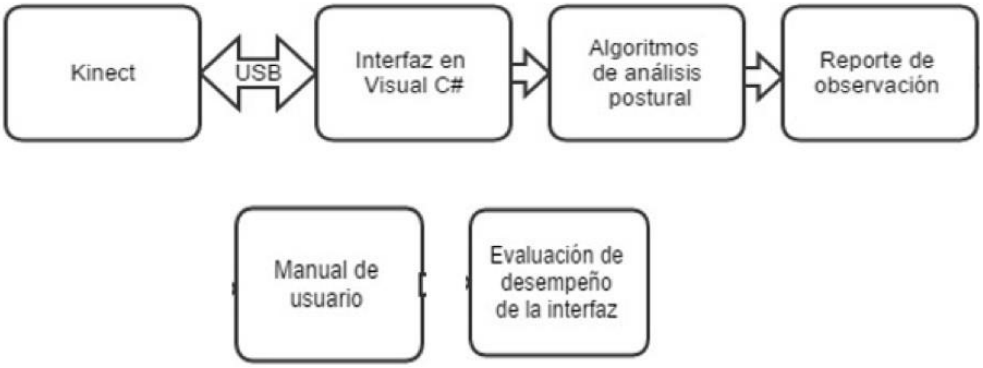

Figura 1.1: Planteamiento general.

Interfaz en Visual C#: Después de adquirir los datos por medio de la conexión USB, estos son utilizados en un programa de Visual C# en el que se realizaran las adecuaciones necesarias.

Algoritmos de análisis postural: En esta sección se realizan los algoritmos para encontrar las distancias y ángulos entre las articulaciones requeridas para realizar un análisis postural.

Reporte de observación: Se realizara un reporte en el que se plasmaran las observaciones realizadas a lo largo de las pruebas realizadas.

Manual de usuario: la elaboración de un manual de usuario es indispensable ya que en este se dará la serie de pasos para la correcta instalación y uso del software para futuros usuarios.

Evaluación de desempeño de la interfaz: Para finalizar se realizara una evaluación del desempeño que mostró la interfaz.

Capítulo 2

Revisión de la literatura

2.1. Estado del arte

En la actualidad los sensores son indispensables y una de las principales herramientas para el desarrollo de instrumentos tecnológicos, los cuales nos facilitan nuestra vida diaria, uno de esos sensores es el sensor Kinect que a pesar de que es de una muy reciente aparición, se han elaborado una gran cantidad de aplicaciones a partir de este. En el presente capítulo se presentan las herramientas necesarias para la realización del trabajo propuesto en esta tesis.

2.2. Sensores TOF

De acuerdo a Hansard et al. (2013), los sensores TOF son cámaras capaces de producir una imagen de profundidad donde cada uno de los píxeles de esta imagen codifica la distancia hasta el punto correspondiente en la escena. Estas cámaras se pueden utilizar para estimar la estructura 3D directamente, sin la ayuda de algoritmos de visión por ordenador tradicionales. Hay muchas aplicaciones prácticas para esta nueva modalidad de detección, incluyendo la navegación de robots, la reconstrucción 3D, y la interacción hombre-máquina Los sensores TOF funcionan midiendo el retardo de fase de la reflexión de luz infrarroja (IR). Esta no es la única manera de estimar la

profundidad; por ejemplo, un patrón de luz IR estructurada se puede proyectar en la escena, a fin de facilitar la triangulación visual. Los dispositivos de este tipo, como el Kinect, comparten muchas aplicaciones con cámaras TOF. En la Fig. 2.1 se ilustra el principio de funcionamiento del sensor de profundidad TOF. Una onda IR de color rojo se dirige al objeto de destino, y el sensor detecta la componente IR del reflejo.

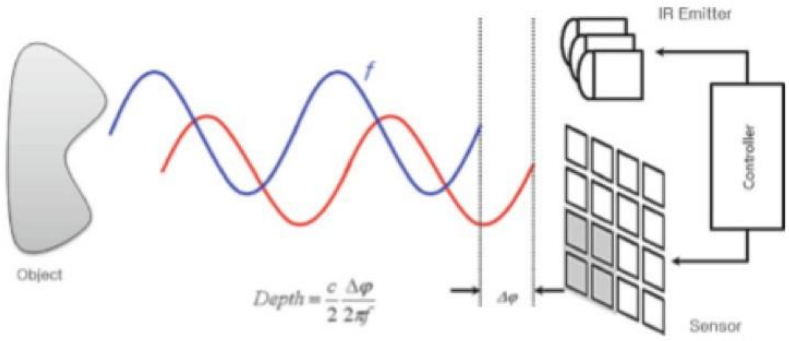

Figura 2.1: El principio de funcionamiento del sensor de profundidad TOF.

Mediante la medición de la diferencia de fase entre las ondas IR enviadas y reflejadas, es posible calcular la distancia al objeto. La diferencia de fase se calcula a partir de la relación entre cuatro valores diferentes de carga eléctrica como se muestra en la Fig. 2.2.

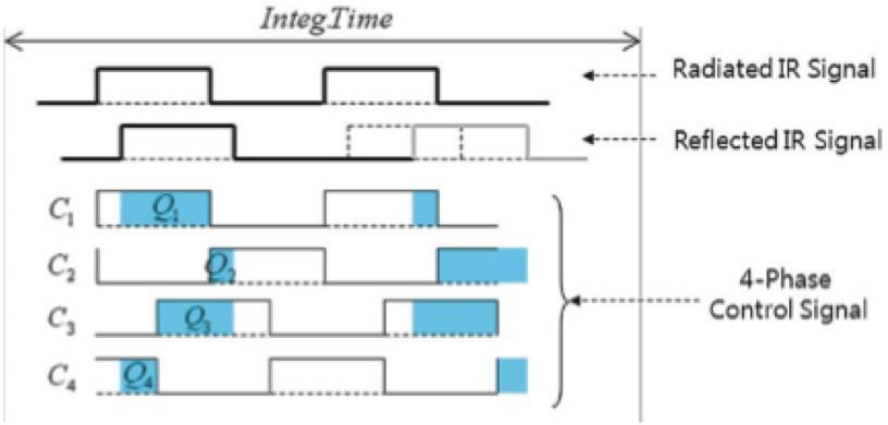

Figura 2.2: La profundidad puede ser calculada midiendo el retardo de fase entre señales IR emitida y reflejada.

Las cuatro señales de control de fase tienen retardos de fase 90 grados uno del otro, determinan

la colección de electrones de la señal IR aceptados. Los cuatro valores de carga eléctrica resultantes se utilizan para estimar la diferencia de fase t_d como se muestra en la Ec. 2.1.

$$t_d = \frac{Q_3 - Q_4}{Q_1 - Q_2} \tag{2.1}$$

Donde Q1 a Q4 representan la cantidad de carga eléctrica para las señales de control C1 a C4, respectivamente. La distancia correspondiente d se puede calcular, usando la velocidad c de la luz y $\mathbf{f}$ la frecuencia de la señal, Ec. 2.2.

$$d = \frac{c}{2\mathbf{f}} \frac{t_d}{2\pi} \tag{2.2}$$

2.3. Sensor Kinect

De acuerdo a la página web de Microsoft http://kinectforwindows.org/ el sensor Kinect apareció en Noviembre de 2011 como un accesorio de la consola de vídeo juegos Xbox 360. Fue desarrollado por la compañía Prime Sense en colaboración con Microsoft. En Febrero de 2012 fue lanzada una versión especialmente desarrollada para el sistema operativo Windows 7. El sensor Kinect (Fig. 2.3) tiene una cámara RGB así como un emisor y sensor de luz infrarroja, en conjunto permiten capturar imágenes a color y entregar la información de profundidad de cada píxel en la escena (Cruz, 2012).

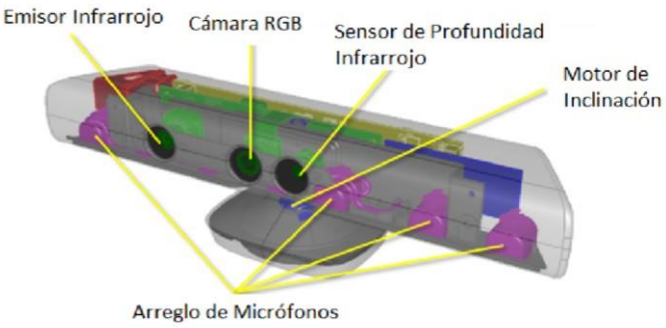

Figura 2.3: Elementos del sensor Kinect (Fuente: http://kinectforwindows.org/).

La cámara RGB opera a 30 Hz y entrega imágenes de 640x480 píxeles con 8 bits por canal. El campo de visión (FOV, por sus siglas en inglés) para la imagen a color es de 57 grados en horizontal y 43 grados en vertical. Es posible cambiar la resolución de la cámara a 1280x1024 píxeles pero corriendo a 10 cuadros por segundo. Por su parte, la cámara infrarroja opera a 30 Hz y entrega imágenes con 1200x960 píxeles, estas son disminuidas a 640x480 píxeles con 11 bits, con lo cual provee 2048 niveles de sensibilidad. El FOV del sistema infrarrojo es de también de 57 grados en horizontal y 43 grados en vertical. El rango de operación estándar es de entre 0,8 y 4 m, sin embargo el rango de operación adecuados para el sensor se encuentra entre 1,2 y 3,5 m de separación, así como como también abarca un ancho de alrededor de 3 m, Fig. 2.4.

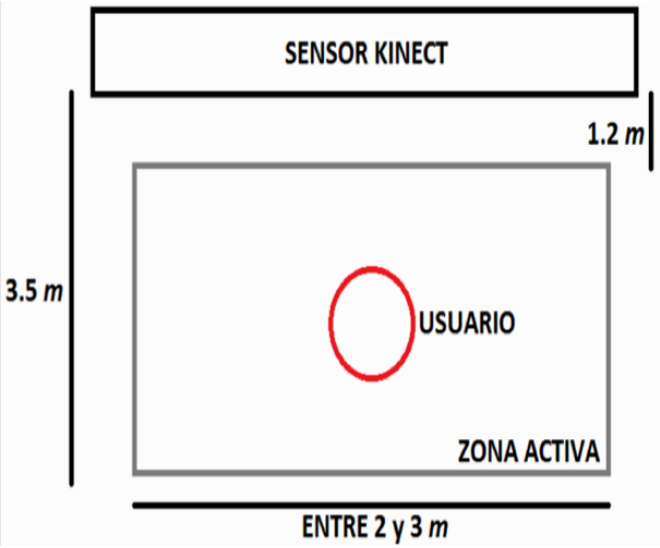

Figura 2.4: Zona activa del sensor Kinect.

La medición de profundidad se basa en un método de luz estructurada para medirla, un patrón de puntos conocido es proyectado por el emisor infrarrojo, el sensor infrarrojo los captura y compara con el patrón conocido. Cada píxel contiene la distancia cartesiana en mm desde el plano de la cámara hasta el objeto más cercano en esa coordenada particular (x, y), esto se muestra en la Fig. 2.5.

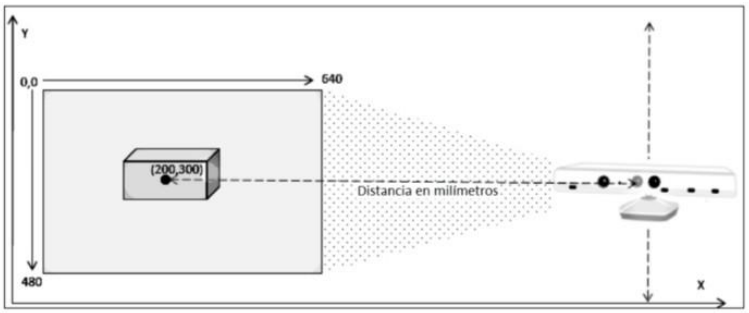

Figura 2.5: Medición de profundidad.

El sensor de profundidad cuenta con dos rangos, estos se muestran en la Fig. 2.6. El primero es conocido como rango de default (default range) y el otro como rango cercano (near mode), ambos están disponibles en el Kinect para Windows más no así para el de Xbox 360 el cual no cuenta con la información del rango cercano.

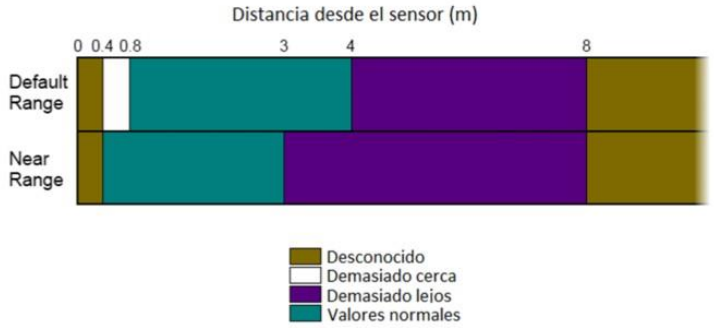

Figura 2.6: Rango del espacio de profundidad (Fuente: http://kinectforwindows.org/).

Probablemente la función más conocida de este dispositivo es la que tiene que ver con la detección de articulaciones de usuario, esta función es medular para el objetivo original del Kinect que es la de proveer control de vídeo juegos sin la necesidad de un mando físico. El dispositivo puede reconocer hasta 6 usuarios dentro del campo de visión del sensor, Fig. 2.7.

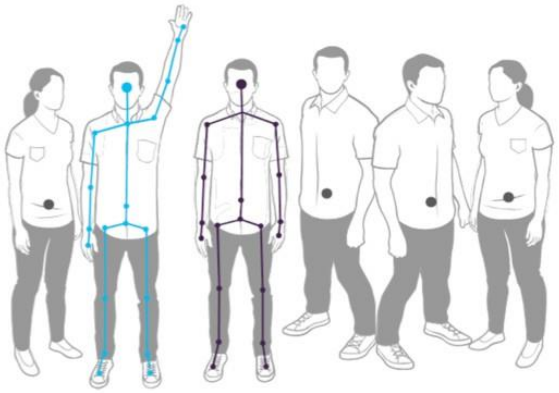

Figura 2.7: Detección de articulaciones con Kinect (Fuente: htto://kinectforwindows.org/).

Esta función del sensor permite monitorear 20 articulaciones de hasta dos usuarios, para esto, utiliza la información del sensor infrarrojo y de este modo estima la posición de las articulaciones, Fig. 2.8.

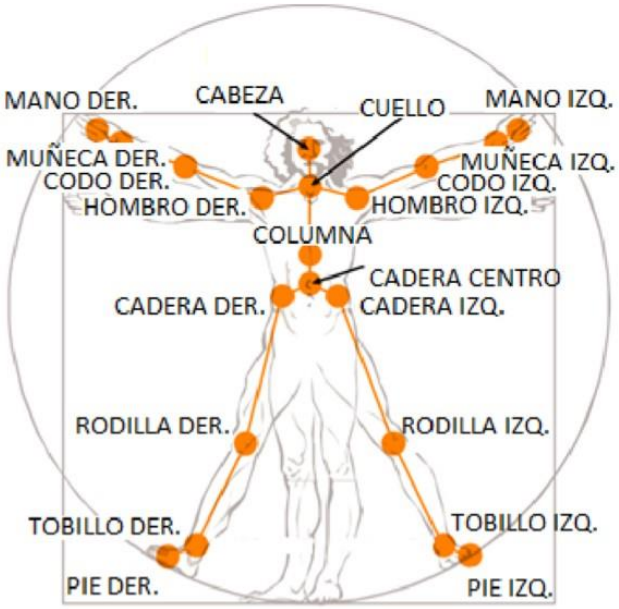

Figura 2.8: Articulaciones detectadas por el Kinect (Fuente: https://kinectforwindows.org/).

2.4. Análisis postural

De acuerdo a Peñaloza et al. (2013): “La postura bípeda ideal es aquella que permite un estado de equilibrio muscular y esquelético que protege estructuras corporales frente a lesiones o deformidades progresivas, independientemente de la posición en la que estas estructuras se encuentran, ya sea en movimiento o en reposo. En esta postura se requiere que exista una perfecta distribución de la masa corporal alrededor del centro de gravedad, que minimice los esfuerzos y las tensiones realizadas por el sistema de soporte a causa de la gravedad. Esta postura es el punto de referencia a la hora de detectar deficiencias posturales al observar la alineación corporal estática en posición de pie.”

Un formato para la estandarización de un análisis postural es el llamado formato FOSAC que significa: Formato de Observación Sistemática de la Alineación Corporal, en el que se pide al paciente tome cuatro posiciones a partir de las cuales se realizaran las observaciones, de frente al observador, de espaldas al observador, lateral derecho y lateral izquierdo. En la Fig. 2.9 se muestra el formato FOSAC en el que se pueden ver las cuatro posiciones que debe de tomar el paciente, así como también los puntos a analizar en cada una de las posiciones.

Este formato solo pide rellenar las columnas con las observaciones realizadas por el especialista, en algunas ocasiones el paciente es posicionado en frente de una pared cuadriculada, Fig. 2.10, para facilitar al especialista el poder observar o detectar algún desequilibrio o falla de postura. Para realizar un análisis de este tipo el especialista debe de observar detalladamente cada uno de los puntos en cuestión en el paciente y en algunos caso también recurre a palpar el cuerpo de este para así tener una mayor seguridad en su diagnóstico. Además de que también se le pide al paciente quitarse la ropa quedándose solo con sus prendas íntimas, ya que de esta manera es posible observar directamente la musculatura y la orientación ósea. Además también, en algunas ocasiones el especialista se apoya de “poner al paciente a plomo” como comúnmente se le conoce para así verificar la verticalidad de su postura.

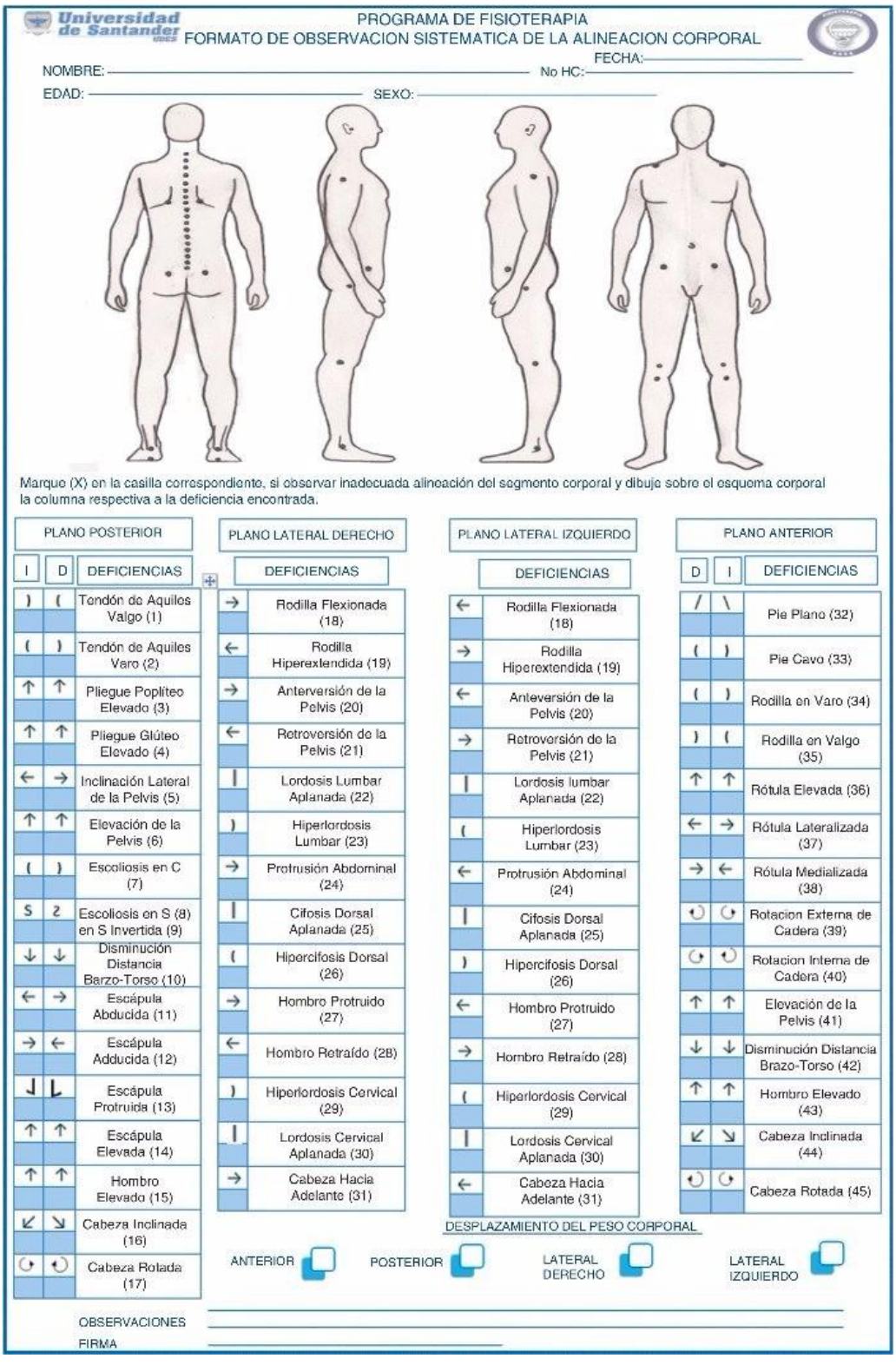

Figura 2.9: Formato de registro FOSAC.

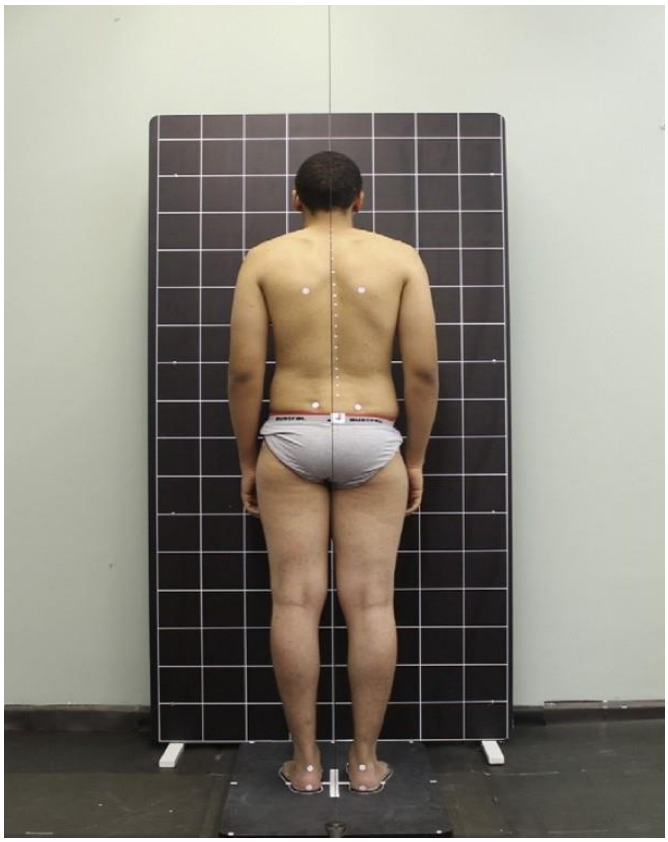

Figura 2.10: Paciente frente a pared cuadriculada.

2.5. Distancia euclidiana

La distancia euclidiana es la distancia ordinaria (que se mediría con una regla) entre dos puntos de un espacio euclidiano.

2.5.1. 2D

La distancia euclidiana se deduce a partir del teorema de Pitágoras al realizarse el análisis de las distancias entre dos puntos en el plano a partir de los vectores que los unen (Anton, 1994), como se muestra en la Fig. 2.11.

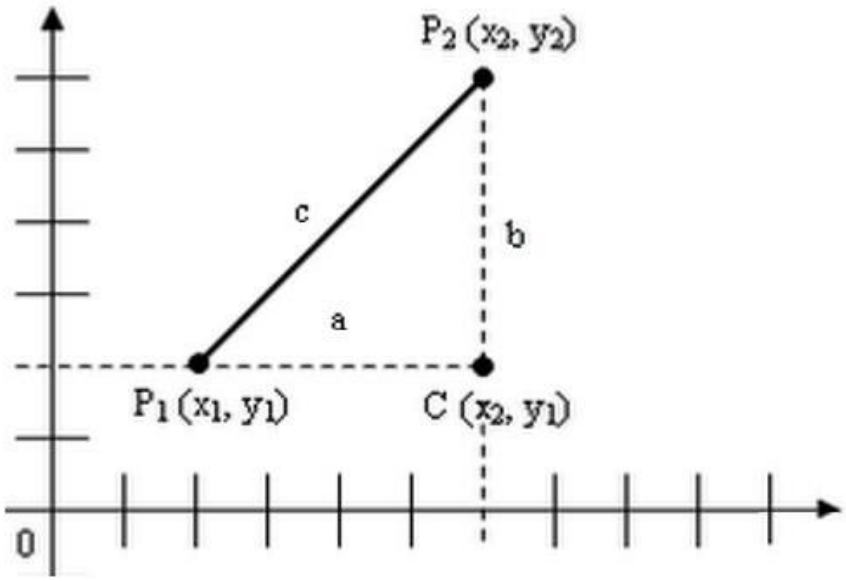

Figura 2.11: Distancia en un sistema de coordenadas cartesianas.

De lo que se obtiene la expresión con la cual se conoce la magnitud de la distancia entre dos puntos en un plano (2D) como se muestra en la Ec. 2.3.

Si P 1 = (x_1, y_1) y P 2 = (x_2, y_2), entonces la distancia entre el punto P 1 y P 2 en el plano es:

$$d(P_1, P\,2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.3}$$

2.5.2. 3D

De la misma manera en la que se obtuvo la distancia entre dos puntos para un plano, la ecuación se expande a la tercera dimensión, obteniendo la Ec. 2.4.

Si P 1 = (x_1, y_1, z_1) y P 2 = (x_2, y_2, z_2), entonces la distancia entre el punto P 1 y P 2 en el espacio es:

$$d(P_1, P\,2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.4}$$

2.6. Ángulos entre vectores

De acuerdo a Anton (1994), el ángulo (α) que forman dos vectores $u, v \in Rn$, no nulos, se obtiene de la igualdad: $u \cdot v = (||u|| \cdot ||v||) * \cos(\alpha)$, en la que se despeja el ángulo obteniendo la Ec. 2.5.

$$\alpha = \cos^{-1} \frac{u \cdot v}{||u|| \cdot ||v||} \tag{2.5}$$

Donde $||u||$ es la norma del vector u, o bien la magnitud de la distancia entre los extremos de este vector, como se muestra en la Ec. 2.6. Lo mismo sucede con $||v||$, Ec. 2.7. Con $u = (u_1, u_2, u_3)$ y $v = (v_1, v_2, v_3)$.

$$||u|| = \sqrt{(u_1)^2 + (u_2)^2 + (u_3)^2} \tag{2.6}$$

$$||v|| = \sqrt{(v_1)^2 + (v_2)^2 + (v_3)^2} \tag{2.7}$$

Obteniendo entonces una expresión general como se muestra en la ecuación 2.8.

$$\alpha = \cos^{-1} \frac{(u_1,u_2,u_3) \cdot (v_1,v_2,v_3)}{\sqrt{(u_1)^2 + (u_2)^2 + (u_3)^2} * \sqrt{(v_1)^2 + (v_2)^2 + (v_3)^2}} \tag{2.8}$$

2.7. Imágenes digitales

De acuerdo a Burger y Burge (2009) una imagen digital puede ser definida como una función bidimensional $\mathbf{f}(x, y)$; por lo que es necesario imponer un sistema de coordenadas, con lo que es posible saber qué posición de cada pixel corresponde a cada elemento de la imagen, con esto es posible hacer una medición de las distancias en la imagen real ya que existe una proporcionalidad entre las distancias reales y la distancia entre cada píxel referente a esa posición.

El sistema coordenado para el tratamiento de imágenes se encuentra un tanto en contra de las convenciones matemáticas normales, ya que en este sistema el eje "y" no arranca de abajo hacia arriba como es normalmente, sino más bien de arriba hacia abajo, y el eje coordenado "x" si se encuentra en la misma dirección, como se muestra en la Fig. 2.12.

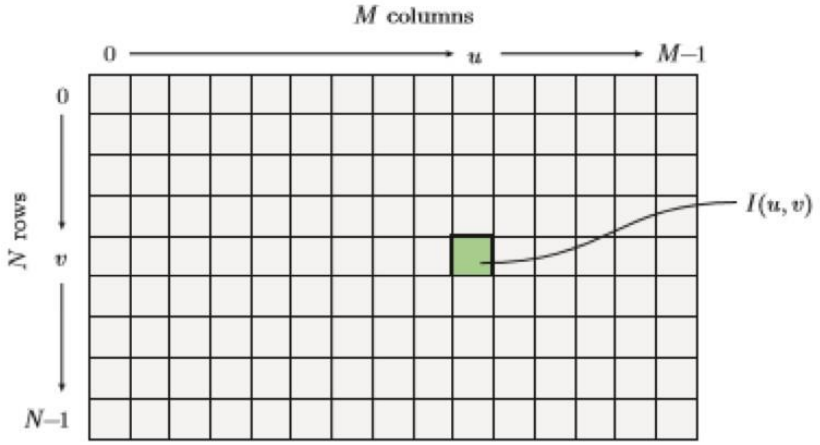

Figura 2.12: Sistema de coordenadas en imágenes.

En la Fig. 2.12 se puede observar que si se hace una analogía con el plano coordenado común, el eje (x) está representado por la letra u y el eje y por la letra v, además se puede observar también que contiene M columnas y N filas, teniendo así una matriz con dimensiones M xN , sin embargo la numeración inicia desde cero en el punto $(u = 0, v = 0)$ por lo cual el máximo número de columna es: $u_{max} = M - 1$, y el máximo número de filas es: $v_{max} = N - 1$. Con lo anterior se establece que una coordenada en el plano de la imagen esta representada por $\mathbf{f}(u, v)$.

De acuerdo a lo anterior es posible encontrar la distancia entre dos píxeles en la imagen lo que es directamente proporcional a la distancia entre los dos puntos mostrados de la imagen, por lo cual es necesario encontrar estas distancias dentro de la imagen digital. Para encontrar dicha distancia se puede utilizar la Ec. 2.9 explicada anteriormente con la que es posible calcular la distancia entre dos puntos cualesquiera dentro de un plano coordenado.

$$d(P_1, P\,2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.9}$$

El método mencionado es ampliamente utilizado, pero el sensor Kinect también nos entrega la información de profundidad de la escena teniendo así información de un espacio tridimensional (3D). Para analizar las distancias entre puntos en el espacio se utiliza la Ec. 2.10 con la que es posible determinar la distancia entre dos puntos cualesquiera en el espacio coordenado.

$$d(P_1, P\,2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.10}$$

2.8. Evaluación heurística

La “evaluación heurística” (HE, por sus siglas en inglés) es una técnica utilizada para analizar la facilidad de uso en las primeras etapas del diseño de interfaces de usuario. Se desarrolló en Dinamarca hace diez años (Molich y Nielsen, 1990) con el objetivo de crear un método para analizar el diseño de interfaces de una manera informal, manejable y fácil de enseñar. La técnica se ha investigado, se han refinado las heurísticas, resultando en una técnica bien establecida (Nielsen, 1993).

Las “heurísticas” HE, o los principios de usabilidad, se han establecido con la práctica en el diseño y evaluación de interfaces de usuario. Esto es, contienen una compilación de prácticas de buen diseño y fallas en diseño conocidas que han surgido en el campo del desarrollo de interfaces de usuario en los últimos treinta años. Para llevar a cabo una evaluación heurística es necesario que varias personas analicen el diseño de interfaz de usuario para verificar si infringe alguna de las

heurísticas. Si infringe alguna heurística es necesario modificar el diseño.

El diseño puede estar en cualquier etapa de desarrollo del producto. Puede ser tan preliminar como un borrador a lápiz del sistema, un prototipo parcial en Visual Basic, o un sistema operacional completo. Sin embargo, conforme el sistema esté más avanzado, más caro costará arreglar los problemas de usabilidad - por lo tanto la HE tiene más valor en los primeros borradores o prototipos.

De manera general, varias personas harán evaluación HE de un diseño porque las investigaciones han demostrado que diferentes personas encontrarán diferentes infracciones. En un sistema sencillo (como el de un cajero automático que permite aproximadamente 6 funciones diferentes), probablemente cuatro o cinco examinadores serán suficiente para encontrar problemas de utilidad (Nielsen, 1993). Para sistemas más complicados se necesitan un número mayor de personas que evalúen el sistema (Slavkovic y Cross, 1999). Sin embargo, debido a que es una técnica muy simple de aprender a usar, no es difícil encontrar miembros del equipo de desarrollo que realicen la evaluación. Además, con que una o dos personas evalúen el sistema se encontrarán errores de usabilidad que una vez corregidos mejorarán el uso del sistema que estas diseñando.

En los siguientes incisos se encuentran nueve heurísticas enumeradas en Nielsen (1993) y una (la última) es de Nielsen y Mack (1994).

- Visibilidad del estatus del sistema.

  La computadora debe mantener al usuario informado de lo que está ocurriendo, las entradas que ha recibido la computadora, el proceso que se está llevando a cabo y los resultados del procesamiento. La computadora debe proveerle información al usuario de manera visual o a través de algún sonido. Si no es así, el usuario no tiene la información necesaria para entender lo que está sucediendo.

- Comparación entre el sistema y el mundo real

  El diseño de la interfaz del usuario debe utilizar los conceptos, lenguaje, y las convenciones del mundo real que sean familiares para el usuario. Esto significa que el programador del sistema debe comprender la tarea que el sistema debe desempeñar desde el punto de vista del

usuario. Los aspectos culturales se hacen relevantes para el diseño de sistemas computacionales globales. Los conceptos familiares, el idioma, y las convenciones del mundo real de la interfaz pueden hacer uso del conocimiento en la memoria a largo plazo del usuario por su experiencia con la tarea (independiente de la computadora).

- Control y libertad del usuario

  Permite que el usuario tenga control de la interacción. Los usuarios deben poder corregir sus acciones fácilmente, salirse de la interacción rápidamente en cualquier momento, y no verse forzados a hacer una serie de acciones controladas por la computadora. Los usuarios cometerán errores y por lo tanto, necesitan una opción fácil para corregirlos.

- Estándares y consistencia

  La información que es igual debe verse igual (mismas palabras, iconos y posiciones en la pantalla). La información que es diferente debe ser expresada de manera diferente. Tal consistencia debe mantenerse dentro de la aplicación y de la plataforma. Los programadores deben conocer las convenciones de las plataformas. Así como la heurística de relación entre el sistema y el mundo real, esta heurística debe hacer uso del conocimiento previo del usuario y la experiencia con otras secciones de la misma aplicación y con otras aplicaciones en la misma plataforma.

- Prevención de errores

  El sistema debe prevenir que se cometan errores tanto como sea posible. Por ejemplo, si hay una cantidad limitada de acciones legales en una parte de la aplicación, ayuda a los usuarios a seleccionarlas en lugar de permitir que lleven a cabo una acción para luego decirles que cometieron un error. Esto se puede considerar como una parte de la primera heurística, Visibilidad del Estatus del Sistema que entradas son aceptables para el sistema computacional), pero debido a que es tan importante y tan frecuentemente violada, merece su propia heurística. Los usuarios pueden cometer errores debido a fallas en la percepción, a falta de conocimiento del paso siguiente, a solo recordar la esencia del comando y no todos los detalles, o un error al teclear o apuntar. Algunos de estos errores pueden se pueden prevenir al

mostrar solo las acciones que son aceptables en un punto particular de interacción (al sombrear los botones inapropiados). Otros errores pueden ser detectados tan pronto el usuario lo cometa (no aceptar las abreviaciones incorrectas de un estado de la república en una forma de direcciones).

- Reconocer en vez de recordar

  Mostrarle al usuario todos los objetos y acciones disponibles. No le pidas al usuario que se acuerde de la información de otra pantalla de la aplicación. Esta heurística es una aplicación directa de las teorías de la memoria humana. Es más fácil para alguien reconocer lo que tiene que hacer si existen ciertas pistas en el ambiente que le lleguen a la memoria de trabajo a través de la percepción y a través del conocimiento almacenado en la memoria de largo plazo.

- Flexibilidad y eficiencia en el uso

  El diseño debe contar con atajos en tecleado que permita que los usuarios expertos puedan acelerar su interacción (en lugar de siempre usar los menús e iconos con un ratón). Los usuarios expertos deben de poder adaptar la interfaz para que las acciones frecuentes se hagan con mayor rapidez. El valor de los atajos proviene principalmente de los procesos motores del usuario. Por lo general es más rápido teclear que cambiar la mano continuamente entre el tecleado y el ratón y apuntar a objetos en la pantalla. Además, los usuarios expertos desarrollan planes de acción, las cuales querrán utilizar frecuentemente, así que al personalizar pueden capturar sus planes en la interfaz.

- Diseño estético y minimalista

  Elimina los elementos irrelevantes de la pantalla que sólo representan desorden y confusión para el usuario. Esta heurística se relaciona con el aspecto de percepción de búsqueda visual y de memoria. Mientras exista más amontonamiento, hay más información en la que se tiene que hacer la búsqueda para encontrar la información deseada. Adicionalmente, mientras más información entre por la percepción al hacer la búsqueda visual, más información interfiere con la búsqueda en la memoria de largo plazo.

- Como ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores

  Los mensajes de error deben ser escritos en un lenguaje simple, deben explicar el problema dar consejo constructivo de como corregir el error. Una vez más, esto se puede considerar parte de la primera heurística, visibilidad del estatus del sistema, pero es tan importante y violada tan frecuentemente que merece su propia heurística. Es necesario darle al usuario suficiente información para que comprenda la situación.

- Ayuda y documentación

  Si el sistema no es extremadamente sencillo, va a ser necesario que cuente con sus opciones de ayuda y con una documentación adecuada. La ayuda y documentación debe estar siempre disponible, fácil de buscar y debe aportar consejos directos que sean aplicables a las tareas del usuario. Una vez más, esto es para dar al usuario suficiente información para comprender la aplicación. La característica de búsqueda debe permitir que se encuentre información al preguntar por la esencia del significado en lugar de la palabra clave exacta, porque las personas recuerdan la esencia y no las palabras exactas (si es que acaso lo supieron alguna vez).

Capítulo 3

Metodología

En este capítulo se presentan los pasos implementados para la realización de este trabajo de tesis. La metodología utilizada es concentrada en el diagrama a bloques que es mostrado en la Fig. 3.1. En este diagrama se muestra de forma general la manera en la que se desarrolla el proyecto, mostrando bloques por cada una de las tareas a realizar.

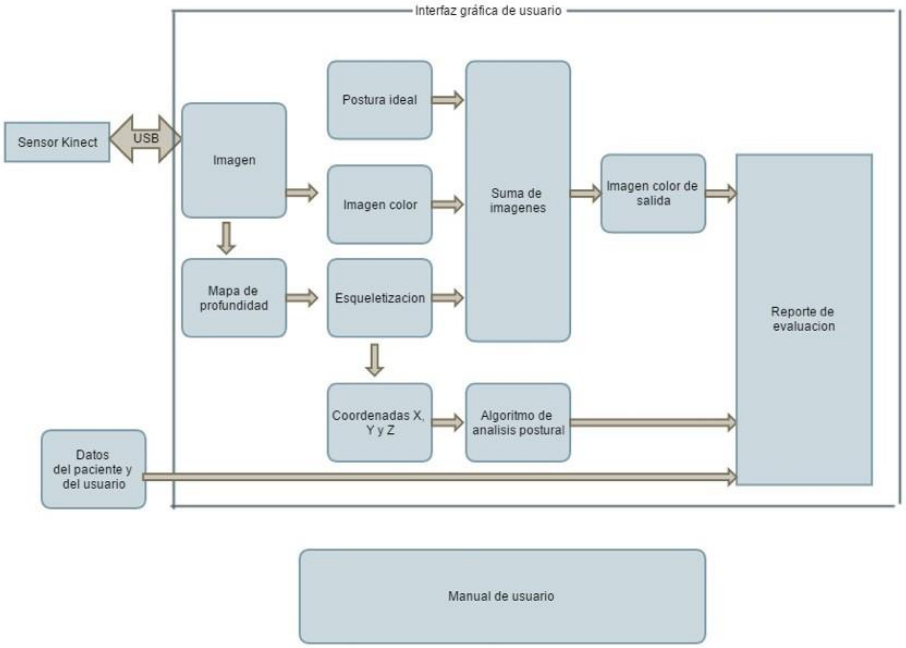

Figura 3.1: Diagrama a bloques de la metodología propuesta.

3.1. Instalación y configuración del sensor Kinect

Se siguieron los pasos de instalación y configuración del sensor Kinect dados por Garduño et al. (2014) para realizar la aplicación propuesta. Estos pasos se muestran en el diagrama a bloques de la Fig 3.2.

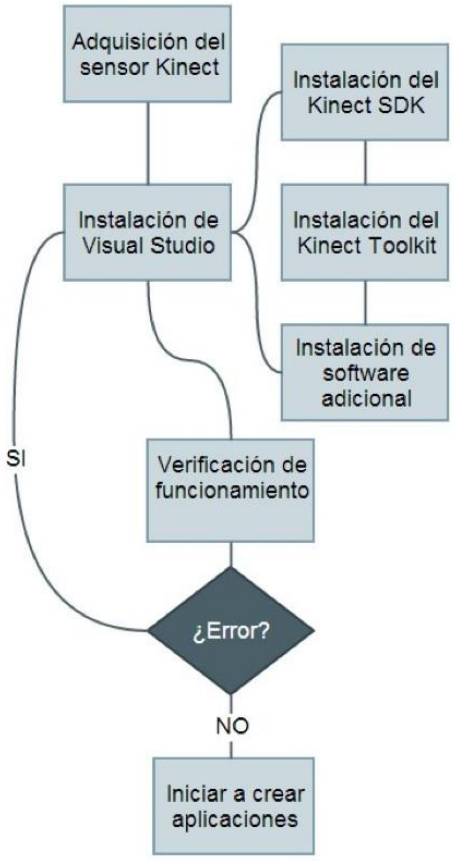

Figura 3.2: Diagrama a bloques de la instalación del sensor Kinect.

Adquisición del sensor Kinect: El sensor Kinect para Windows incluye el sensor Kinect y un cable "Y" de alimentación con conexión USB, dichos elementos son mostrados en la Fig. 3.3.

Figura 3.3: Elementos que contiene el sensor Kinect para Windows.

Instalación del IDE de desarrollo Visual Studio: Para el desarrollo de aplicaciones de Kinect es necesario descargar e instalar el ambiente integrado de desarrollo (IDE, por sus siglas en inglés) de Visual Studio la cual puede ser a partir de su versión 2013 en adelante.

Instalación del SDK de Kinect para Windows: En aras de lograr la completa funcionalidad del sensor Kinect se deben descargar e instalar los controladores y demás elementos de software que proporciona el mismo Microsoft a través de su página web www.kinectforwindows.com. El primer paquete de software que Microsoft recomienda instalar es el Kinect for Windows SDK. En este se incluyen todos los drivers necesarios para el funcionamiento de la cámara de vídeo, del emisor y sensor infrarrojo, de los micrófonos, del acelerómetro así como del motor de orientación vertical del dispositivo.

Instalación del conjunto de herramientas para desarrollador del sensor Kinect: A continuación se debe instalar el Kinect for Windows Developer Toolkit (conjunto de herramientas para desarrollador windows para Kinect). Aquí se incluyen ejemplos básicos de uso del sensor Kinect y de programación desarrollados en Visual Studio abarcando los lenguajes Visual Basic, Visual C++ y Visual C#.

Instalación de software opcional: Dado el caso de que solamente se requiera ejecutar una aplicación de Kinect en una PC, como por ejemplo en un equipo de un usuario final o similar, basta con instalar el Kinect Runtime el cual únicamente incluye los drivers del sensor Kinect así como el

ambiente de tiempo de ejecución necesarios para ejecutar dichas aplicaciones.

Verificación de funcionamiento: Una vez instalado el software anterior, se debe conectar el sensor Kinect a la alimentación y posteriormente a la PC por medio de cable USB. La primera vez que se conecte el sensor la PC detectara el dispositivo e iniciara la configuración de drivers, cuando todo el proceso termine se verá una ventana como la de la (Fig. 3.4). Además, el sensor Kinect deberá mostrar el led que tiene en la parte frontal iluminado de color verde sin parpadear.

Figura 3.4: Primera conexión del sensor Kinect a la PC.

En caso de que dicha ventana no aparezca o el led frontal del sensor se ilumine de color rojo se debe verificar en el administrador de dispositivos de Windows que los diferentes componentes del Kinect hayan sido reconocidos e instalados correctamente, ver Fig. 3.5. En caso afirmativo se recomienda desconectar el sensor, reiniciar la computadora y repetir los pasos anteriores. En el caso contrario y en el de no lograr un funcionamiento correcto del sensor se debe desconectar el sensor y desinstalar todo software relacionado con el sensor Kinect, reiniciar la computadora y volver a iniciar con la instalación del software. En un caso último en que bajo ninguna circunstancia se logre echar a andar el dispositivo, se debe probar en otra PC y descartar un mal funcionamiento del sensor o de la misma computadora y, en un caso extremo, solicitar soporte al distribuidor del sensor Kinect y al fabricante.

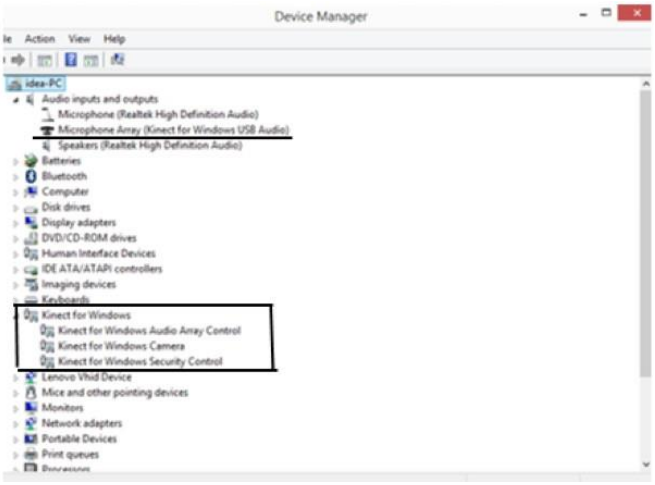

Figura 3.5: Dispositivos instalados correctamente.

3.2. Aplicación básica para sensor Kinect

De nueva cuenta, se siguieron los pasos dados por Garduño et al. (2014) para generar una aplicación básica, haciendo uso del sensor Kinect y el lenguaje de programación Visual C#. Estos pasos se muestran en el diagrama a bloques de la Fig. 3.6.

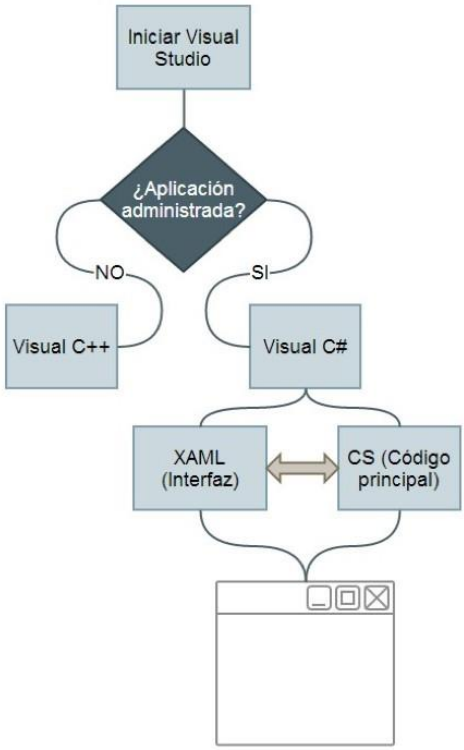

Figura 3.6: Pasos para generar una aplicación que use el sensor Kinect.

Los pasos para crear la aplicación que haga uso del sensor Kinect se enumeran a continuación:

1. Iniciar Visual Studio 2010, 2012, ó 2013.
2. Seleccionar new project.
3. En templates, seleccionar el lenguaje de programación Visual C#.
4. En el menú de apps de C# seleccionar WPF Application.
5. Indicar un nombre para el proyecto.
6. Dar click en OK.
7. Agregar la referencia Microsoft.Kinect.dll.
8. Diseñar la interfaz en XAML:

- Agregar ventanas para mostrar imagen color, imagen de profundidad o de usuario esqueletizado según se necesite.
- Agregar botones y demás controles WPF según sea necesario.

9. En el código cs:

   - Iniciar Kinect.
   - Habilitar flujo de información color, profundidad, etc.
   - Convertir la información de color, de profundidad, etc., a mapa de bits (BMP) para desplegar en la interfaz.
   - Almacenar información de color, de profundidad o la que se requiera.
   - Deshabilitar flujo de información color, profundidad, etc.
   - Detener Kinect.

3.3. Adquisición imagen color

Para adquirir la imagen color es necesario primero haber realizado todos los pasos de los dos puntos anteriores. Una vez instalado correctamente se prosigue a utilizar algunas de las herramientas del sensor, que en este caso es la adquisición de imagen color. En la Fig. 3.7 se muestra la herramienta de imagen color del Kinect en funcionamiento, la cual nos entrega imágenes de 640x480 píxeles. La imagen obtenida será la base para montar sobre ella la esqueletización y las correcciones realizadas por la interfaz, así como también al ser un mapa de píxeles, funciona como plano coordenado como se explico anteriormente.

Figura 3.7: Imagen color obtenida con el sensor Kinect.

3.4. Obtener coordenadas del esqueleto

El proceso de obtener las coordenadas del esqueleto es tal vez la parte más importante en este proyecto, ya que a partir de ellas se pueden aplicar los algoritmos de postura para a partir de estos generar un reporte de evaluación. Para poder obtener las coordenadas del esqueleto es necesario primero generar una esqueletización del usuario. Para realizar una correcta esqueletización de usuario se siguió el ejemplo otorgado por el Developer Toolkit Browser 1.8 del sensor Kinect, el cual nos otorga un ejemplo funcional de una esqueletización simple, y nos da acceso al código fuente de dicha aplicación, nombrada SkeletonBasics-WPF en C#. Partiendo de ahí se explicara el código para realizar una esqueletización.

El primer paso a seguir es que se cuente con las librerías necesarias para esta aplicación, las cuales están escritas al inicio del programa, seguido de contar con ellas, se ingresan las características que se deseen para el dibujo del esqueleto, como el ancho de hueso y color con que se dibujaran, así como el color y radio de las articulaciones etc. A continuación se muestra la manera en la que se crea la brocha para pintar el centro del esqueleto.

```
1 private const double BodyCenterThickness = 10;
```

```
2  private readonly Brush centerPointBrush = Brushes.Blue;
```

De esta manera en la primera línea de código se define el grosor del punto central del esqueleto, y en la segunda se define el color con el que este será dibujado, esto pasos se siguen para las demás partes a dibujar. Después de haber definido todas las características que se seguirán para dibujar el esqueleto, se inicializa la clase, con el código siguiente.

```
1  public MainWindow()
2          {
3              InitializeComponent();
4          }
```

El sensor tiene un rango de operación, por lo que si el usuario se encuentra fuera de ese rango o en una dirección no precisamente frente al sensor, puede ser que el usuario no quede completamente en la imagen, por lo que no es posible realizar una buena captura, para esto, si el cuerpo del usuario no entra en la captura desde algún lado abajo o arriba, se pinta una línea roja en el extremo de la pantalla en el que se esta saliendo la imagen, para de esta manera invitar al usuario a reacomodarse para ser correctamente capturado. A continuación se muestran las líneas de código para lo antes mencionado.

```
1  if (skeleton.ClippedEdges.HasFlag(FrameEdges.Bottom))
2              {
3                  drawingContext.DrawRectangle(
4                      Brushes.Red,
5                      null,
6                      new Rect(0, RenderHeight - ClipBoundsThickness, RenderWidth,
                           ClipBoundsThickness));
7              }
```

En este código se delimita el espacio hacia arriba, y es de la misma manera que se delimita hacia los lados y hacia abajo. Ahora se ejecutan tareas de inicio como buscar si hay algún sensor conectado, en las siguientes líneas se busca un sensor conectado.

```
1  foreach (var potentialSensor in KinectSensor.KinectSensors)
2              {
3                  if (potentialSensor.Status == KinectStatus.Connected)
```

```
                {
                    this.sensor = potentialSensor;
                    break;
                }
            }
```

En caso de haberlo, se enciende el sensor, y en caso de que no se manda un texto informando de que no hay ningún sensor conectado. A continuación se muestra el código requerido para verificar y actuar dependiendo el resultado.

```
if (null != this.sensor)
            {
                try
                {
                    this.sensor.Start();
                }
                catch (IOException)
                {
                    this.sensor = null;
                }
            }

            if (null == this.sensor)
            {
                this.statusBarText.Text = Properties.Resources.NoKinectReady;
            }
```

Después de haber iniciado el sensor, es posible que de alguna manera la aplicación se cierre súbitamente, por lo que se cuenta con un medio de seguridad para detener el sensor en estos casos, que es lo que se hace en el código siguiente.

```
private void WindowClosing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
        }
```

El paso siguiente es crear el arreglo donde se guardaran los datos que se generan en la esqueletización, y en caso de existir alguna se guarda en el arreglo.

```
Skeleton[] skeletons = new Skeleton[0];

            using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
            {
                if (skeletonFrame != null)
                {
                    skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
                    skeletonFrame.CopySkeletonDataTo(skeletons);
                }
            }
```

Para poder utilizar la esqueletización sobre una imagen, es necesario hacer lo que se llamaría una suma de imágenes, simplemente sobreponiendo una en la otra, que en este caso es sobreponer la esqueletización en la imagen color, para esto es necesario que el fondo de nuestra esqueletización sea transparente.por lo cual en las siguientes líneas se dibuja el fondo o “background” transparente para nuestro esqueleto.

```
dc.DrawRectangle(Brushes.Transparent, null, new Rect(0.0, 0.0, RenderWidth, RenderHeight));
```

Después, en caso de que exista alguna esqueletización, se manda la orden de iniciar a dibujar, pero, primero se verifica que el esqueleto sea encontrado correctamente, en caso de haberlo, la orden se envía directamente, en caso de que solo se conozca la posición del sujeto, sin haber encontrado la orientación del esqueleto, se dibuja solo el llamado “center point” o bien punto central, que solo indica el centro de masa de la persona localizada

```
if (skeletons.Length != 0)
                {
                    foreach (Skeleton skel in skeletons)
                    {
                        RenderClippedEdges(skel, dc);

                if (skel.TrackingState == SkeletonTrackingState.Tracked)
                        {
                            this.DrawBonesAndJoints(skel, dc);
                        }
```

```
                else if (skel.TrackingState == SkeletonTrackingState.PositionOnly)
                    {
                        dc.DrawEllipse(
                        this.centerPointBrush,
                        null,
                        this.SkeletonPointToScreen(skel.Position),
                        BodyCenterThickness,
                        BodyCenterThickness);
                    }
                }
            }
```

Las siguientes líneas de código son para tomar precaución en no salirse del área de dibujo.

```
this.drawingGroup.ClipGeometry = new RectangleGeometry(new Rect(0.0, 0.0, RenderWidth,
    RenderHeight));
```

A continuación se empieza a dibujar la esqueletización, en las siguientes líneas de código se dibuja un "hueso" o línea entre dos articulaciones, para hacer esto, se llama la función DrawBone,a la que se le ingresa los datos necesarios para dibujarlo, entre los que cabe resaltar el punto inicial y final del hueso, que son las articulaciones entre las que se encuentra.

```
this.DrawBone(skeleton, drawingContext, JointType.ShoulderLeft, JointType.ElbowLeft);
            this.DrawBone(skeleton, drawingContext, JointType.ElbowLeft, JointType.WristLeft);
            this.DrawBone(skeleton, drawingContext, JointType.WristLeft, JointType.HandLeft);
```

En las líneas de código anteriores se dibujaron los huesos del brazo izquierdo, y de la misma manera en la que se dibujaron estos huesos se dibujan todos los demás de nuestra esqueletización.

Ahora bien, ya se ha mostrado el código para dibujar los huesos, pero ahora falta dibujar las articulaciones, las cuales se crean con un círculo en el punto en el que son detectadas. A continuación se muestra el código para dibujar las articulaciones, verificando si se encuentran correctamente, inferidas o no han sido encontradas, para actuar de acuerdo a ello.

```
foreach(Joint joint in skeleton.Joints)
            {
                Brush drawBrush = null;

```

```
                    if (joint.TrackingState == JointTrackingState.Tracked)
                    {
                        drawBrush = this.trackedJointBrush;
                    }
                    else if (joint.TrackingState == JointTrackingState.Inferred)
                    {
                        drawBrush = this.inferredJointBrush;
                    }

                    if (drawBrush != null)
                    {
                     drawingContext.DrawEllipse(drawBrush, null, this.SkeletonPointToScreen(joint.
                         Position), JointThickness, JointThickness);
                    }
                }
```

Anterior mente uno de los primeros pasos realizados fue declarar el grosor y color de las líneas y círculos con los que se trabajaría, en ese momento se declararon tanto para articulaciones bien localizadas, como para articulaciones inferidas, el código mostrado anteriormente determina el estado de la articulación y decide de que manera dibujarla, o en su caso no hacerlo.

Como se menciono en la parte de dibujar los huesos, se manda llamar una función especializada en dibujarlos, a continuación se muestra dicha función. Primero se obtienen las articulaciones o puntos entre los que se dibujara, para después utilizarlos. Una vez contando con los puntos necesarios se toman las precauciones pertinentes. El primer condicional determina si ninguna de las articulaciones se encuentra, no dibuja nada. El segundo condicional determina si ambas articulaciones se encuentran inferidas, en caso de estarlo tampoco se dibuja nada. Para finalizar, a menos de que ambas articulaciones se encuentren correctamente, se dibuja correctamente, en caso de que una de las dos articulaciones se encuentre inferida, se dibuja al hueso como inferido también.

```
private void DrawBone(Skeleton skeleton, DrawingContext drawingContext, JointType jointType0,
    JointType jointType1)
        {
            Joint joint0 = skeleton.Joints[jointType0];
            Joint joint1 = skeleton.Joints[jointType1];

            if (joint0.TrackingState == JointTrackingState.NotTracked ||
                joint1.TrackingState == JointTrackingState.NotTracked)
```

```
            {
                return;
            }
            if (joint0.TrackingState == JointTrackingState.Inferred &&
                joint1.TrackingState == JointTrackingState.Inferred)
            {
                return;
            }
            Pen drawPen = this.inferredBonePen;
            if (joint0.TrackingState == JointTrackingState.Tracked && joint1.TrackingState ==
                JointTrackingState.Tracked)
            {
                drawPen = this.trackedBonePen;
            }

            drawingContext.DrawLine(drawPen, this.SkeletonPointToScreen(joint0.Position), this.
                SkeletonPointToScreen(joint1.Position));
        }
```

Por ultimo se muestra lo que es conocido como el mapeo de los puntos, lo que hace la siguiente función es dibujar nuestro esqueleto con coordenadas tridimensionales en un plano 2D.

```
private Point SkeletonPointToScreen(SkeletonPoint skelpoint)
        {
            // Convert point to depth space.
            // We are not using depth directly, but we do want the points in our 640x480 output
                resolution.
            DepthImagePoint depthPoint = this.sensor.CoordinateMapper.MapSkeletonPointToDepthPoint
                (skelpoint, DepthImageFormat.Resolution640x480Fps30);
            return new Point(depthPoint.X, depthPoint.Y);
        }
```

De esta manera hemos completado una esqueletización de acuerdo a los pasos obtenidos del Developer Toolkit Browser 1.8 de Kinect, la Fig. 3.8 es la esqueletización creada a partir de este ejemplo.

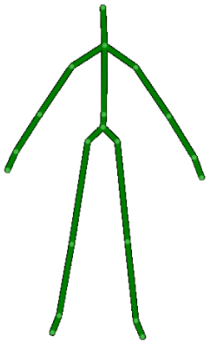

Figura 3.8: Esqueletización con fondo transparente.

Para este proyecto no es suficiente con tener una esqueletización, puesto que se requiere visualizar a un paciente con dicha esqueletización sobrepuesta, para así observar de manera mas ilustrativa y clara las las fallas de postura. Para obtener la imagen color de un paciente con la esqueletización sobrepuesta, es necesario realizar una suma de imágenes, que en este caso es sobre montar el esqueleto con fondo transparente en la imagen color del paciente. En el siguiente diagrama (Fig. 3.9) se muestra la manera en la que se realiza operación.

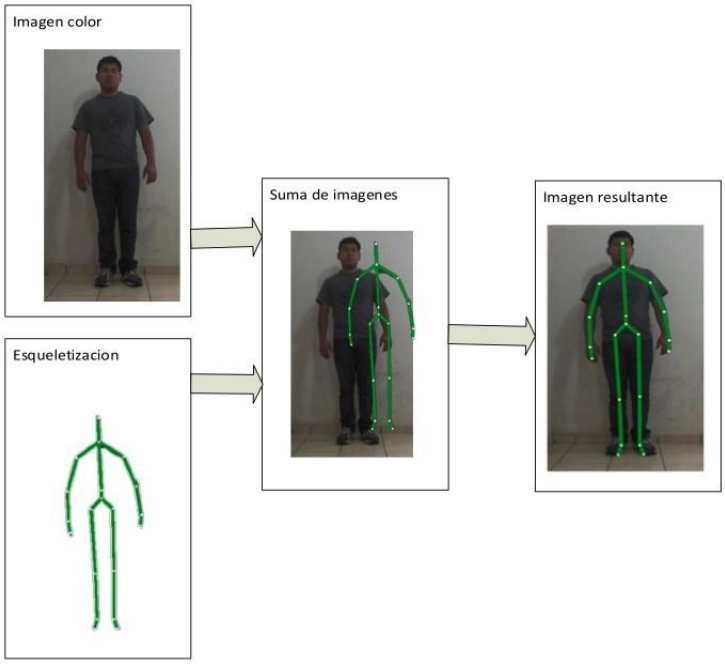

Figura 3.9: Suma de imágenes.

Así se ha obtenido la imagen con esqueletización, pero ahora es necesario obtener las coordenadas de cada articulación de este esqueleto, ya que es el objetivo principal, a continuación se muestran las líneas de código necesarias para extraer las coordenadas en el espacio tridimensional de una articulación.

```
var head = skeleton.Joints[JointType.Head];
            Coordenada de la cabeza en x = head.Position.X;
            Coordenada de la cabeza en y = head.Position.Y;
            Coordenada de la cabeza en z = head.Position.Z;
```

Como se puede observar, es necesario primero extraer el vector complete a una variable, para después de ahí extraer cada coordenada por separado. En este caso se mostró el código para extraer las coordenadas de la cabeza, y estos pasos son los mismos que deben de ser seguidos para obtener las coordenadas de cualquier otra articulación que se desee.

3.5. Obtener distancia entre dos puntos en una esqueletización

Para encontrar la distancia entre articulaciones, o bien, la distancia entre dos puntos en una esqueletización realizada por el sensor Kinect se sigue el método mencionado el capitulo anterior ya que estamos hablando de una imagen, pero al ser una imagen con la suma del mapa de profundidad, esta se convierte en una imagen en 3 dimensiones por así decirlo, por lo que se utiliza la ecuación extendida a 3D (Ec. 3.1).

$$d(P_1, P\,2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{3.1}$$

Entonces al tener ya las coordenadas de cada una de las articulaciones ya es posible simplemente sustituir estas en la Ec. 3.1 para así encontrar la distancia entre dos articulaciones en el espacio mostradas en la esqueletización. En la Fig. 3.10 se muestra una comparación entre la medida obtenida en un brazo utilizando una cinta métrica contra una medición realizada por el sensor.

Figura 3.10: Comparación entre medidas.

3.6. Obtener ángulo entre dos líneas de una esqueletización

Para la obtención del ángulo entre dos vectores dentro de una esqueletización es necesario contar con la información adecuada de los vectores, la cual fue obtenida en los puntos 3.4 y 3.5, que son las coordenadas del inicio y fin del vector, así como la distancia entre estos, o bien la longitud del vector. Después de contar con la información necesaria se prosigue a la obtención del ángulo entre los vectores, para lo cual se utiliza la Ec. 3.2.

$$\alpha = \cos^{-1}\frac{(u_1,u_2,u_3)\cdot(v_1,v_2,v_3)}{\sqrt{(u_1)^2+(u_2)^2+(u_3)^2}*\sqrt{(v_1)^2+(v_2)^2+(v_3)^2}} \tag{3.2}$$

Al analizar la ecuación mostrada anteriormente se deduce que al sustituir los datos con los que ya se cuenta se obtiene una ecuación de manera más adecuada a nuestras necesidades (Ec. 3.3).

$$\alpha = \cos^{-1}\frac{(u_1,u_2,u_3)\cdot(v_1,v_2,v_3)}{Lv_1 * Lv_2} \tag{3.3}$$

De donde u_1, u_2 y u_3 son las coordenadas del punto final del vector 1, v_1, v_2 y v_3 son las coordenadas del punto final del vector dos, y para finalizar Lv_1 y Lv_2 son las longitudes del vector 1 y 2 respectivamente. En las Figs. 3.11 y 3.12 se muestran un brazo y una pierna formando un ángulo siendo analizados por el sensor.

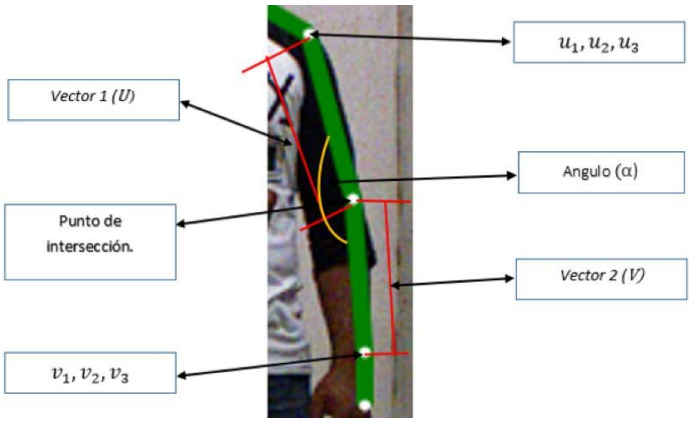

Figura 3.11: Vectores en brazo con esqueletización.

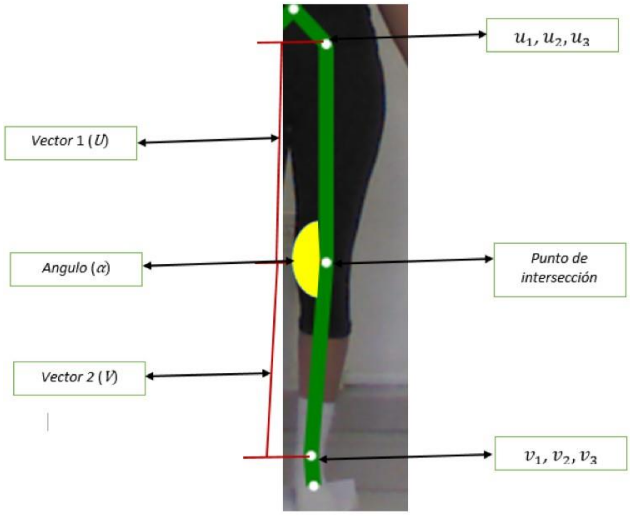

Figura 3.12: Vectores en pierna con esqueletización.

Al haber identificado a ambos vectores necesarios para la ecuación, así como sus puntos finales e intersección es posible obtener el ángulo, ya que se conocen los valores de las coordenadas gracias a lo visto en el punto 3.4, y las distancias gracias a lo mostrado en el punto 3.5. Obteniendo de esta manera el ángulo en grados que se encuentra entre los vectores en cuestión.

Capítulo 4

Resultados

En esta sección se muestran los resultados obtenidos a partir de seguir la metodología descrita en el capítulo 3, en este capítulo se muestran las medidas realizadas, así como los reportes elaborados a partir del análisis de distancias y ángulos entre articulaciones de una esqueletización. Se muestra también los dos tipos de reportes guardados por la interfaz, para sus diferentes usos, además de las imágenes obtenidas.

4.1. Creación de imagen con esqueletizacion

Se creó una imagen a partir de la suma de la imagen color que entrega el sensor Kinect y una esqueletizacion, la cual es de igual manera proporcionada por este sensor. La imagen color es obtenida y guardada en formato .png (Portable Network Graphics, por sus siglas en inglés) con una resolución de 640x480, para después ser utilizada. La esqueletizacion se genera a partir de las coordenadas brindadas por el sensor, dibujando círculos en cada una de las articulaciones referentes a las coordenadas obtenidas, y uniendo estos círculos con líneas gruesas para así simular la esqueletizacion. Por ultimo estas dos imágenes se sobreponen para lograr así la imagen con esqueletizacion (Fig. 4.1)

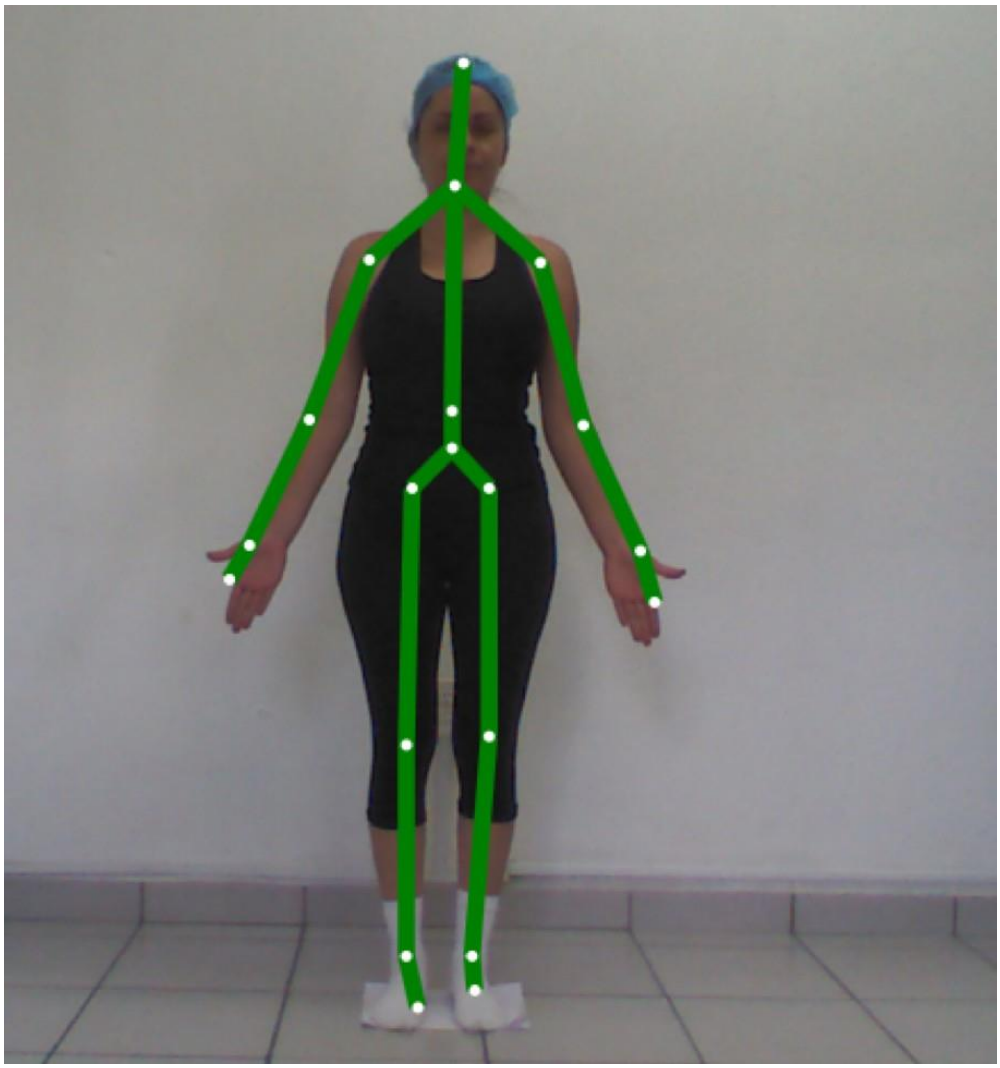

Figura 4.1: Imagen con esqueletización.

4.2. Interfaz de usuario

Se diseñó una interfaz de usuario de fácil comprensión para facilitar su uso (Fig. 4.2), esta interfaz fue desarrollada en Visual C#, la cual permite conectarse con el sensor Kinect, extraer la imagen color, así como las coordenadas de la esqueletizacion, permitiéndonos así mostrar a la salida una imagen con la suma de las imagen color y la esqueletización. Esta interfaz también pide datos importantes para la realización de un reporte de evaluación, se de alguna manera no se ingresan todos los datos requerido la misma interfaz hace aparecer un mensaje de texto donde te

menciona ingresar todos los datos. La interfaz cuenta con algunas otras funciones, como lo son el poder de inicializar y detener el sensor en el momento que se desee. A continuación se muestra la interfaz antes mencionada, así como se identifica y explica cada parte de esta.

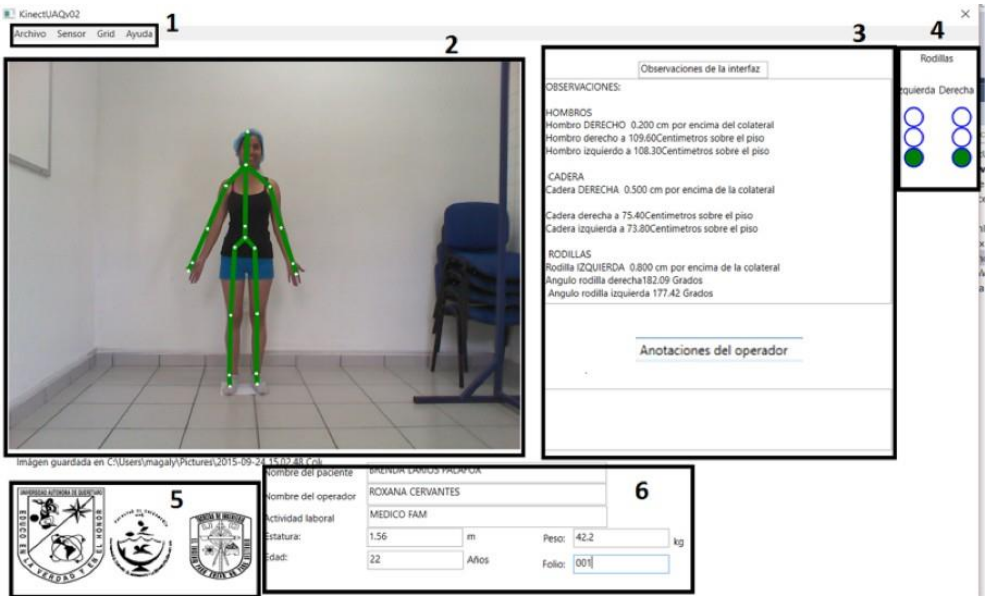

Figura 4.2: Interfaz de usuario.

1. Menú desplegable. Mediante este menú se puede acceder a una gran cantidad de opciones que nos ofrece la interfaz, como por ejemplo al dar clic en ”Archivo” nos aparecen las opciones: “Nuevo”, que inicializa la interfaz, borrando los datos guardados anteriormente, “Guardar”, que guarda una imagen de la interfaz, la imagen del paciente con esqueletizacion, un archivo .txt con todos los datos obtenidos y genera un reporte en Excel. Y por ultimo “Salir”, que cierra la interfaz de manera segura.

   Mediante el menú “Sensor”, se puede inicializar y detener el sensor Kinect para realizar capturas, al dar click en “Iniciar sensor” y “Detener sensor” respectivamente. Por medio del menú “Grid” se puede aplicar un grid que funciona a modo de malla de referencia para apreciar longitudes. “Grid on” y “Grid off” Aparecen y desaparecen la malla. Por ultimo al dar click en el menú “Ayuda” accedemos a la opción “Manual de usuario” la cual nos redirige a un manual de uso para la interfaz en un archivo tipo PDF y a la opción “Contáctanos” En la cual aparece un cuadro de dialogo en el que se muestran el correo y teléfono del creador

de la interfaz para mayor información.

2. Imagen con esqueletización. Se muestra una imagen con su respectiva esqueletización en tiempo prácticamente real ya que se actualiza la captura unas 30 veces por segundo. Esta imagen se congela al dar click en detener sensor y al dar guardar esta será la que se muestre en el reporte de Excel. Esta imagen cuenta con apoyo para visualizar fácilmente las fallas en las rodillas como valgo y varo, que es en lo que se especializa esta interfaz. A continuación se muestra una imagen con ambas rodillas bien en comparación de una en la cual se inducen fallas distintas a ambas rodillas.

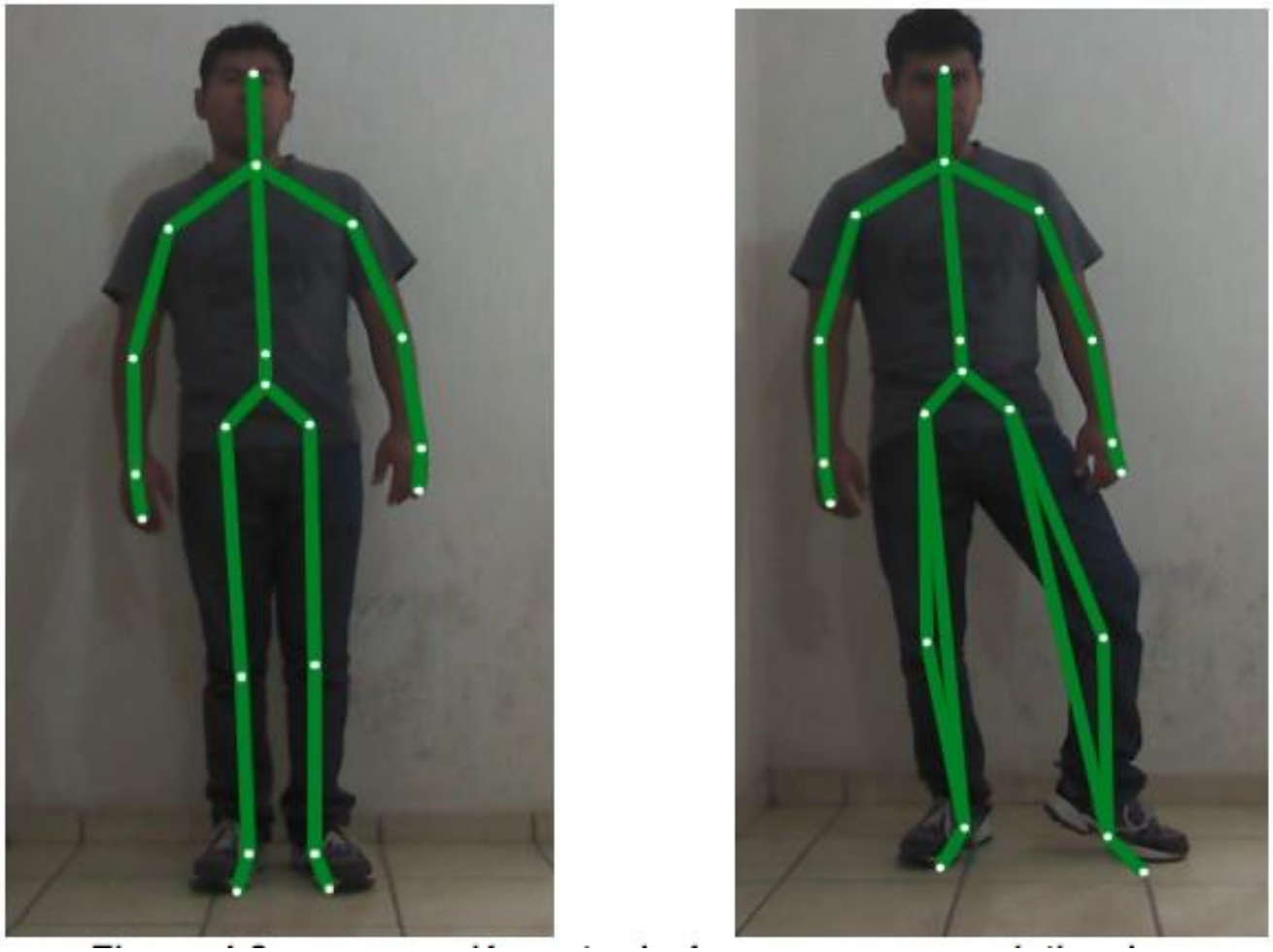

Figura 4.3: Comparación entre imágenes con esqueletización.

Como se puede observar en la Fig. 4.3 cuando las rodillas se encuentran en buen estado, las esqueletización es normal (imagen de la izquierda) mientras que al tener algún fallo (imagen

de la derecha) se genera una línea que muestra el camino que debería de seguir la esqueletización en comparación con el que sigue.

3. Cuadros de anotaciones. En estos recuadros se muestran las observaciones realizadas por la misma interfaz, las que al igual que la imagen se actualizan constantemente dependiendo de la lectura realizada por el sensor en ese preciso momento, y las anotaciones hechas por el operador. Las anotaciones antes mencionadas se incluyen en el reporte, además de que el operador solo puede hacer anotaciones y observaciones pero no puede cambiar las observaciones hechas por la interfaz. Dentro de las anotaciones de la interfaz se encuentran el desequilibro en hombros, caderas y rodillas, así como el angulo que forman ambas rodillas, determinando de acuerdo a este si tiene algún problema de valgo o varo, y si este problema es critico o ligero.

4. Semáforos indicadores. Estos semáforos son una herramienta muy sencilla pero de mucha ayuda para alertar en caso de algún fallo en rodillas, ya que si la rodilla respectiva al semáforo se encuentra bien el semáforo se ilumina de verde, mostrando un nivel de tranquilidad, al presentar dicha rodilla un problema determinado por la interfaz como ligero, el semáforo se iluminará de naranja, mostrando un bajo nivel de preocupación, pero recomendando algún tipo de rehabilitación, mientras que en el caso de que el semáforo se iluminé de rojo, significa que esta rodilla tiene un problema critico que debe ser atendido a la brevedad. En la Fig. 4.4 se muestra la forma en la que actúan los semáforos.

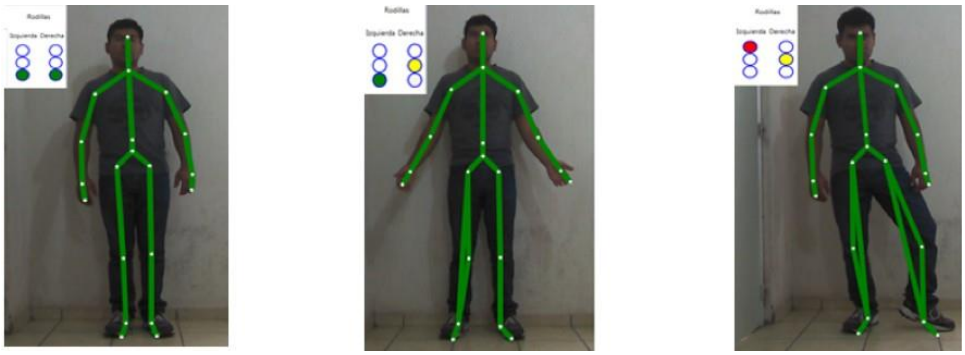

Figura 4.4: Funcionamiento de los semáforos.

5. Escudos. Aquí solo se muestran los escudos de las facultades que trabajaron en vinculación

para la realización de este proyecto, así como el de la universidad.

6. Datos requeridos. En esta sección aparecen cuadros de texto donde se deben ingresar los datos necesarios del paciente y operador para realizar el reporte. Se piden datos del paciente para llenar una ficha común, como lo son el nombre, la edad, peso, altura y actividad laboral, así como también el nombre del operador y el numero de expediente del paciente para llevar un fácil manejo de los archivos.

4.3. Extracción de información completa de coordenadas y observaciones en archivo.txt.

Para posibles próximas actualizaciones de la interfaz se extraen todos los datos de coordenadas y observaciones, al igual que los datos del paciente, usuario, fecha y hora de cada operación, estos datos son guardados en un documento tipo texto (Fig. 4.5).

Figura 4.5: Documento de texto.

4.4. Reporte en excel

Para finalizar la operación, el reporte se guarda en una hoja de Excel, ya que de esta manera fue solicitado por los especialistas del área de fisioterapia. En este reporte se ordenan los datos de fecha y hora de la prueba, así como también los datos del usuario y paciente, seguidos de la imagen color

más la esqueletización sobrepuesta. Para poder ingresar la imagen color fue necesario utilizar la clase ScreenCapturer, obtenida del sitio web http://stackoverflow.com/a/11957567 ya que se utilizó la misma imagen que se tiene en la interfaz de usuario. También se imprimen las observaciones realizadas mediante la interfaz y se deja abierto a modificaciones que se desee realizar por el operador, además de que es posible utilizar formulas sencillas del mismo Excel. El reporte de Excel (Fig. 4.6) se guarda en el disco local C: de la computadora en la que se utiliza la interfaz, teniendo como nombre el numero de folio del paciente, así como la fecha en la que se realizo el análisis para facilitar su búsqueda y llevar un buen archivo de análisis realizados.

Operador	ROXANA
Pacinte	DANIEL
Folio	0
Actividad	INGENIERO
Edad	22 años
Peso	72kg
Estatura	1.78m

OBSERVACIONES:

HOMBROS
Hombro IZQUIERDO 0.200 cm por encima del colateral
Hombro derecho a 134.50Centimetros sobre el piso
Hombro izquierdo a 135.00Centimetros sobre el piso

CADERA
Cadera DERECHA 0.200 cm por encima de la colateral

Cadera derecha a 94.60Centimetros sobre el piso
Cadera izquierda a 94.70Centimetros sobre el piso

RODILLAS
Rodilla IZQUIERDA 0.600 cm por encima de la colateral
Angulo rodilla derecha182.36 Grados
Angulo rodilla izquierda 179.75 Grados

Figura 4.6: Reporte de Excel.

Al haber guardado el reporte de Excel exitósamente, puedes regresar a la interfaz, dar archivo nuevo, y comenzar con otro análisis.

4.5. Manual de usuario

Para facilitar el uso de la interfaz, y el sistema en general, se elaboró un manual de usuario, con el propósito de facilitar la instalación y uso del sistema, así como también para apoyo en caso de algún problema. En este se dan una serie de pasos específicos, los cuales se deben de seguir para realizar la instalación, los cuales son apoyados por medio de diagramas e imágenes. En el caso de alguna falla, se muestran las formas de solucionar las fallas más comunes, las cuales a menudo se solucionan de maneras muy sencillas, en dado caso de que la falla se suficiente mente complicada como para no solucionarla por medio de las instrucciones mostradas en el manual, se deja un correo para comunicarse con los creadores de la interfaz. El manual de usuario se adjunta completo en la parte de los anexos de este trabajo de tesis.

Capítulo 5

Conclusiones

Utilizar el SDK fue relativamente fácil, ya que aunque no es un medio de programación con la que muchos estemos familiarizados, cuenta con mucha ayuda otorgada por la propia corporación de Microsoft en línea, así como con muchos ejemplos simples otorgados en el mismo toolkit de Kinect, de donde puedes obtener el código fuente.

Trabajar con el Kinect es muy práctico, ya que es un sensor que puedes obtener fácilmente, debido a que esta en venta al público en general, a un precio muy accesible, además de que es ligero y de un tamaño considerablemente pequeño tomando en cuenta lo que puede hacer.

Al realizar pruebas se encontró que el Kinect, como todo, no es infalible, ya que presento algunas fallas, por ejemplo:

- Tiene un rango determinado, por lo que se podría decir que se encuentra limitado en cuanto a espacio de operación
- Es frágil, por lo que debe ser tratado con cuidado.
- Es necesario contar con una PC que cuente con un sistema operativo compatible con el sensor para poder utilizarlo, lo que hace crecer un poco el costo para trabajar con el.

- Es necesario trabajar dentro de un espacio con una cantidad de luz apropiada, ya que la mala iluminación afecta a las capturas
- En algunas ocasiones las esqueletizaciones no se sobreponen a las imágenes de la manera mas exacta estrictamente hablando, por lo que se ve una esqueletizacion un poco desfasada.
- Los valores de las coordenadas del esqueleto obtenidas no son exactas, mas sin embargo están dentro de un rango considerablemente bueno.

En general el trabajar con Kinect es muy llamativo, ya que es posible realizar un sinfín de aplicaciones basándonos en sus herramientas, y es de alguna manera un sensor fácil de trabajar, tomando en cuenta el tipo de programación, tamaño y precio.

Utilizar los algoritmos de análisis postural, resulto ser meramente matemático, ya que después de obtener las coordenadas de las articulaciones, el proceso se limita a geometría y aritmética.

Obtener las coordenadas de las articulaciones es de gran ayuda, ya que abre muchas puertas, no solo para lo que se realizó en este proyecto, sino también para muchas otras cosas, ya que al saber en dónde se encuentra cada articulación se pueden desarrollar un gran número de aplicaciones.

Capítulo 6

Prospectivas

Existen una gran variedad de cosas que se pueden hacer a futuro dentro de este proyecto, a continuación se muestran algunas ideas a futuro.

- Integrar más análisis dentro de la postura frontal, como pueden ser el arqueo en brazos, alineación del cuello y problemas de cadera.
- Implementar también el análisis de las diferentes vistas de la postura, como lo son la vista posterior, lateral derecha y lateral izquierda.
- Realizar análisis dinámicos, donde se le pida al paciente realizar un movimiento mientras es analizado por el Kinect, determinando de esta manera si el movimiento se realiza correctamente, y a partir de estos análisis contar con un reporte mas completo.
- Programar rutinas de rehabilitación, dependiendo el análisis que entregue el sistema, en las cuales también puede participar el Kinect pidiendo realizar algunos movimientos al paciente y calificando si este se realiza bien.

Capítulo 7

Referencias

Alfonso-Peñaloza, Y., Cepeda-López, J., Navarro-Valencia, M., Tirado-Todaro, a., Quintero-Moya, S., Ramírez, P., & Angarita, a. (2013). Reproducibilidad interevaluador del Formato de Observación Sistemática de la Alineación Corporal en estudiantes universitarios. Fisioterapia, 35(4), 154–166. doi:10.1016/j.ft.2012.09.006.

Anton, H. (1994). Introducción al álgebra lineal. LIMUSA SA. de CV.

Borghese, N. A., Pirovano, M., Mainetti, R., & Lanzi, P. L. (2012). An integrated low-cost system for at-home rehabilitation. 2012 18th International Conference on Virtual Systems and Multimedia, 553–556. doi:10.1109/VSMM.2012.6365975.

Burger, W., & Burge, M. J. (2009). Principles of Digital Image Processing (Fundamental Techniques).

Chien-Yen Chang , Belinda Lange, Mi Zhang, Sebastian Koenig , Phil Requejo, Noom Somboon, Alexander A. Sawchuk, and A. A. R. (2012). Towards Pervasive Physical Rehabilitation Using Microsoft Kinect, 159–162.

Diego, S., Kurillo, G., Ofli, F., Bajcsy, R., Seto, E., Jimison, H., & Pavel, M. (2012). Accuracy and Robustness of Kinect Pose Estimation in the Context of Coaching of Elderly Population. 34th Annual International Conference of the IEEE EMBS. San Diego California USA, 28 Agust- 1

September, 2012, 1188–1193.

Fernández-Baena, A., Susin, A., & Lligadas, X. (2012). Biomechanical Validation of Upper-Body and Lower-Body Joint Movements of Kinect Motion Capture Data for Rehabilitation Treatments. 2012 Fourth International Conference on Intelligent Networking and Collaborative Systems, 656–661. doi:10.1109/iNCoS.2012.66.

Gama, A. Da, Chaves, T., Figueiredo, L., & Teichrieb, V. (2012). Guidance and Movement Correction Based on Therapeutics Movements for Motor Rehabilitation Support Systems. 2012 14th Symposium on Virtual and Augmented Reality, 191–200. doi:10.1109/SVR.2012.15.

Garduño, M. A., Morales, L. A., & Osornio, R. A. (2014). Morphological filters applied to Kinect depth images for noise removal as pre-processing stage, 2–6.

Garrido Navarro, J. E., R. Penichet, V., Lozano, Ma., & Marset, I. (2013). Balance Disorder Rehabilitation through Movement Interaction. Proceedings of the ICTs for Improving Patients Rehabilitation Research Techniques, 319–322. doi:10.4108/pervasivehealth.2013.252368.

Hansard, M., Lee, S., Choi, O., & Horaud, R. (2013). Time-of-Flight Cameras Principles, Methods and Applications. Springer.

Lange, B., Chang, C.-Y., Suma, E., Newman, B., Rizzo, A. S., & Bolas, M. (2011). Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor. Conference Proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference, 2011, 1831–4. doi:10.1109/IEMBS.2011.6090521.

Lange, B., Koenig, S., McConnell, E., Chang, C.-Y., Juang, R., Suma, E., Rizzo, A. (2012). Interactive game-based rehabilitation using the Microsoft Kinect. 2012 IEEE Virtual Reality (VR), 171–172. doi:10.1109/VR.2012.6180935.

Yeh, S. C., Hwang, W. Y., Huang, T. C., Liu, W. K., Chen, Y. T., & Hung, Y. P. (2012). A study for the application of body sensing in assisted rehabilitation training. Proceedings - 2012 International Symposium on Computer, Consumer and Control, IS3C 2012, 922–925. doi:10.1109/IS3C.2012.240.

Páginas web consultadas:

Kinect for Windows: http://www.microsoft.com/en-us/kinectforwindows/ Septiembre 2014.

http://stackoverflow.com/a/11957567/ Abril 2016

Apéndice A

Artículo 3er Encuentro de Jovénes Investigadores del Estado de Querétaro

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

CONACYT
Consejo Nacional de Ciencia y Tecnología

CONCYTEQ

3ER. ENCUENTRO DE JÓVENES INVESTIGADORES DEL ESTADO DE QUERÉTARO

"IDEAS JÓVENES PARA IMPULSAR EL CRECIMIENTO DEL PAÍS"

A través de la Dirección de Investigación y Posgrado

Otorgan la presente

Constancia

A

Daniel Jaramillo Quintanar

POR SU PARTICIPACIÓN COMO PONENTE DEL TRABAJO TITULADO

Desarrollo de interfaz para análisis postural mediante sensor Tof Kinect

Dr. Irineo Torres Pacheco
Secretario Académico de la
Universidad Autónoma de Querétaro

Dra. Ma. Guadalupe Flavia Loarca Piña
Directora de Investigación y Posgrado de la
Universidad Autónoma de Querétaro

Santiago de Querétaro, Qro., 7 y 8 de Octubre de 2015.

DESARROLLO DE INTERFAZ PARA ANÁLISIS POSTURAL MEDIANTE SENSOR TOF KINECT

Daniel Jaramillo Quintanar*, Marco Antonio Garduño Ramón, Arely Guadalupe Morales Hernández, Luis Alberto Morales Hernández.

Universidad Autónoma de Querétaro, Facultad de Ingeniería, Campus San Juan del Río

djaramilllo5@gmail.com

Antecedentes y/o fundamentación teórica

La postura de una persona es analizada mediante observación y ciertas herramientas [1]. Sin embargo, al ser algo meramente cualitativo se generan diferencias en el diagnóstico, se han usado herramientas auxiliares como láser [5], o incluso múltiples cámaras [6] para ayudar en esta tarea. Recientemente el sensor Kinect ha sido usado para rehabilitación y análisis de postura [3, 4] con buenos resultados.

Sensor Kinect

El sensor Kinect apareció en Noviembre de 2011. Fue desarrollado por la compañía Prime Sense en colaboración con Microsoft. Tiene una cámara RGB así como un emisor y sensor de luz infrarroja, en conjunto permiten capturar imágenes a color y entregar la información de profundidad de cada píxel en la escena. La medición de profundidad se basa en un método de luz estructurada, el cual consiste en proyectar una serie de puntos conocidos con el emisor infrarrojo y usar el sensor infrarrojo para medir su deformación al incidir sobre los objetos. La función más popular de este dispositivo la de detección automática de hasta 20 articulaciones del cuerpo humano para dos usuarios.

Análisis postural

Para realizar el análisis postural se pide al paciente adopte ciertas posiciones a partir de las cuales se realizaran las observaciones. El especialista debe de tomar nota de algunos puntos de interés en el paciente y en algunos casos es necesario palpar el cuerpo de este. Ocasionalmente, se le pide al paciente quedarse solo con su ropa interior para observar

directamente la musculatura así como la orientación ósea. El especialista se apoya del uso de instrumentos como la plomada o del posturómetro para verificar la verticalidad de la postura del paciente.

Distancia Euclidiana

La distancia euclidiana se deduce a partir del teorema de Pitágoras al realizarse el análisis de las distancias entre dos puntos en el plano a partir de los vectores que los unen. A partir de este principio se obtiene la expresión con la cual se puede conocer la magnitud de la distancia entre dos puntos $P_1 = (x_1, y_1, z_1)$ $P_2 = (x_2, y_2, z_2)$ en el espacio tridimensional:

$$d(P_1, P_2) \Rightarrow \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad \text{.....(1)}$$

Ángulos entre vectores

El ángulo (α) que forman dos vectores u, v ∈ Rn, no nulos, se obtiene de la igualdad: u·v = (‖u‖·‖v‖)*cos(α), en la que se despeja el ángulo obteniendo la ecuación 2:

$$\alpha = \cos^{-1}\frac{4 \cdot 5}{(\|4\| \cdot \|5\|)} \quad \text{..... (2)}$$

Donde ‖u‖ es la norma del vector u, o bien la magnitud de la distancia entre los extremos de este vector. Lo mismo sucede con ‖v‖. Esto permite obtener una expresión general como se muestra en la ecuación 3.

$$\alpha = \cos^{-1}\frac{(4_1, 4_2, 4_3) \cdot (5_1, 5_2, 5_3)}{\left(\sqrt{(4_1)^2 + (4_2)^2 + (4_3)^2}\right) * \sqrt{(5_1)^2 + (5_2)^2 + (5_3)^2}} \quad \text{..... (3)}$$

Descripción del problema

De acuerdo a la Secretaria de Salud, el 70% de la población estudiantil presenta problemas de postura. La confiabilidad de los análisis de postura generalmente recae en la experiencia de los especialistas del área de fisioterapia quienes emplean un método tradicional basado en observaciones, así como mediante el uso de ciertos instrumentos como la plomada o el posturómetro. Se han desarrollado herramientas que emplean diferentes sensores para complementar dicho diagnósticos y convertirlos algo más cuantitativos, sin embargo sus

costos la mayoría de las veces son elevados. Se hace evidente la necesidad de desarrollar una herramienta de bajo costo que permita realizar análisis de postura y que además facilite su implementación en zonas marginadas.

Justificación del proyecto

El procesamiento de imágenes ofrece una buena opción para abordar la problemática planteada. Existen algoritmos los cuales tienen como finalidad extraer ciertas características de una escena u objeto, con lo cual no solo se busca tener como salida una imagen procesada, sino también obtener información cuantitativa la cual sea de utilidad. Los sistemas de visión de tiempo de vuelo (TOF), por ejemplo, permiten obtener información tridimensional de la escena observada haciendo uso de emisores y sensores de luz infrarroja. Sensores TOF de bajo costo como el sensor Kinect de Microsoft incorporan algoritmos de detección para hasta 20 articulaciones del cuerpo humano. Combinando ambas herramientas se propone desarrollar un sistema el cual sirva como sistema auxiliar para los expertos del área de fisioterapia en la realización de análisis de postura.

Hipótesis

Es posible desarrollar un sistema para análisis de postura mediante el uso de procesamiento de imágenes y sistemas de visión TOF de bajo costo.

Objetivo

Desarrollar una interfaz gráfica de usuario utilizando la información del sensor Kinect y algoritmos de análisis postural para realizar la detección automática de problemas posturales.

Metodología

El diagrama a bloques (figura 1) muestra la metodología propuesta. El sensor Kinect entrega un mapa de profundidad y una imagen color. A partir del mapa de profundidad el sistema obtiene una esqueletización del sujeto con las coordenadas x, y, z en milímetros medidas a partir del sensor, las cuales son ingresadas a un algoritmo de análisis postural (medición de distancias y ángulos entre miembros articulares). En un reporte de evaluación se concentra

el resultado obtenido por los algoritmos de análisis postural, así como una imagen color donde igual se muestra la esqueletización para evidenciar los problemas detectados, observaciones realizadas por el especialista y una serie de datos personales del usuario.

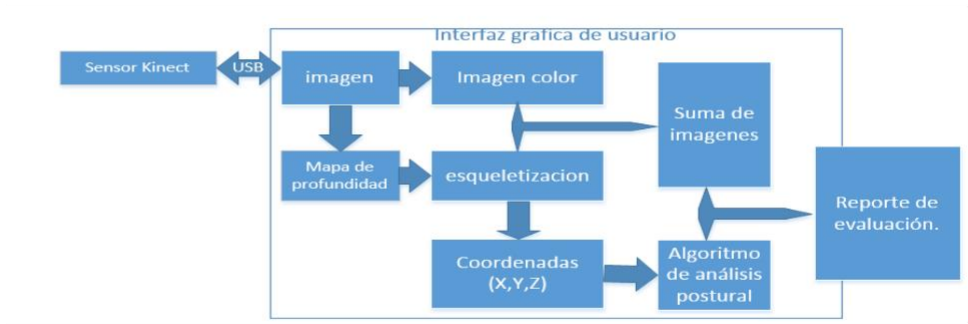

Figura 1: Diagrama a bloques de la metodología

Resultados y discusión

Se probó el sistema con la ayuda de una persona que simulo una serie de fallas posturales como se ve en las figuras 2, 3 y 4.

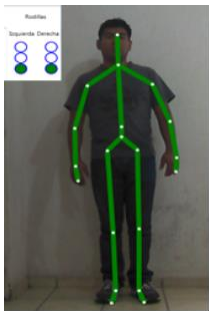

Figura 2
Ambas rodillas bien

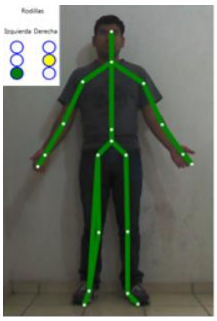

Figura 3
Rodilla derecha en valgo

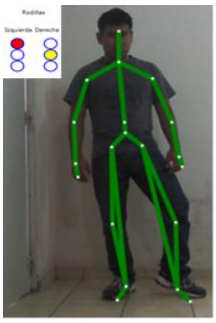

Figura 4
Ambas rodillas en varo

La figura 2 muestra al sujeto sin problemas de rodillas con ambos semáforos en verde, la figura 3 muestra al sujeto con un valgo ligero en la rodilla derecha con un semáforo en verde y otro en amarillo, y la figura 4 muestra a el sujeto con un varo ligero en rodilla derecha y al mismo tiempo varo critico en izquierda. Además del análisis de rodillas el sistema mide el desnivel en hombros, caderas y rodillas, para conjuntar todo a un reporte de observación.

Conclusiones

Se desarrolló un sistema para análisis de postura frontal usando un sensor Kinect que se especializa en la detección de problemas en rodillas y desbalances articulares clasificándolos de manera automática. El sistema se probó induciendo errores de postura donde se observaron buenos resultados, sin embargo como trabajo futuro se tiene previsto validarlo con ayuda de personal de la Facultad de Enfermería de la Universidad Autónoma de Querétaro.

Agradecimientos

Se agradece a la Facultad de Enfermería y al Proyecto Nuevos Talentos Científicos y Tecnológicos del CONCYTEQ por el apoyo brindado.

Referencias

[1] Alfonso-Peñaloza, Y., Cepeda-López, J., Navarro-Valencia, M., Tirado-Todaro, a., Quintero-Moya, S., Ramírez, P., & Angarita, a. (2013). Reproducibilidad interevaluador del Formato de Observación Sistemática de la Alineación Corporal en estudiantes universitarios. *Fisioterapia*, *35*(4), 154–166. doi:10.1016/j.ft.2012.09.006

[2] Anton, H. (1994). *Introducción al álgebra lineal*. LIMUSA SA. de CV.

[3] Borghese, N. A., Pirovano, M., Mainetti, R., & Lanzi, P. L. (2012). An integrated low-cost system for at-home rehabilitation. *2012 18th International Conference on Virtual Systems and Multimedia*, 553–556. doi:10.1109/VSMM.2012.6365975

[4] Lange, B., Koenig, S., McConnell, E., Chang, C.-Y., Juang, R., Suma, E., … Rizzo, A. (2012). Interactive game-based rehabilitation using the Microsoft Kinect. *2012 IEEE Virtual Reality (VR)*, 171–172. doi:10.1109/VR.2012.6180935

[5] Miguel Reyes, José Ramírez-Moreno, Juan R. Revilla, Petia Radeva1, y Sergio Escalera1. ADiBAS: Sistema Multisensor de Adquisición Automática de Datos Corporales Objetivos, Robustos y Fiables para el Análisis de la Postura y el Movimiento

Páginas web consultadas

[6]http://www.biomec.com.co/archivos/CATALOGO-LABORATORIO-DE-BIOMECANICA36.pdf. (Septiembre 2015).

Apéndice B

Manual de Usuario

PosturalKinectUAQ

Universidad Autonoma de Querétaro

Daniel Jaramillo Quintanar
M. en C. Marco Antonio Garduño Ramon
Dr. Luis Alberto Morales Hernandez
Version 1

Resumen

El PosturalKinectUAQ es un sistema que se encarga de hacer analisis de postura de pacientes, mediante la captura de una imagen color y de profundidad de cada uno de los puntos en la escena de un usuario colocado de pie frente al sistema. El software detecta 20 articulaciones del cuerpo humano de manera automatica lo cual permite aplicar algoritmos de análisis de postura y generar un reporte con las observaciones detectadas. La aplicacion se especializa en la deteccion de problemas en rodillas, como lo son valgo y varo, así como la deteccion de desequilibrios corporales. El sistema no requiere el uso de un posturometro, ni de poner a plomo al paciente; se busca que la herramienta sirva como auxiliar de los expertos del área de fisioterapia.

Índice

Para la correcta instalacion y uso adecuado del PosturalKinectUAQ se recomienda seguir la sencilla serie de pasos mostrados a continuacion.

1. Instalacion de programas en la PC

Para poder utilizar el PosturalKinectUAQ en tu PC es necesario instalar algunos programas previamente, los cuales se daran a conocer a continuacion, así como la forma de instalarlos.

1. Kinect SDK
   Para realizar la instalacion del Kinect SDK se recomienda seguir la sencilla serie de pasos dados a continuacion.

   - Dirigirse a la carpeta proporcionada ”PosturalKinectUAQ”, dentro de esta carpeta se encuentra otra carpeta llamada programas, en la cual se encuentran los tres programas que se debe de instalar.

   - Al encontrarse ya en esta carpeta localizar el KinectSDK-v1.8-setup y dar clic derecho sobre él, ejecutar como administrador.

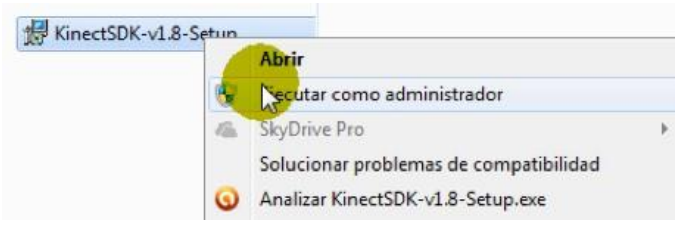

Figura 1:

   - Después de haberlo ejecutado aparece automaticamente una ventana que te pide permiso para realizar cambios en tu PC, a la que solo debes de proporcionar el permiso presionando OK (Si ya estas como administrador es probable que este mensaje no aparezca).
   - Aparecer un cuadro de dialogo que le pedir permiso para ejecutar el archivo en el cual simplemente debe de presionar ejecutar.

Figura 2:

- Después de haber ejecutado el archivo aparecer otro mensaje

  en el que nos da a conocer un contrato de licencia para usuario final, en el cual después de haber leído los requisitos debe de checar que acepta los términos y condiciones de licencia, para a continuacion presionar instalar y comenzar la instalacion.

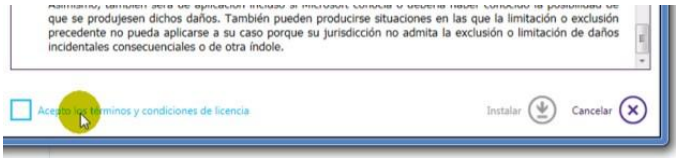

Figura 3:

- Después de esperar un momento mientras se realiza la instalacion la misma ventana nos avisara con un mensaje de instalacion finalizada en la cual solo debemos de presionar cerrar para concluir.

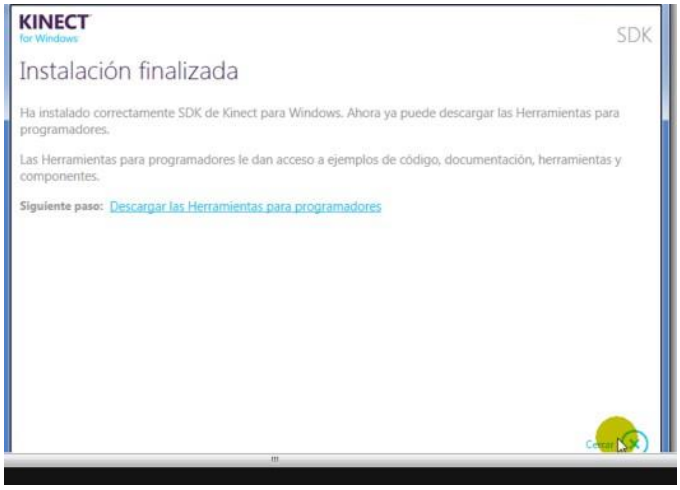

Figura 4:

2. Kinect Developer Toolkit
   Para la instalacion del Kinect Developer Toolkit se siguen los mismos pasos que para el SDK con la única diferencia que en este caso se da clic derecho sobre el KinectDeveloperToolkit-v1.8-setup.

   Nota: Se recomienda instalar los accesorios del Kinect en la forma mostrada, primero el SDK seguido del Developer Toolkit.

2. Instalacion física del sensor Kinect

Después de haber obtenido el sensor Kinect en conjunto con el cable tipo "Y" (Fig. 5) y una PC con un Windows 7 u 8, para realizar su correcta conexion prosiga a realizar la siguiente lista de pasos.

Figura 5:

1. Verificar que se cuenta con todos los elementos mostrados en la Fig. 5 que son:

- Sensor Kinect
- Cable tipo “Y”

2. Identificar los diferentes tipos de conexion con los que se cuenta, ya que los cables contienen dos muy parecidas, la conexion USB normal que es de color negro y la conexion al Kinect que es de color naranja. Fig. 6.

Figura 6:

3. Al haber identificado los cables se puede proseguir con la conexion, para realizar esta se recomienda seguir los siguientes pasos.
   - Seleccionar los dos extremos de los cables con color naranja y conectarlos entre sí.

Figura 7:

   - Conectar el extremo en forma de cargador a una fuente de alimentacion de 127v en corriente alterna, que es la que se encuentra en los enchufes comunes de un hogar.
     - Al haber realizado este paso se debe de encender una pequeña luz de color verde en el cable en la parte de la separacion del tipo “Y”.
     - En caso de que no se encienda verifique que su fuente de alimentacion sea correcta y vuelva a intentarlo.
   - Conectar el extremo tipo USB (color negro) a un puerto de la PC (2.0 o 3.0 de preferencia).
     - Al haber terminado las conexiones exitosamente se debe de encender también una luz color verde en el frente del Kinect.

Figura 8:

3. Instalacion del sensor Kinect en la PC

Una vez realizados los pasos anteriores y haber conectado el sensor Kinect a la PC, si es la primera vez que se conecta el dispositivo, la PC lo detectara e iniciara la configuracion de drivers, cuando todo el proceso termine se ver una ventana como la siguiente.

Figura 9:

Ademas, el sensor Kinect deber mostrar el led que tiene en la parte frontal iluminado de color verde sin parpadear.

- En caso de que dicha ventana no aparezca o el led frontal del sensor se ilumine de color rojo se debe verificar en el administrador de dispositivos de Windows que los diferentes componentes del Kinect hayan sido reconocidos e instalados correctamente, ver Fig. 10.
- En caso afirmativo se recomienda desconectar el sensor, reiniciar la computadora y repetir los pasos anteriores.
- En el caso contrario y en el de no lograr un funcionamiento correcto del sensor se debe desconectar el sensor y desinstalar todo software relacionado con el sensor Kinect, reiniciar la computadora y volver a iniciar con la instalacion del software.
- En un caso último en que bajo ninguna circunstancia se logre echar a andar el dispositivo, se debe probar en otra PC y descartar un mal funcionamiento del sensor o de la misma computadora .
- Por ultimo en un caso extremo, solicitar soporte al distribuidor del sensor Kinect y al fabricante.

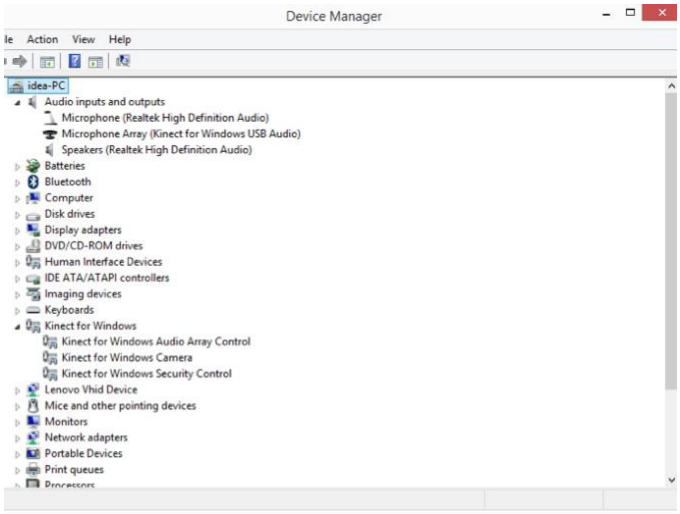

Figura 10:

4. Colocación del sensor

Después de haber conectado el sensor correctamente se debe de posicionar adecuadamente, así como también posicionar al paciente en cuestion.

A continuacion se muestran las especificaciones de ubicacion.

1. Colocar el sensor a una altura de alrededor de 1m, se recomienda que se una base firme (como una mesa).
2. El paciente debe de ser colocado a una distancia del sensor de 2.7m en línea recta al Kinect.
3. Se recomienda realizar las operaciones en un lugar con iluminacion constante, sin rayos de luz que incidan en el area de operacion del Kinect.
4. Por último se recomienda no utilizar ropa holgada, preferentemente ropa deportiva o ajustable para que no genere interferencia en la lectura del sensor.

5. Uso de la Interfaz

La interfaz se encuentra diseñada de una manera intuitiva y contiene un menú didactico pero aun así es necesario saber la serie de pasos necesarios a seguir para usarla correctamente.

A continuacion se numeran una serie de pasos recomendados para el uso de la interfaz.

1. Ejecutar el archivo KinectUAQvDaniel.
2. Al iniciarse la interfaz ir hacia la parte del menú e ingresar en el menú Archivo para ahí seleccionar nuevo.

Figura 11:

3. Introducir los datos del paciente así como también el nombre del operador.

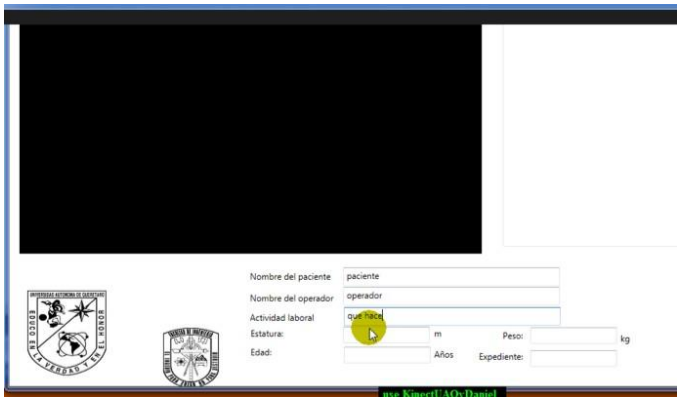

Figura 12:

4. Al haber ingresado todos los datos solicitados, dirigirse al menú nombrado sensor en el que vamos a iniciar el sensor simplemente presionando el menú que hace referencia a tal accion.

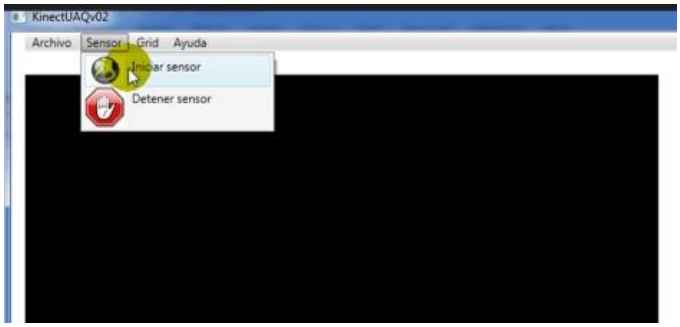

Figura 13:

5. Cuando el sensor ya haya iniciado se observara una imagen a color en el recuadro que se mostraba negro, y si algún individuo se encuentra en el rango apropiado lo detectara y al mismo tiempo realizara una esqueletizacion en tiempo real así como un analisis de su postura.

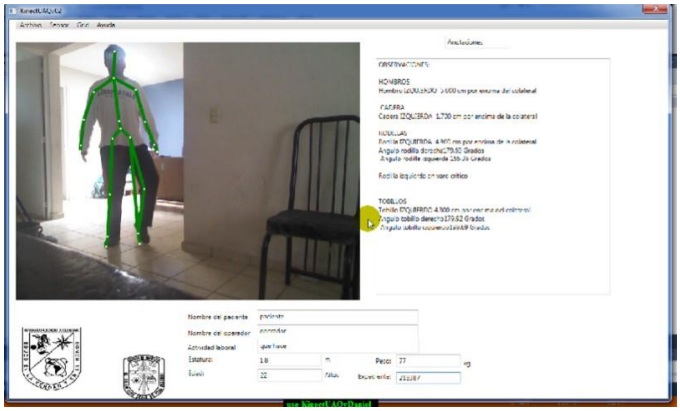

Figura 14:

6. Cuando usted observe que el paciente se encuentra en buena posicion para realizarle el analisis, se dirige nuevamente al menú sensor y presiona detener sensor para así guardar una captura del momento en el que presiona.
7. Para concluir, después de haber realizado todos los pasos antes mencionados se dirige al menú archivo- guardar para así guardar tanto la imagen como los datos obtenidos, ademas de generar un reporte en Excel con el expediente del paciente como nombre del archivo, el cual se abre automaticamente.

6. Notas finales

- En caso de no haber ingresado todos los datos requeridos e intentar guardar, aparecer un mensaje que nos recuerda que es necesario introducir todos los datos, ya que en caso contrario es imposible proseguir con la operacion.
- Si aparece un mensaje diciendo que el sensor Kinect no se encuentra conectado, simplemente revise su conexion y que el led color verde se encuentre encendido, de no ser así realizar nuevamente los pasos de la seccion II y III.
- Si en el momento de tratar de iniciar la esqueletizacion esta no se puede realizar, pedirle al paciente que se mueva un poco para que el sensor lo detecte, o en caso de que solo no detecte alguna parte del cuerpo, pedirle que mueva dicha parte.
- En dado caso de que después de realizar el paso anterior no detecte aún se recomienda reiniciar la interfaz y comenzar de nuevo.
- Si después de reiniciar la interfaz sigue sin funcionar, reiniciar la PC y comenzar de nuevo

- En caso de no poder hacer funcionar después de todo lo antes mencionado, se recomienda pedir apoyo técnico a la siguiente direccion: djaramilllo5@gmail.com.

Apéndice C

Código

1 MainWindow.xaml

```
<Window x:Class="KinectUAQv02.MainWindow"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Title="KinectUAQv02" Height="745.388" Width="1242" Loaded="WindowLoaded"
          Closing="WindowClosing" ResizeMode="NoResize">
 <Grid Name="layoutGrid" Margin="14,0,-39,-28" HorizontalAlignment="Left"
     Width="1261">
     <Grid.RowDefinitions>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="151*" />
     </Grid.RowDefinitions>
     <!--<StackPanel>-->
     <Image Name="KinectRGB" Width="640" Height="480" HorizontalAlignment="
         Left" Margin="0,12,0,181" Grid.RowSpan="3" RenderTransformOrigin="
         0.5,0.5">
         <Image.RenderTransform>
             <TransformGroup>
                 <ScaleTransform ScaleY="1" ScaleX="-1"/>
                 <SkewTransform AngleY="0" AngleX="0"/>
                 <RotateTransform Angle="0"/>
                 <TranslateTransform/>
             </TransformGroup>
         </Image.RenderTransform>
     </Image>
     <!--<Image Name="KinectDepth" Width="320" Height="240"
         HorizontalAlignment="Center" />-->
     <Image Name="KinectSkelleton" Width="640" Height="480"
         HorizontalAlignment="Left" ImageFailed="KinectSkelleton_ImageFailed"
         Margin="0,12,0,181" Grid.Row="2" RenderTransformOrigin="0.5,0.5">
         <Image.RenderTransform>
             <TransformGroup>
                 <ScaleTransform ScaleY="1" ScaleX="-1"/>
                 <SkewTransform AngleY="0" AngleX="0"/>
                 <RotateTransform Angle="0"/>
                 <TranslateTransform/>
             </TransformGroup>
         </Image.RenderTransform>
     </Image>
     <!--</StackPanel>
     <StackPanel>-->
     <!--</StackPanel>-->
     <StatusBar Grid.Row="3" HorizontalAlignment="Stretch" Name="statusBar"
         VerticalAlignment="Bottom" Background="White">
         <StatusBarItem Padding="0 0 0 10"></StatusBarItem>
     </StatusBar>
     <TextBlock Name="statusBarText" Margin="10,512,869,140" Grid.Row="2"></
         TextBlock>
     <TextBlock Grid.Row="2" Margin="869,630,-25,24" Name="textBlock1">
         Facultad de Ingenier a Campus San Juan del R o </TextBlock>
```

```
<TextBlock Grid.Row="2" Margin="869,603,10,51" Name="textBlock2">
    Universidad Aut noma de Quer taro UAQ</TextBlock>
<TextBox Grid.Row="2" Height="26" HorizontalAlignment="Left" Margin="
    448,521,0,0" Name="textBox1" VerticalAlignment="Top" Width="300" />
<TextBox Grid.Row="2" Height="26" HorizontalAlignment="Left" Margin="
```

```
    448,548,0,0" Name="textBox2" VerticalAlignment="Top" Width="300" />
<TextBox x:Name="textBox4" HorizontalAlignment="Left" Height="23" Margin
    ="707,604,0,0" Grid.Row="2" TextWrapping="Wrap" VerticalAlignment="
    Top" Width="120"/>
<TextBox x:Name="textBox5" HorizontalAlignment="Left" Height="23" Margin
    ="448,603,0,0" Grid.Row="2" TextWrapping="Wrap" VerticalAlignment="
    Top" Width="120"/>
<TextBox Grid.Row="2" Height="278" HorizontalAlignment="Left" Margin="
    669,47,0,0" Name="textBox3" VerticalAlignment="Top" Width="436" />
<RadioButton Content="Anterior" x:Name="radioButton1"
    HorizontalAlignment="Left" Margin="816,522,0,0" Grid.Row="2"
    VerticalAlignment="Top" GroupName="Group1" Checked="
    RadioButton_Checked" IsChecked="True" Visibility="Hidden"/>
<TextBlock HorizontalAlignment="Left" Margin="669,607,0,0" Grid.Row="2"
    TextWrapping="Wrap" Text="Peso:" VerticalAlignment="Top"/>
<TextBlock HorizontalAlignment="Left" Margin="320,604,0,0" Grid.Row="2"
    TextWrapping="Wrap" Text="Estatura:" VerticalAlignment="Top"/>
<TextBlock HorizontalAlignment="Left" Margin="832,611,0,0" Grid.Row="2"
    TextWrapping="Wrap" Text="kg" VerticalAlignment="Top"/>
<TextBlock HorizontalAlignment="Left" Margin="574,604,0,0" Grid.Row="2"
    TextWrapping="Wrap" Text="m" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="22" Margin="802,374,0,0"
    Grid.Row="2" TextWrapping="Wrap" Text="Anotaciones del operador"
    VerticalAlignment="Top" Width="162"/>
<Menu>
    <MenuItem Header="Archivo"  >
        <MenuItem Header="Nuevo" Click="MenuItem_Click_1">
            <MenuItem.Icon>
                <Image Source="C:\KinectUAQvDaniel\KinectUAQv02\media\
                    nuevo.png"></Image>
            </MenuItem.Icon>

        </MenuItem>
        <MenuItem Header="Guardar" Click="MenuItem_Click_2">
            <MenuItem.Icon>
                <Image Source="C:\KinectUAQvDaniel\KinectUAQv02\media\
                    SaveAs.png"></Image>
            </MenuItem.Icon>

        </MenuItem>
        <MenuItem Header="Salir" Click="MenuItem_Click_3">
            <MenuItem.Icon>
                <Image Source="C:\KinectUAQvDaniel\KinectUAQv02\media\
                    btn_cerrar_off.png"></Image>
            </MenuItem.Icon>

        </MenuItem>
    </MenuItem>
    <MenuItem Header="Sensor">
        <MenuItem Header="Iniciar sensor" Click="MenuItem_Click_4">
            <MenuItem.Icon>
                <Image Source="C:\KinectUAQvDaniel\KinectUAQv02\media\
                    iniciarsensor.jpg"></Image>
            </MenuItem.Icon>

        </MenuItem>
        <MenuItem Header="Detener sensor" Click="MenuItem_Click_5">
```

```
            <MenuItem.Icon>
                <Image Source="C:\KinectUAQvDaniel\KinectUAQv02\media\
                    detener.jpg"></Image>
            </MenuItem.Icon>

        </MenuItem>
    </MenuItem>
    <MenuItem Header="Grid">
        <MenuItem Header="Grid on" Click="MenuItem_Click_6">

        </MenuItem>
        <MenuItem Header="Grid off" Click="MenuItem_Click_7">

        </MenuItem>
    </MenuItem>
    <MenuItem Header="Ayuda" >
        <MenuItem Header="Manual de usuario" Click="MenuItem_Click_8" >

        </MenuItem>
        <MenuItem Header="Contactanos" Click="MenuItem_Click_9"  >

        </MenuItem>
    </MenuItem>
</Menu>
<Rectangle HorizontalAlignment="Left" Height="120" Margin="10,552,0,0"
    Grid.Row="2" Stroke="White" VerticalAlignment="Top" Width="111"
    ScrollViewer.VerticalScrollBarVisibility="Hidden">
    <Rectangle.Fill>
        <ImageBrush ImageSource="C:\KinectUAQvDaniel\KinectUAQv02\media\
            Escudo-UAQ.JPG"/>
    </Rectangle.Fill>
</Rectangle>
<Rectangle HorizontalAlignment="Left" Height="85" Margin="226,582,0,0"
    Grid.Row="2" Stroke="White" VerticalAlignment="Top" Width="79"
    ScrollViewer.VerticalScrollBarVisibility="Hidden">
    <Rectangle.Fill>
        <ImageBrush ImageSource="C:\KinectUAQvDaniel\KinectUAQv02\media\
            e-ingenieria2.JPG"/>
    </Rectangle.Fill>
</Rectangle>
<Button x:Name="iniciar" Content="" HorizontalAlignment="Left" Margin="
    0,5,0,0" Grid.Row="2" VerticalAlignment="Top" Width="1117" Height="
    667" Click="iniciar_Click" Visibility="Hidden">
    <Button.Background>
        <ImageBrush ImageSource="C:\KinectUAQvDaniel\KinectUAQv02\media\
            Escudo-UAQ.JPG"/>
    </Button.Background>
</Button>
<TextBox x:Name="eda" HorizontalAlignment="Left" Height="23" Margin="
    449,631,0,0" Grid.Row="2" TextWrapping="Wrap" VerticalAlignment="Top"
     Width="120"/>
<TextBlock HorizontalAlignment="Left" Margin="320,630,0,0" Grid.Row="2"
    TextWrapping="Wrap" Text="Edad:" VerticalAlignment="Top"/>
<TextBlock HorizontalAlignment="Left" Margin="574,633,0,0" Grid.Row="2"
    TextWrapping="Wrap" Text="A os" VerticalAlignment="Top"
    RenderTransformOrigin="0.444,-0.689"/>
<TextBlock Grid.Row="2" Margin="319,579,1045,113" x:Name="
```

```
      textBlock3_Copy"><Run Text="Actividad lavoral:"/></TextBlock>
  <TextBox Grid.Row="2" Height="26" HorizontalAlignment="Left" Margin="
      449,576,0,0" x:Name="actividad" VerticalAlignment="Top" Width="300"
      />
  <TextBlock HorizontalAlignment="Left" Margin="636,636,0,0" Grid.Row="2"
      TextWrapping="Wrap" Text="Expediente:" VerticalAlignment="Top"/>
  <TextBox x:Name="exp" HorizontalAlignment="Left" Height="23" Margin="
      707,636,0,0" Grid.Row="2" TextWrapping="Wrap" VerticalAlignment="Top"
       Width="120"/>
  <TextBlock HorizontalAlignment="Left" Margin="319,525,0,0" Grid.Row="2"
      TextWrapping="Wrap" Text="Nombre del paciente" VerticalAlignment="Top
      "/>
  <TextBlock HorizontalAlignment="Left" Margin="319,555,0,0" Grid.Row="2"
      TextWrapping="Wrap" Text="Nombre del operador" VerticalAlignment="Top
      "/>
  <TextBlock HorizontalAlignment="Left" Margin="319,582,0,0" Grid.Row="2"
      TextWrapping="Wrap" Text="Actividad laboral" VerticalAlignment="Top"
      />

  <Canvas Name="canvasMiHoja">
      <Ellipse
         Name="rojo" Canvas
         .Top="100" Canvas.
         Left="1120" Fill="
         White"
Height="25" Width
="25"
StrokeThickness="2"
Stroke="Blue"/>
      <Ellipse
          Name="amarillo"
         Canvas.Top="125"
         Canvas.Left="1120"
         Fill="White"
Height="25" Width
="25"
StrokeThickness="2"
Stroke="Blue"/>
      <Ellipse
          Name="verde"
         Canvas.Top="150"
         Canvas.Left="1120"
         Fill="Green"
Height="25" Width
="25"
StrokeThickness="2"
Stroke="Blue"/>
      <Ellipse
         Name="rojo1"
         Canvas.Top="100"
         Canvas.Left="1180"
         Fill="White"
Height="25" Width
="25"
StrokeThickness="2"
Stroke="Blue"/>
      <Ellipse
```

```
          Name="amarillo1"
         Canvas.Top="125"
         Canvas.Left="1180"
         Fill="White"
Height="25" Width
="25"
StrokeThickness="2"
Stroke="Blue"/>
      <Ellipse
          Name="verde1"
         Canvas.Top="150"
         Canvas.Left="1180"
         Fill="Green"
Height="25" Width
="25"
StrokeThickness="2"
Stroke="Blue"/>
      <Line X1="0" Y1="50" X2="660" Y2="50" Stroke="Red" StrokeThickness="
          2" Name="l1"/>
      <Line X1="0" Y1="110" X2="660" Y2="110" Stroke="Red" StrokeThickness
          ="2" Name="l2"/>
      <Line X1="0" Y1="170" X2="660" Y2="170" Stroke="Red" StrokeThickness
          ="2" Name="l3"/>
      <Line X1="0" Y1="230" X2="660" Y2="230" Stroke="Red" StrokeThickness
          ="2" Name="l4"/>
      <Line X1="0" Y1="290" X2="660" Y2="290" Stroke="Red" StrokeThickness
          ="2" Name="l5"/>
      <Line X1="0" Y1="350" X2="660" Y2="350" Stroke="Red" StrokeThickness
          ="2" Name="l6"/>
      <Line X1="0" Y1="410" X2="660" Y2="410" Stroke="Red" StrokeThickness
          ="2" Name="l7"/>
      <Line X1="0" Y1="470" X2="660" Y2="470" Stroke="Red" StrokeThickness
          ="2" Name="l8"/>
      <Line X1="0" Y1="530" X2="660" Y2="530" Stroke="Red" StrokeThickness
          ="2" Name="l21"/>
      <Line X1="0" Y1="50" X2="0" Y2="530" Stroke="Red" StrokeThickness="2
          " Name="l9"/>
      <Line X1="60" Y1="50" X2="60" Y2="530" Stroke="Red" StrokeThickness=
          "2" Name="l10"/>
      <Line X1="120" Y1="50" X2="120" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l11"/>
      <Line X1="180" Y1="50" X2="180" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l12"/>
      <Line X1="240" Y1="50" X2="240" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l13"/>
      <Line X1="300" Y1="50" X2="300" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l14"/>
      <Line X1="360" Y1="50" X2="360" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l15"/>
      <Line X1="420" Y1="50" X2="420" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l16"/>
      <Line X1="480" Y1="50" X2="480" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l17"/>
      <Line X1="540" Y1="50" X2="540" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l18"/>
      <Line X1="600" Y1="50" X2="600" Y2="530" Stroke="Red"
          StrokeThickness="2" Name="l19"/>
```

```
            <Line X1="660" Y1="50" X2="660" Y2="530" Stroke="Red"
                StrokeThickness="2" Name="l20"/>
        </Canvas>
        <TextBlock HorizontalAlignment="Left" Margin="1141,13,0,0" Grid.Row="2"
            TextWrapping="Wrap" Text="Rodillas" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="1110,52,0,0" Grid.Row="2"
            TextWrapping="Wrap" Text="Izquierda" VerticalAlignment="Top"/>
        <TextBlock HorizontalAlignment="Left" Margin="1164,52,0,0" Grid.Row="2"
            TextWrapping="Wrap" Text="Derecha" VerticalAlignment="Top"/>
        <Rectangle HorizontalAlignment="Left" Height="85" Margin="126,579,0,0"
            Grid.Row="2" Stroke="White" VerticalAlignment="Top" Width="79"
            ScrollViewer.VerticalScrollBarVisibility="Hidden">
            <Rectangle.Fill>
                <ImageBrush ImageSource="C:\KinectUAQvDaniel\KinectUAQv02\media\
                    escudofisioterapia.jpg"/>
            </Rectangle.Fill>
        </Rectangle>
        <TextBox Grid.Row="2" Height="76" HorizontalAlignment="Left" Margin="
            669,431,0,0" x:Name="anota" VerticalAlignment="Top" Width="436" />
        <TextBox HorizontalAlignment="Left" Height="22" Margin="788,25,0,0" Grid
            .Row="2" TextWrapping="Wrap" Text="Observaciones de la interfaz"
            VerticalAlignment="Top" Width="162"/>
    </Grid>
</Window>
```

2 MainWindow.xaml.cs

```
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;

using System.Reflection;
using Microsoft.Office.Interop.Excel;
using Microsoft.Office.Core;

using System.Diagnostics;
using System.ComponentModel;


namespace KinectUAQv02
{
    using System;
    using System.Globalization;
    using System.IO;
    using System.Windows;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using Microsoft.Kinect;



    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        string coordinates;

        double rodilladerecha;
        double rodillaizquierda;



        /// <summary>
        /// Active Kinect sensor
        /// </summary>
        private KinectSensor sensor;

        // Color Basics
```

```
/// <summary>
/// Bitmap that will hold color information
/// </summary>
private WriteableBitmap colorBitmap;

/// <summary>
/// Intermediate storage for the color data received from the camera
/// </summary>
private byte[] colorPixels;

// Color Basics

// Depth Basics

/// <summary>
/// Bitmap that will hold color information
/// </summary>
//private WriteableBitmap depthBitmap;

/// <summary>
/// Intermediate storage for the depth data received from the camera
/// </summary>
//private DepthImagePixel[] depthPixels;

/// <summary>
/// Intermediate storage for the depth data converted to color
/// </summary>
//private byte[] colorDepth;

// Depth Basics

// Skelleton Basics

/// <summary>
/// Width of output drawing
/// </summary>
private const float RenderWidth = 640.0f;

/// <summary>
/// Height of our output drawing
/// </summary>
private const float RenderHeight = 480.0f;

/// <summary>
/// Thickness of drawn joint lines
/// </summary>
private const double JointThickness = 2;

/// <summary>
/// Thickness of body center ellipse
/// </summary>
private const double BodyCenterThickness = 10;

/// <summary>
/// Thickness of clip edge rectangles
/// </summary>
```

```
private const double ClipBoundsThickness = 10;

/// <summary>
/// Brush used to draw skeleton center point
/// </summary>
private readonly Brush centerPointBrush = Brushes.Blue;

/// <summary>
/// Brush used for drawing joints that are currently tracked
/// </summary>
private readonly Brush trackedJointBrush = Brushes.White;

/// <summary>
/// Brush used for drawing joints that are currently inferred
/// </summary>
private readonly Brush inferredJointBrush = Brushes.Yellow;

/// <summary>
/// Pen used for drawing bones that are currently tracked
/// </summary>
//private readonly Pen trackedBonePen = new Pen(Brushes.Green, 6);
private readonly Pen trackedBonePen = new Pen(Brushes.Green, 6);

/// <summary>
/// Pen used for drawing bones that are currently inferred
/// </summary>
private readonly Pen inferredBonePen = new Pen(Brushes.Gray, 1);

/// <summary>
/// Drawing group for skeleton rendering output
/// </summary>
private DrawingGroup drawingGroup;

/// <summary>
/// Drawing image that we will display
/// </summary>
private DrawingImage imageSource;

// Skelleton Basics

/// Initializes a new instance of the MainWindow class.
public MainWindow()
{
    InitializeComponent();
}

/// <summary>
/// Draws indicators to show which edges are clipping skeleton data
/// </summary>
/// <param name="skeleton">skeleton to draw clipping information for</
    param>
/// <param name="drawingContext">drawing context to draw to</param>
private static void RenderClippedEdges(Skeleton skeleton, DrawingContext
     drawingContext)
{
    if (skeleton.ClippedEdges.HasFlag(FrameEdges.Bottom))
    {
```

```
            drawingContext.DrawRectangle(
                Brushes.Red,
                null,
                new Rect(0, RenderHeight - ClipBoundsThickness, RenderWidth,
                     ClipBoundsThickness));
        }

        if (skeleton.ClippedEdges.HasFlag(FrameEdges.Top))
        {
            drawingContext.DrawRectangle(
                Brushes.Red,
                null,
                new Rect(0, 0, RenderWidth, ClipBoundsThickness));
        }

        if (skeleton.ClippedEdges.HasFlag(FrameEdges.Left))
        {
            drawingContext.DrawRectangle(
                Brushes.Red,
                null,
                new Rect(0, 0, ClipBoundsThickness, RenderHeight));
        }

        if (skeleton.ClippedEdges.HasFlag(FrameEdges.Right))
        {
            drawingContext.DrawRectangle(
                Brushes.Red,
                null,
                new Rect(RenderWidth - ClipBoundsThickness, 0,
                    ClipBoundsThickness, RenderHeight));
        }
    }

    /// Execute startup tasks
    /// <param name="sender">object sending the event</param>
    /// <param name="e">event arguments</param>
    private void WindowLoaded(object sender, RoutedEventArgs e)
    {
        // Look through all sensors and start the first connected one.
        // This requires that a Kinect is connected at the time of app
            startup.
        // To make your app robust against plug/unplug,
        // it is recommended to use KinectSensorChooser provided in
            Microsoft.Kinect.Toolkit (See components in Toolkit Browser).
        foreach (var potentialSensor in KinectSensor.KinectSensors)
        {
            if (potentialSensor.Status == KinectStatus.Connected)
            {
                this.sensor = potentialSensor;
                break;
            }
        }

        if (null != this.sensor)
        {

            // Create the drawing group we'll use for drawing
```

```
this.drawingGroup = new DrawingGroup();
// Create an image source that we can use in our image control
this.imageSource = new DrawingImage(this.drawingGroup);
// Display the drawing using our image control
this.KinectSkelleton.Source = this.imageSource;
///this.KinectRGB.Source = this.imageSource;

// Turn on the color stream to receive color frames
this.sensor.ColorStream.Enable(ColorImageFormat.
    RgbResolution1280x960Fps12/*RgbResolution640x480Fps30*/);
// Allocate space to put the pixels we'll receive
this.colorPixels = new byte[this.sensor.ColorStream.
    FramePixelDataLength];
// This is the bitmap we'll display on-screen
this.colorBitmap = new WriteableBitmap(this.sensor.ColorStream.
    FrameWidth, this.sensor.ColorStream.FrameHeight, 96.0, 96.0,
    PixelFormats.Bgr32, null);
// Set the image we display to point to the bitmap where we'll
    put the image data
this.KinectRGB.Source = this.colorBitmap;
// Add an event handler to be called whenever there is new color
     frame data
this.sensor.ColorFrameReady += this.SensorColorFrameReady;

//// Turn on the depth stream to receive depth frames
//this.sensor.DepthStream.Enable(DepthImageFormat.
    Resolution640x480Fps30);
//// Allocate space to put the depth pixels we'll receive
//this.depthPixels = new DepthImagePixel[this.sensor.DepthStream
    .FramePixelDataLength];
//// Allocate space to put the color pixels we'll create
//this.colorDepth = new byte[this.sensor.DepthStream.
    FramePixelDataLength * sizeof(int)];
//// This is the bitmap we'll display on-screen
//this.depthBitmap = new WriteableBitmap(this.sensor.DepthStream
    .FrameWidth, this.sensor.DepthStream.FrameHeight, 96.0, 96.0,
     PixelFormats.Bgr32, null);
//// Set the image we display to point to the bitmap where we'll
     put the image data
//this.KinectDepth.Source = this.depthBitmap;
//// Add an event handler to be called whenever there is new
    depth frame data
//this.sensor.DepthFrameReady += this.SensorDepthFrameReady;

// Turn on the skeleton stream to receive skeleton frames
this.sensor.SkeletonStream.Enable();

// Add an event handler to be called whenever there is new color
     frame data
this.sensor.SkeletonFrameReady += this.SensorSkeletonFrameReady;

// Start the sensor!
try
{
    //this.sensor.Start();
}
catch (IOException)
```

```
        {
            this.sensor = null;
        }
    }

    if (null == this.sensor)
    {
        this.statusBarText.Text = Properties.Resources.NoKinectReady;
    }
}

/// <summary>
/// Execute shutdown tasks
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void WindowClosing(object sender, System.ComponentModel.
    CancelEventArgs e)
{
    if (null != this.sensor && this.sensor.IsRunning)
    {
        this.sensor.Stop();
    }
}

/// <summary>
/// Event handler for Kinect sensor's ColorFrameReady event
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void SensorColorFrameReady(object sender,
    ColorImageFrameReadyEventArgs e)
{
    using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
    {
        if (colorFrame != null)
        {
            // Copy the pixel data from the image to a temporary array
            colorFrame.CopyPixelDataTo(this.colorPixels);

            // Write the pixel data into our bitmap
            this.colorBitmap.WritePixels(
                new Int32Rect(0, 0, this.colorBitmap.PixelWidth, this.
                    colorBitmap.PixelHeight),
                this.colorPixels,
                this.colorBitmap.PixelWidth * sizeof(int),
                0);
        }
    }
}

//private void SensorDepthFrameReady(object sender,
    DepthImageFrameReadyEventArgs e)
//{
//    using (DepthImageFrame depthFrame = e.OpenDepthImageFrame())
//    {
//        if (depthFrame != null)
```

```
//        {
//            // Copy the pixel data from the image to a temporary array
//            depthFrame.CopyDepthImagePixelDataTo(this.depthPixels);

//            // Get the min and max reliable depth for the current
    frame
//            int minDepth = depthFrame.MinDepth;
//            int maxDepth = depthFrame.MaxDepth;

//            // Convert the depth to RGB
//            int colorDepthIndex = 0;
//            for (int i = 0; i < this.depthPixels.Length; ++i)
//            {
//                // Get the depth for this pixel
//                short depth = depthPixels[i].Depth;

//                // To convert to a byte, we're discarding the most-
    significant
//                // rather than least-significant bits.
//                // We're preserving detail, although the intensity
    will "wrap."
//                // Values outside the reliable depth range are mapped
    to 0 (black).

//                // Note: Using conditionals in this loop could degrade
     performance.
//                // Consider using a lookup table instead when writing
    production code.
//                // See the KinectDepthViewer class used by the
    KinectExplorer sample
//                // for a lookup table example.
//                byte intensity = (byte)(depth >= minDepth && depth <=
    maxDepth ? depth : 0);

//                // Write out blue byte
//                this.colorDepth[colorDepthIndex++] = intensity;

//                // Write out green byte
//                this.colorDepth[colorDepthIndex++] = intensity;

//                // Write out red byte
//                this.colorDepth[colorDepthIndex++] = intensity;

//                // We're outputting BGR, the last byte in the 32 bits
    is unused so skip it
//                // If we were outputting BGRA, we would write alpha
    here.
//                ++colorDepthIndex;
//            }

//            // Write the pixel data into our bitmap
//            this.depthBitmap.WritePixels(
//                new Int32Rect(0, 0, this.depthBitmap.PixelWidth, this.
    depthBitmap.PixelHeight),
//                this.colorDepth,
//                this.colorBitmap.PixelWidth * sizeof(int),
//                0);
```

```
//        }
//    }
//}

/// <summary>
/// Event handler for Kinect sensor's SkeletonFrameReady event
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void SensorSkeletonFrameReady(object sender,
    SkeletonFrameReadyEventArgs e)
{
    Skeleton[] skeletons = new Skeleton[0];

    using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
    {
        if (skeletonFrame != null)
        {
            skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
            skeletonFrame.CopySkeletonDataTo(skeletons);
        }
    }

    using (DrawingContext dc = this.drawingGroup.Open())
    {
        // Draw a transparent background to set the render size
        dc.DrawRectangle(Brushes.Transparent, null, new Rect(0.0, 0.0,
            RenderWidth, RenderHeight));

        if (skeletons.Length != 0)
        {
            foreach (Skeleton skel in skeletons)
            {
                RenderClippedEdges(skel, dc);

                if (skel.TrackingState == SkeletonTrackingState.Tracked)
                {
                    this.DrawBonesAndJoints(skel, dc);
                    coordinates = GetJointCordinates(skel);
                    PosturalAnalysis(skel);
                }
                else if (skel.TrackingState == SkeletonTrackingState.
                    PositionOnly)
                {
                    dc.DrawEllipse(
                    this.centerPointBrush,
                    null,
                    this.SkeletonPointToScreen(skel.Position),
                    BodyCenterThickness,
                    BodyCenterThickness);
                }
            }
        }

        // prevent drawing outside of our render area
        this.drawingGroup.ClipGeometry = new RectangleGeometry(new Rect
            (0.0, 0.0, RenderWidth, RenderHeight));
```

```
    }
}

/// <summary>
/// Draws a skeleton's bones and joints
/// </summary>
/// <param name="skeleton">skeleton to draw</param>
/// <param name="drawingContext">drawing context to draw to</param>
private void DrawBonesAndJoints(Skeleton skeleton, DrawingContext
    drawingContext)
{

    if (radioButton1.IsChecked.Value == true )
    {
        // Render Torso
        this.DrawBone(skeleton, drawingContext, JointType.Head,
            JointType.ShoulderCenter);
        this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter
            , JointType.ShoulderLeft);
        this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter
            , JointType.ShoulderRight);
        this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter
            , JointType.Spine);
        this.DrawBone(skeleton, drawingContext, JointType.Spine,
            JointType.HipCenter);
        this.DrawBone(skeleton, drawingContext, JointType.HipCenter,
            JointType.HipLeft);
        this.DrawBone(skeleton, drawingContext, JointType.HipCenter,
            JointType.HipRight);

        //JointType.Head.CompareTo(JointType.ShoulderCenter);

        // Left Arm
        this.DrawBone(skeleton, drawingContext, JointType.ShoulderLeft,
            JointType.ElbowLeft);
        this.DrawBone(skeleton, drawingContext, JointType.ElbowLeft,
            JointType.WristLeft);
        this.DrawBone(skeleton, drawingContext, JointType.WristLeft,
            JointType.HandLeft);

        // Right Arm
        this.DrawBone(skeleton, drawingContext, JointType.ShoulderRight,
             JointType.ElbowRight);
        this.DrawBone(skeleton, drawingContext, JointType.ElbowRight,
            JointType.WristRight);
        this.DrawBone(skeleton, drawingContext, JointType.WristRight,
            JointType.HandRight);

        // Left Leg
        this.DrawBone(skeleton, drawingContext, JointType.HipLeft,
            JointType.KneeLeft);
        this.DrawBone(skeleton, drawingContext, JointType.KneeLeft,
            JointType.AnkleLeft);
        this.DrawBone(skeleton, drawingContext, JointType.AnkleLeft,
            JointType.FootLeft);

        // Right Leg
```

```
    this.DrawBone(skeleton, drawingContext, JointType.HipRight,
        JointType.KneeRight);
    this.DrawBone(skeleton, drawingContext, JointType.KneeRight,
        JointType.AnkleRight);
    this.DrawBone(skeleton, drawingContext, JointType.AnkleRight,
        JointType.FootRight);

    if (rodilladerecha == 1)
    {
        this.DrawBone(skeleton, drawingContext, JointType.HipRight,
            JointType.AnkleRight);
    }
    if (rodillaizquierda == 1)
    {
        this.DrawBone(skeleton, drawingContext, JointType.HipLeft,
            JointType.AnkleLeft);
    }
}


else// if (radioButton3.IsChecked.Value == true)
{
    // Render Torso
    this.DrawBone(skeleton, drawingContext, JointType.Head,
        JointType.ShoulderCenter);
    this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter
        , JointType.ShoulderLeft);
    //this.DrawBone(skeleton, drawingContext, JointType.
        ShoulderCenter, JointType.ShoulderRight);
    this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter
        , JointType.Spine);
    this.DrawBone(skeleton, drawingContext, JointType.Spine,
        JointType.HipCenter);
    this.DrawBone(skeleton, drawingContext, JointType.HipCenter,
        JointType.HipLeft);
    //this.DrawBone(skeleton, drawingContext, JointType.HipCenter,
        JointType.HipRight);

    //JointType.Head.CompareTo(JointType.ShoulderCenter);

    // Left Arm
    //this.DrawBone(skeleton, drawingContext, JointType.ShoulderLeft
        , JointType.ElbowLeft);
    //this.DrawBone(skeleton, drawingContext, JointType.ElbowLeft,
        JointType.WristLeft);
    //this.DrawBone(skeleton, drawingContext, JointType.WristLeft,
        JointType.HandLeft);

    // Right Arm
    //this.DrawBone(skeleton, drawingContext, JointType.
        ShoulderRight, JointType.ElbowRight);
    //this.DrawBone(skeleton, drawingContext, JointType.ElbowRight,
        JointType.WristRight);
    //this.DrawBone(skeleton, drawingContext, JointType.WristRight,
        JointType.HandRight);

    // Left Leg
```

```
            this.DrawBone(skeleton, drawingContext, JointType.HipLeft,
                JointType.KneeLeft);
            this.DrawBone(skeleton, drawingContext, JointType.KneeLeft,
                JointType.AnkleLeft);
            this.DrawBone(skeleton, drawingContext, JointType.AnkleLeft,
                JointType.FootLeft);

            // Right Leg
            //this.DrawBone(skeleton, drawingContext, JointType.HipRight,
                JointType.KneeRight);
            //this.DrawBone(skeleton, drawingContext, JointType.KneeRight,
                JointType.AnkleRight);
            //this.DrawBone(skeleton, drawingContext, JointType.AnkleRight,
                JointType.FootRight);
        }

        //this.DrawBone(skeleton, drawingContext, JointType.HandLeft,
            JointType.HandRight);

        ////////////////////////////////////////////////////////////

        //double distancehands = Math.Sqrt(Math.Pow(handleftx - handrightx,
            2) + Math.Pow(handlefty - handrighty, 2) + Math.Pow(handleftz -
            handrightz, 2));

        //textBox1.Text = handleftz.ToString();
        //textBox2.Text = handrightz.ToString();

        //textBox1.Text = distancehands.ToString();

        // Render Joints
        foreach (Joint joint in skeleton.Joints)
        {
            Brush drawBrush = null;

            if (joint.TrackingState == JointTrackingState.Tracked)
            {
                drawBrush = this.trackedJointBrush;
            }
            else if (joint.TrackingState == JointTrackingState.Inferred)
            {
                //drawBrush = this.inferredJointBrush;
            }

            if (drawBrush != null)
            {
                drawingContext.DrawEllipse(drawBrush, null, this.
                    SkeletonPointToScreen(joint.Position), JointThickness,
                    JointThickness);
            }
        }
    }

    /// <summary>
    /// Maps a SkeletonPoint to lie within our render space and converts to
        Point
    /// </summary>
```

```
/// <param name="skelpoint">point to map</param>
/// <returns>mapped point</returns>
private Point SkeletonPointToScreen(SkeletonPoint skelpoint)
{
    // Convert point to depth space.
    // We are not using depth directly, but we do want the points in our
        640x480 output resolution.
    DepthImagePoint depthPoint = this.sensor.CoordinateMapper.
        MapSkeletonPointToDepthPoint(skelpoint, DepthImageFormat.
        Resolution640x480Fps30);
    return new Point(depthPoint.X, depthPoint.Y);
}

/// <summary>
/// Draws a bone line between two joints
/// </summary>
/// <param name="skeleton">skeleton to draw bones from</param>
/// <param name="drawingContext">drawing context to draw to</param>
/// <param name="jointType0">joint to start drawing from</param>
/// <param name="jointType1">joint to end drawing at</param>
private void DrawBone(Skeleton skeleton, DrawingContext drawingContext,
    JointType jointType0, JointType jointType1)
{
    Joint joint0 = skeleton.Joints[jointType0];
    Joint joint1 = skeleton.Joints[jointType1];

    // If we can't find either of these joints, exit
    if (joint0.TrackingState == JointTrackingState.NotTracked ||
        joint1.TrackingState == JointTrackingState.NotTracked)
    {
        return;
    }

    // Don't draw if both points are inferred
    if (joint0.TrackingState == JointTrackingState.Inferred &&
        joint1.TrackingState == JointTrackingState.Inferred)
    {
        return;
    }

    // We assume all drawn bones are inferred unless BOTH joints are
        tracked
    Pen drawPen = this.inferredBonePen;
    if (joint0.TrackingState == JointTrackingState.Tracked && joint1.
        TrackingState == JointTrackingState.Tracked)
    {
        drawPen = this.trackedBonePen;
    }

    drawingContext.DrawLine(drawPen, this.SkeletonPointToScreen(joint0.
        Position), this.SkeletonPointToScreen(joint1.Position));
}

/// <summary>
/// Handles the checking or unchecking of the seated mode combo box
/// </summary>
/// <param name="sender">object sending the event</param>
```

```
/// <param name="e">event arguments</param>
/*
private void CheckBoxSeatedModeChanged(object sender, RoutedEventArgs e)
{
    if (null != this.sensor)
    {
        if (this.checkBoxSeatedMode.IsChecked.GetValueOrDefault())
        {
            this.sensor.SkeletonStream.TrackingMode =
                SkeletonTrackingMode.Seated;
        }
        else
        {
            this.sensor.SkeletonStream.TrackingMode =
                SkeletonTrackingMode.Default;
        }
    }
}*/


/// <summary>
/// Handles the user clicking on the screenshot button
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void ButtonScreenshotClick(object sender, RoutedEventArgs e)
{
    if (null == this.sensor)
    {
        this.statusBarText.Text = Properties.Resources.
            ConnectDeviceFirst;
        return;
    }

    // create a png bitmap encoder which knows how to save a .png file
    BitmapEncoder encoder = new PngBitmapEncoder();

    // create frame from the writable bitmap and add to encoder
    encoder.Frames.Add(BitmapFrame.Create(this.colorBitmap));
    //encoder.Frames.Add(BitmapFrame.Create(this.DrawBone)); // Prueba
        para guardar im gen de skelleton

    //string time = System.DateTime.Now.ToString("hh'-'mm'-'ss",
        CultureInfo.CurrentUICulture.DateTimeFormat);
    //string time = System.DateTime.Now.ToString("yyyy MM d HH 'hr' mm '
        min' ss 'seg'", CultureInfo.CurrentUICulture.DateTimeFormat);
    string time = System.DateTime.Now.ToString("yyyy'-'MM'-'d HH'.'mm'.'
        ss", CultureInfo.CurrentUICulture.DateTimeFormat);

    string myPhotos = Environment.GetFolderPath(Environment.
        SpecialFolder.MyPictures);

    string path = Path.Combine(myPhotos, time + " Color.png");

    // write the new file to disk
    try
    {
        using (FileStream fs = new FileStream(path, FileMode.Create))
```

```
        {
            encoder.Save(fs);
        }

        this.statusBarText.Text = string.Format("{0} {1}", Properties.
            Resources.ScreenshotWriteSuccess, path);
    }
    catch (IOException)
    {
        this.statusBarText.Text = string.Format("{0} {1}", Properties.
            Resources.ScreenshotWriteFailed, path);
    }

    string[] lines = { textBlock2.Text, textBlock1.Text, "\n",
                       "Fecha y hora de prueba: " + time, "Operador: "
                            + textBox2.Text, "\n",
                       "Paciente: " + textBox1.Text, "Peso: " +
                           textBox4.Text + " kg", "Estatura: " +
                           textBox5.Text + " m", "\n",
                       textBox3.Text, "\n",
                       "COORDENADAS", coordinates };

    ScreenCapturer.CaptureAndSave("C:\\UAQKinectPostural\\" + time + "
        Interfaz.png", CaptureMode.Window, System.Drawing.Imaging.
        ImageFormat.Png);
    System.IO.File.WriteAllLines("C:\\UAQKinectPostural\\" + time + ".
        txt", lines);
}

private void KinectSkelleton_ImageFailed(object sender,
    ExceptionRoutedEventArgs e)
{

}

private void button1_Click(object sender, RoutedEventArgs e)
{
    if(this.sensor != null && !this.sensor.IsRunning)
    {
        this.sensor.Start();
    }
}

private void button2_Click(object sender, RoutedEventArgs e)
{
    if(this.sensor != null && this.sensor.IsRunning)
    {
        this.sensor.Stop();
    }
}

private string GetJointCordinates(Skeleton skeleton)
{
    ////////////////////////////////////////////////////////////
    // Torso
    var head = skeleton.Joints[JointType.Head];
```

```
float headx = head.Position.X;
float heady = head.Position.Y;
float headz = head.Position.Z;

var shouldercenter = skeleton.Joints[JointType.ShoulderCenter];

float shoulderx = shouldercenter.Position.X;
float shouldery = shouldercenter.Position.Y;
float shoulderz = shouldercenter.Position.Z;

var shoulderleft = skeleton.Joints[JointType.ShoulderLeft];

float shoulderleftx = shoulderleft.Position.X;
float shoulderlefty = shoulderleft.Position.Y;
float shoulderleftz = shoulderleft.Position.Z;

var shoulderright = skeleton.Joints[JointType.ShoulderRight];

float shoulderrightx = shoulderright.Position.X;
float shoulderrighty = shoulderright.Position.Y;
float shoulderrightz = shoulderright.Position.Z;

var spine = skeleton.Joints[JointType.Spine];

float spinex = spine.Position.X;
float spiney = spine.Position.Y;
float spinez = spine.Position.Z;

var hipcenter = skeleton.Joints[JointType.HipCenter];

float hipcenterx = hipcenter.Position.X;
float hipcentery = hipcenter.Position.Y;
float hipcenterz = hipcenter.Position.Z;

var hipleft = skeleton.Joints[JointType.HipLeft];

float hipleftx = hipleft.Position.X;
float hiplefty = hipleft.Position.Y;
float hipleftz = hipleft.Position.Z;

var hipright = skeleton.Joints[JointType.HipRight];

float hiprightx = hipright.Position.X;
float hiprighty = hipright.Position.Y;
float hiprightz = hipright.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Left Arm
var elbowleft = skeleton.Joints[JointType.ElbowLeft];

float elbowleftx = elbowleft.Position.X;
float elbowlefty = elbowleft.Position.Y;
float elbowleftz = elbowleft.Position.Z;

var wristleft = skeleton.Joints[JointType.WristLeft];
```

```
float wristleftx = wristleft.Position.X;
float wristlefty = wristleft.Position.Y;
float wristleftz = wristleft.Position.Z;

var handleft = skeleton.Joints[JointType.HandLeft];

float handleftx = handleft.Position.X;
float handlefty = handleft.Position.Y;
float handleftz = handleft.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Right Arm
var elbowright = skeleton.Joints[JointType.ElbowRight];

float elbowrightx = elbowright.Position.X;
float elbowrighty = elbowright.Position.Y;
float elbowrightz = elbowright.Position.Z;

var wristright = skeleton.Joints[JointType.WristRight];

float wristrightx = wristright.Position.X;
float wristrighty = wristright.Position.Y;
float wristrightz = wristright.Position.Z;

var handright = skeleton.Joints[JointType.HandRight];

float handrightx = handright.Position.X;
float handrighty = handright.Position.Y;
float handrightz = handright.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Left Leg
var kneeleft = skeleton.Joints[JointType.KneeLeft];

float kneeleftx = kneeleft.Position.X;
float kneelefty = kneeleft.Position.Y;
float kneeleftz = kneeleft.Position.Z;

var ankleleft = skeleton.Joints[JointType.AnkleLeft];

float ankleleftx = ankleleft.Position.X;
float anklelefty = ankleleft.Position.Y;
float ankleleftz = ankleleft.Position.Z;

var footleft = skeleton.Joints[JointType.FootLeft];

float footleftx = footleft.Position.X;
float footlefty = footleft.Position.Y;
float footleftz = footleft.Position.Z;

//////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////
// Right Leg
var kneeright = skeleton.Joints[JointType.KneeRight];

float kneerightx = kneeright.Position.X;
float kneerighty = kneeright.Position.Y;
float kneerightz = kneeright.Position.Z;

var ankleright = skeleton.Joints[JointType.AnkleRight];

float anklerightx = ankleright.Position.X;
float anklerighty = ankleright.Position.Y;
float anklerightz = ankleright.Position.Z;

var footright = skeleton.Joints[JointType.FootRight];

float footrightx = footright.Position.X;
float footrighty = footright.Position.Y;
float footrightz = footright.Position.Z;

string coords = "Cabeza = " + headx + ", " + heady + ", " + headz +
    "\n " +
                "Hombro centro = " + shouldercenterx + ", " +
                    shouldercentery + ", " + shouldercenterz + "\n "
                     +
                "Hombro izquierdo = " + shoulderleftx + ", " +
                    shoulderlefty + ", " + shoulderleftz + "\n " +
                "Hombro derecho = " + shoulderrightx + ", " +
                    shoulderrighty + ", " + shoulderrightz + "\n " +
                "Columna vertebral = " + spinex + ", " + spiney + "
                    , " + spinez + "\n " +
                "Cadera centro = " + hipcenterx + ", " + hipcentery
                     + ", " + hipcenterz + "\n " +
                "Cadera izquierda = " + hipleftx + ", " + hiplefty
                    + ", " + hipleftz + "\n " +
                "Cadera derecha = " + hiprightx + ", " + hiprighty
                    + ", " + hiprightz + "\n " +

                "Codo izquierdo = " + elbowleftx + ", " +
                    elbowlefty + ", " + elbowleftz + "\n " +
                "Mu eca izquierdo = " + wristleftx + ", " +
                    wristlefty + ", " + wristleftz + "\n " +
                "Mano izquierdo = " + handleftx + ", " + handlefty
                    + ", " + handleftz + "\n " +

                "Codo derecho = " + elbowrightx + ", " +
                    elbowrighty + ", " + elbowrightz + "\n " +
                "Mu eca derecha = " + wristrightx + ", " +
                    wristrighty + ", " + wristrightz + "\n " +
                "Mano derecha = " + handrightx + ", " + handrighty
                    + ", " + handrightz + "\n " +

                "Rodilla izquierda = " + kneeleftx + ", " +
                    kneelefty + ", " + kneeleftz + "\n " +
                "Tobillo izquierdo = " + ankleleftx + ", " +
                    anklelefty + ", " + ankleleftz + "\n " +
                "Pie izquierdo = " + footleftx + ", " + footlefty +
```

```
                        ", " + footleftz + "\n " +

                    "Rodilla derecha = " + kneerightx + ", " +
                        kneerighty + ", " + kneerightz + "\n " +
                    "Tobillo derecho = " + anklerightx + ", " +
                        anklerighty + ", " + anklerightz + "\n " +
                    "Pie derecho = " + footrightx + ", " + footrighty +
                         ", " + footrightz + "\n ";

        return coords;
    }

    private void PosturalAnalysis(Skeleton skeleton)
    {
        ////////////////////////////////////////////////////////////
        // Torso
        float headx;
        float heady;
        float headz;

        float shouldercenterx;
        float shouldercentery;
        float shouldercenterz;

        float shoulderleftx;
        float shoulderlefty;
        float shoulderleftz;

        float shoulderrightx;
        float shoulderrighty;
        float shoulderrightz;

        float spinex;
        float spiney;
        float spinez;

        float hipcenterx;
        float hipcentery;
        float hipcenterz;

        float hipleftx;
        float hiplefty;
        float hipleftz;

        float hiprightx;
        float hiprighty;
        float hiprightz;
        ////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////////////
        // Left Arm
        float elbowleftx;
        float elbowlefty;
        float elbowleftz;

        float wristleftx;
        float wristlefty;
```

```
float wristleftz;

float handleftx;
float handlefty;
float handleftz;
//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Right Arm
float elbowrightx;
float elbowrighty;
float elbowrightz;

float wristrightx;
float wristrighty;
float wristrightz;

float handrightx;
float handrighty;
float handrightz;
//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Left Leg
float kneeleftx;
float kneelefty;
float kneeleftz;

float ankleleftx;
float anklelefty;
float ankleleftz;

float footleftx;
float footlefty;
float footleftz;
//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Right Leg
float kneerightx;
float kneerighty;
float kneerightz;

float anklerightx;
float anklerighty;
float anklerightz;

float footrightx;
float footrighty;
float footrightz;
//////////////////////////////////////////////////////////////

if (radioButton1.IsChecked == true )
{
    //////////////////////////////////////////////////////////////
    // Torso
    var head = skeleton.Joints[JointType.Head];
```

```
headx = head.Position.X;
heady = head.Position.Y;
headz = head.Position.Z;

var shouldercenter = skeleton.Joints[JointType.ShoulderCenter];

shouldercenterx = shouldercenter.Position.X;
shoulderCentery = shouldercenter.Position.Y;
shoulderCenterz = shouldercenter.Position.Z;

var shoulderleft = skeleton.Joints[JointType.ShoulderLeft];

shoulderleftx = shoulderleft.Position.X;
shoulderlefty = shoulderleft.Position.Y;
shoulderleftz = shoulderleft.Position.Z;

var shoulderright = skeleton.Joints[JointType.ShoulderRight];

shoulderrightx = shoulderright.Position.X;
shoulderrighty = shoulderright.Position.Y;
shoulderrightz = shoulderright.Position.Z;

var spine = skeleton.Joints[JointType.Spine];

spinex = spine.Position.X;
spiney = spine.Position.Y;
spinez = spine.Position.Z;

var hipcenter = skeleton.Joints[JointType.HipCenter];

hipcenterx = hipcenter.Position.X;
hipcentery = hipcenter.Position.Y;
hipcenterz = hipcenter.Position.Z;

var hipleft = skeleton.Joints[JointType.HipLeft];

hipleftx = hipleft.Position.X;
hiplefty = hipleft.Position.Y;
hipleftz = hipleft.Position.Z;

var hipright = skeleton.Joints[JointType.HipRight];

hiprightx = hipright.Position.X;
hiprighty = hipright.Position.Y;
hiprightz = hipright.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Left Arm
var elbowleft = skeleton.Joints[JointType.ElbowLeft];

elbowleftx = elbowleft.Position.X;
elbowlefty = elbowleft.Position.Y;
elbowleftz = elbowleft.Position.Z;
```

```
var wristleft = skeleton.Joints[JointType.WristLeft];

wristleftx = wristleft.Position.X;
wristlefty = wristleft.Position.Y;
wristleftz = wristleft.Position.Z;

var handleft = skeleton.Joints[JointType.HandLeft];

handleftx = handleft.Position.X;
handlefty = handleft.Position.Y;
handleftz = handleft.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Right Arm
var elbowright = skeleton.Joints[JointType.ElbowRight];

elbowrightx = elbowright.Position.X;
elbowrighty = elbowright.Position.Y;
elbowrightz = elbowright.Position.Z;

var wristright = skeleton.Joints[JointType.WristRight];

wristrightx = wristright.Position.X;
wristrighty = wristright.Position.Y;
wristrightz = wristright.Position.Z;

var handright = skeleton.Joints[JointType.HandRight];

handrightx = handright.Position.X;
handrighty = handright.Position.Y;
handrightz = handright.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Left Leg
var kneeleft = skeleton.Joints[JointType.KneeLeft];

kneeleftx = kneeleft.Position.X;
kneelefty = kneeleft.Position.Y;
kneeleftz = kneeleft.Position.Z;

var ankleleft = skeleton.Joints[JointType.AnkleLeft];

ankleleftx = ankleleft.Position.X;
anklelefty = ankleleft.Position.Y;
ankleleftz = ankleleft.Position.Z;

var footleft = skeleton.Joints[JointType.FootLeft];

footleftx = footleft.Position.X;
footlefty = footleft.Position.Y;
footleftz = footleft.Position.Z;

//////////////////////////////////////////////////////////////
```

```
        ////////////////////////////////////////////////////////////
        // Right Leg
        var kneeright = skeleton.Joints[JointType.KneeRight];

        kneerightx = kneeright.Position.X;
        kneerighty = kneeright.Position.Y;
        kneerightz = kneeright.Position.Z;

        var ankleright = skeleton.Joints[JointType.AnkleRight];

        anklerightx = ankleright.Position.X;
        anklerighty = ankleright.Position.Y;
        anklerightz = ankleright.Position.Z;

        var footright = skeleton.Joints[JointType.FootRight];

        footrightx = footright.Position.X;
        footrighty = footright.Position.Y;
        footrightz = footright.Position.Z;
    }
    else
    {
        ////////////////////////////////////////////////////////////
        // Torso
        var head = skeleton.Joints[JointType.Head];

        headx = head.Position.X;
        heady = head.Position.Y;
        headz = head.Position.Z;

        var shouldercenter = skeleton.Joints[JointType.ShoulderCenter];

        shouldercenterx = shouldercenter.Position.X;
        shouldercentery = shouldercenter.Position.Y;
        shouldercenterz = shouldercenter.Position.Z;

        var shoulderleft = skeleton.Joints[JointType.ShoulderRight]; //
            Right intead of Left

        shoulderleftx = shoulderleft.Position.X;
        shoulderlefty = shoulderleft.Position.Y;
        shoulderleftz = shoulderleft.Position.Z;

        var shoulderright = skeleton.Joints[JointType.ShoulderLeft]; //
            L instead of R

        shoulderrightx = shoulderright.Position.X;
        shoulderrighty = shoulderright.Position.Y;
        shoulderrightz = shoulderright.Position.Z;

        var spine = skeleton.Joints[JointType.Spine];

        spinex = spine.Position.X;
        spiney = spine.Position.Y;
        spinez = spine.Position.Z;
```

```
var hipcenter = skeleton.Joints[JointType.HipCenter];

hipcenterx = hipcenter.Position.X;
hipcentery = hipcenter.Position.Y;
hipcenterz = hipcenter.Position.Z;

var hipleft = skeleton.Joints[JointType.HipRight]; // R instead
    of L

hipleftx = hipleft.Position.X;
hiplefty = hipleft.Position.Y;
hipleftz = hipleft.Position.Z;

var hipright = skeleton.Joints[JointType.KneeLeft]; // L instead
     of R

hiprightx = hipright.Position.X;
hiprighty = hipright.Position.Y;
hiprightz = hipright.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Left Arm
var elbowleft = skeleton.Joints[JointType.ElbowRight];

elbowleftx = elbowleft.Position.X;
elbowlefty = elbowleft.Position.Y;
elbowleftz = elbowleft.Position.Z;

var wristleft = skeleton.Joints[JointType.WristRight];

wristleftx = wristleft.Position.X;
wristlefty = wristleft.Position.Y;
wristleftz = wristleft.Position.Z;

var handleft = skeleton.Joints[JointType.HipRight];

handleftx = handleft.Position.X;
handlefty = handleft.Position.Y;
handleftz = handleft.Position.Z;

//////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////
// Right Arm
var elbowright = skeleton.Joints[JointType.ElbowLeft];

elbowrightx = elbowright.Position.X;
elbowrighty = elbowright.Position.Y;
elbowrightz = elbowright.Position.Z;

var wristright = skeleton.Joints[JointType.WristLeft];

wristrightx = wristright.Position.X;
wristrighty = wristright.Position.Y;
wristrightz = wristright.Position.Z;
```

```
    var handright = skeleton.Joints[JointType.HandLeft];

    handrightx = handright.Position.X;
    handrighty = handright.Position.Y;
    handrightz = handright.Position.Z;

    ////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////
    // Left Leg
    var kneeleft = skeleton.Joints[JointType.KneeRight];

    kneeleftx = kneeleft.Position.X;
    kneelefty = kneeleft.Position.Y;
    kneeleftz = kneeleft.Position.Z;

    var ankleleft = skeleton.Joints[JointType.AnkleRight];

    ankleleftx = ankleleft.Position.X;
    ankleleftz = ankleleft.Position.Z;

    var footleft = skeleton.Joints[JointType.FootRight];

    footleftx = footleft.Position.X;
    footlefty = footleft.Position.Y;
    footleftz = footleft.Position.Z;

    ////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////
    // Right Leg
    var kneeright = skeleton.Joints[JointType.KneeLeft];

    kneerightx = kneeright.Position.X;
    kneerighty = kneeright.Position.Y;
    kneerightz = kneeright.Position.Z;

    var ankleright = skeleton.Joints[JointType.AnkleLeft];

    anklerightx = ankleright.Position.X;
    anklerighty = ankleright.Position.Y;
    anklerightz = ankleright.Position.Z;

    var footright = skeleton.Joints[JointType.FootLeft];

    footrightx = footright.Position.X;
    footrighty = footright.Position.Y;
    footrightz = footright.Position.Z;
}

string evalHombros;
double hhd = (Math.Round(shoulderrighty, 3) - Math.Round(footrighty,
     3)) * 100;
double hhi = (Math.Round(shoulderlefty, 3) - Math.Round(footlefty,
    3)) * 100;
```

```
if (Math.Round(shoulderrighty,3) > Math.Round(shoulderlefty,3))
{
    double deltaShoulders = (Math.Round(shoulderrighty, 3) - Math.
        Round(shoulderlefty, 3)) * 100;
    evalHombros = "Hombro DERECHO  " + String.Format("{0, 1:N3}",
        deltaShoulders) + " cm por encima del colateral";
}
else if (Math.Round(shoulderrighty,3) < Math.Round(shoulderlefty,3))
{
    double deltaShoulders = (Math.Round(shoulderlefty, 3) - Math.
        Round(shoulderrighty, 3)) * 100;
    evalHombros = "Hombro IZQUIERDO  " + String.Format("{0, 1:N3}",
        deltaShoulders) + " cm por encima del colateral";
}
else
{
    evalHombros = "Hombros al mismo nivel";
}


string evalHip;
double hcd = (Math.Round(hiprighty, 3) - Math.Round(footrighty, 3))
    * 100;
double hci = (Math.Round(hiplefty, 3) - Math.Round(footlefty, 3)) *
    100;

if (Math.Round(hiprighty, 3) > Math.Round(hiplefty, 3))
{
    double deltaHips = (Math.Round(hiprighty, 3) - Math.Round(
        hiplefty, 3)) * 100;
    evalHip = "Cadera DERECHA  " + String.Format("{0, 1:N3}",
        deltaHips) + " cm por encima de la colateral";
}
else if (Math.Round(hiprighty, 3) < Math.Round(hiplefty, 3))
{
    double deltaHips = (Math.Round(hiplefty, 3) - Math.Round(
        hiprighty, 3)) * 100;
    evalHip = "Cadera IZQUIERDA  " + String.Format("{0, 1:N3}",
        deltaHips) + " cm por encima de la colateral";
}
else
{
    evalHip = "Caderas al mismo nivel";
}


string evalKnee;

double hrd = (Math.Round(kneerighty, 3) - Math.Round(footrighty, 3))
     * 100;
double hri = (Math.Round(kneelefty, 3) - Math.Round(footlefty, 3)) *
     100;

if (Math.Round(kneerighty, 3) > Math.Round(kneelefty, 3))
{
    double deltaKnees = (Math.Round(kneerighty, 3) - Math.Round(
        kneelefty, 3)) * 100;
    evalKnee = "Rodilla DERECHA  " + String.Format("{0, 1:N3}",
        deltaKnees) + " cm por encima de la colateral";
```

```
}
else if (Math.Round(kneerighty, 3) < Math.Round(kneelefty, 3))
{
    double deltaKnees = (Math.Round(kneelefty, 3) - Math.Round(
        kneerighty, 3)) * 100;
    evalKnee = "Rodilla IZQUIERDA  " + String.Format("{0, 1:N3}",
        deltaKnees) + " cm por encima de la colateral";
}
else
{
    evalKnee = "Rodillas al mismo nivel";
}


string evalAnkle;

if (Math.Round(anklerighty, 3) > Math.Round(anklelefty, 3))
{
    double deltaAnkles = (Math.Round(anklerighty, 3) - Math.Round(
        anklelefty, 3)) * 100;
    evalAnkle = "Tobillo DERECHO arriba por " + String.Format("{0,
        1:N3}", deltaAnkles) + " cm por encima del colateral";
}
else if (Math.Round(anklerighty, 3) < Math.Round(anklelefty, 3))
{
    double deltaAnkles = (Math.Round(anklelefty, 3) - Math.Round(
        anklerighty, 3)) * 100;
    evalAnkle = "Tobillo IZQUIERDO " + String.Format("{0, 1:N3}",
        deltaAnkles) + " cm por encima del colateral";
}
else
{
    evalAnkle = "Tobillos al mismo nivel";
}


//Desvalance hombros
double dishombrorx = (Math.Round(shoulderrightx, 3) - Math.Round(
    shouldercenterx, 3));
double dishombrolx = (Math.Round(shoulderleftx, 3) - Math.Round(
    shouldercenterx, 3));
double dishombrory = (Math.Round(shoulderrighty, 3) - Math.Round(
    shouldercentery, 3));
double dishombroly = (Math.Round(shoulderlefty, 3) - Math.Round(
    shouldercentery, 3));

double tetashoulderright = Math.Atan(dishombrory / dishombrorx);
double tetashoulderleft = Math.Atan(dishombroly / dishombrolx);

double tetashoulder = (3.1416 + tetashoulderleft - tetashoulderright
    ) * 180 / 3.1416;

//Desvalance rodillas (derecha)
double disrodilladabx = (Math.Round(anklerightx, 3) - Math.Round(
    kneerightx, 3));
double disrodilladarx = (Math.Round(hiprightx, 3) - Math.Round(
    kneerightx, 3));
double disrodilladaby = (Math.Round(anklerighty, 3) - Math.Round(
    kneerighty, 3));
```

```
double disrodilladary = (Math.Round(hiprighty, 3) - Math.Round(
    kneerighty, 3));

double tetakneerightup = Math.Atan(disrodilladarx / disrodilladary);
double tetakneerightdown = Math.Atan(disrodilladabx / disrodilladaby
    );

double tetakneeright = (3.1416 - tetakneerightup + tetakneerightdown
    ) * 180 / 3.1416;

string estadorodillad="";

if ((15 <= tetakneeright && tetakneeright <= 165) )
{
    estadorodillad = "Rodilla derecha en valgo critico";
    rodilladerecha = 1;
    //semaforo
    rojo1.Fill = Brushes.Red;
    amarillo1.Fill = Brushes.White;
    verde1.Fill = Brushes.White;
}
if ((15 >= tetakneeright && tetakneeright >= 8)||(165 <=
    tetakneeright && tetakneeright <= 172))
{
    estadorodillad = "Rodilla derecha en valgo ligero";
    rodilladerecha = 1;
    //semaforo
    rojo1.Fill = Brushes.White;
    amarillo1.Fill = Brushes.Yellow;
    verde1.Fill = Brushes.WhiteSmoke;
}
if ((195 <= tetakneeright && tetakneeright <= 345))
{
    estadorodillad = "Rodilla derecha en varo critico";
    rodilladerecha = 1;
    //semaforo
    rojo1.Fill = Brushes.Red;
    amarillo1.Fill = Brushes.White;
    verde1.Fill = Brushes.White;
}
if ((195 >= tetakneeright && tetakneeright >= 188) || (352 >=
    tetakneeright && tetakneeright >= 345))
{
    estadorodillad = "Rodilla derecha en varo ligero";
    rodilladerecha = 1;
    //semaforo
    rojo1.Fill = Brushes.White;
    amarillo1.Fill = Brushes.Yellow;
    verde1.Fill = Brushes.White;
}
if ((188 >= tetakneeright && tetakneeright >= 172) || (8 >=
    tetakneeright && tetakneeright >= 352))
{
    estadorodillad = "";
    rodilladerecha = 0;
    //semaforo
    rojo1.Fill = Brushes.White;
```

```
    amarillo1.Fill = Brushes.White;
    verde1.Fill = Brushes.Green;
}
//Desvalance rodillas (izquierda)
double disrodillaiabx = (Math.Round(ankleleftx, 3) - Math.Round(
    kneeleftx, 3));
double disrodillaiarx = (Math.Round(hipleftx, 3) - Math.Round(
    kneeleftx, 3));
double disrodillaiaby = (Math.Round(anklelefty, 3) - Math.Round(
    kneelefty, 3));
double disrodillaiary = (Math.Round(hiplefty, 3) - Math.Round(
    kneelefty, 3));

double tetakneeleftup = Math.Atan(disrodillaiarx / disrodillaiary);
double tetakneeleftdown = Math.Atan(disrodillaiabx / disrodillaiaby)
    ;

double tetakneeleft = (3.1416 - tetakneeleftup + tetakneeleftdown) *
     180 / 3.1416;

string estadorodillai= "";

if ((15 <= tetakneeleft && tetakneeleft <= 165) )
{
    estadorodillai = "Rodilla izquierda en varo critico";
    rodillaizquierda = 1;
    //semaforo
    verde.Fill = Brushes.White;
    amarillo.Fill = Brushes.White;

    rojo.Fill = Brushes.Red;
}
if ((15 >= tetakneeleft && tetakneeleft >= 8) || (172 >=
    tetakneeleft && tetakneeleft >=165))
{
    estadorodillai = "Rodilla izquierda en varo ligero";
    rodillaizquierda = 1;
    //semaforo
    rojo.Fill = Brushes.White;
    verde.Fill = Brushes.White;
    amarillo.Fill = Brushes.Yellow;
}
if ((195 <= tetakneeleft && tetakneeleft <= 345))
{
    estadorodillai = "Rodilla izquierda en valgo critico";
    rodillaizquierda = 1;
    //semaforo
    verde.Fill = Brushes.White;
    amarillo.Fill = Brushes.White;

    rojo.Fill = Brushes.Red;
}
if ((188 <= tetakneeleft && tetakneeleft <= 195) || (345 <=
    tetakneeleft && tetakneeleft <= 352))
{
    estadorodillai = "Rodilla izquierda en valgo ligero";
    rodillaizquierda = 1;
```

```
    //semaforo
    rojo.Fill = Brushes.White;
    verde.Fill = Brushes.White;
    amarillo.Fill = Brushes.Yellow;

}
if ((188 >= tetakneeleft && tetakneeleft >= 172) || (8 >=
    tetakneeleft && tetakneeleft >= 352))
{
    estadorodillai = "";
    rodillaizquierda = 0;
    //semaforo
    rojo.Fill = Brushes.White;
    amarillo.Fill = Brushes.White;
    verde.Fill = Brushes.Green;
}
//Desvalanve tovillos (derecho)
double distobillodabx = (Math.Round(footrightx, 3) - Math.Round(
    anklerightx, 3));
double distobillodarx = (Math.Round(kneerightx, 3) - Math.Round(
    anklerightx, 3));
double distobillodaby = (Math.Round(footrighty, 3) - Math.Round(
    anklerighty, 3));
double distobillodary = (Math.Round(kneerighty, 3) - Math.Round(
    anklerighty, 3));

double tetaanklerightup = Math.Atan(distobillodarx / distobillodary)
    ;
double tetaanklerightdown = Math.Atan(distobillodabx /
    distobillodaby);

double tetaankleright = (3.1416 - tetaanklerightup +
    tetaanklerightdown) * 180 / 3.1416;

//Desvalance tovillos (izquierdo)
double distobilloiabx = (Math.Round(footleftx, 3) - Math.Round(
    ankleleftx, 3));
double distobilloiarx = (Math.Round(kneeleftx, 3) - Math.Round(
    ankleleftx, 3));
double distobilloiaby = (Math.Round(footlefty, 3) - Math.Round(
    anklelefty, 3));
double distobilloiary = (Math.Round(kneelefty, 3) - Math.Round(
    anklelefty, 3));

double tetaankleleftup = Math.Atan(distobilloiarx / distobilloiary);
double tetaankleleftdown = Math.Atan(distobilloiabx / distobilloiaby
    );

double tetaankleleft = (3.1416 - tetaankleleftup + tetaankleleftdown
    ) * 180 / 3.1416;

//prueva de distancia.

double disbrazod = Math.Sqrt((Math.Round(wristrightx, 3) - Math.
    Round(elbowrightx, 3)) * (Math.Round(wristrightx, 3) - Math.Round
    (elbowrightx, 3)) +
                    (Math.Round(wristrighty, 3) - Math.Round(
```

```
                            elbowrighty, 3)) * (Math.Round(wristrighty,
                            3) - Math.Round(elbowrighty, 3)) +
                        (Math.Round(wristrightz, 3) - Math.Round(
                            elbowrightz, 3)) * (Math.Round(wristrightz,
                            3) - Math.Round(elbowrightz, 3)))
        ;


    textBox3.Text = ("OBSERVACIONES: " + "\n\n" +

                    "Prueba " + string.Format("{0, 1:N2}", disbrazod) +"
                        cm\n"+
                      "HOMBROS " + "\n" +
                          evalHombros +"\n"+ "Hombro derecho a "+string.
                              Format("{0, 1:N2}", hhd)+" Centimetros sobre
                               el piso"+
                          "\n" + "Hombro izquierdo a " + string.Format("
                              {0, 1:N2}", hhi) + " Centimetros sobre el
                              piso" +

                      "\n\n CADERA " + "\n" +
                         evalHip + "\n" +
                         "\n" + "Cadera derecha a " + string.Format("{0,
                             1:N2}", hcd) + " Centimetros sobre el piso" +
                          "\n" + "Cadera izquierda a " + string.Format("
                              {0, 1:N2}", hci) + " Centimetros sobre el
                              piso" +

                      "\n\n RODILLAS " + "\n" +
                          evalKnee +
                          "\n" + "Angulo rodilla derecha" + string.Format
                              ("{0, 1:N2}", tetakneeright) +" Grados"+ "\n
                               Angulo rodilla izquierda " + string.Format(
                              "{0, 1:N2}", tetakneeleft) +" Grados"+ "\n\n
                              " +
                           estadorodillai+"\n"+ estadorodillad +"\n"
                            + "Rodilla derecha a " + string.Format("{0,
                                1:N2}", hrd) + " Centimetros sobre el piso
                                " +
                          "\n" + "Rodilla izquierda a " + string.Format("
                              {0, 1:N2}", hri) + " Centimetros sobre el
                              piso"
                    /* "TOBILLOS " + "\n" +
                        evalAnkle  +
                       "\n" + "Angulo tobillo derecho" + string.Format
                           ("{0, 1:N2}", tetaankleright) +" Grados"+ "\n
                           Angulo tobillo izquierdo" + string.Format("{0,
                            1:N2}", tetaankleleft) +" Grados"+ "\n\n"
                    */
                     );
}

private void RadioButton_Checked(object sender, RoutedEventArgs e)
{
    // Add code to perform some action here.
}
```

```
private void MenuItem_Click_1(object sender, RoutedEventArgs e)
{
    iniciar.Visibility = Visibility.Hidden;

    //this.sensor.Stop();
    textBlock2.Text="";
    textBlock1.Text="";
    textBox2.Text="";
    textBox1.Text="";
    textBox4.Text="";
    textBox5.Text="";
    textBox3.Text="";
    exp.Text = "";
    actividad.Text = "";
    eda.Text = "";
    anota.Text = "";

    textBox3.IsReadOnly = true;

    l1.Visibility = Visibility.Hidden;
    l2.Visibility = Visibility.Hidden;
    l3.Visibility = Visibility.Hidden;
    l4.Visibility = Visibility.Hidden;
    l5.Visibility = Visibility.Hidden;
    l6.Visibility = Visibility.Hidden;
    l7.Visibility = Visibility.Hidden;
    l8.Visibility = Visibility.Hidden;
    l9.Visibility = Visibility.Hidden;
    l10.Visibility = Visibility.Hidden;
    l11.Visibility = Visibility.Hidden;
    l12.Visibility = Visibility.Hidden;
    l13.Visibility = Visibility.Hidden;
    l14.Visibility = Visibility.Hidden;
    l15.Visibility = Visibility.Hidden;
    l16.Visibility = Visibility.Hidden;
    l18.Visibility = Visibility.Hidden;
    l19.Visibility = Visibility.Hidden;
    l20.Visibility = Visibility.Hidden;
    l21.Visibility = Visibility.Hidden;
    l17.Visibility = Visibility.Hidden;
}
private void MenuItem_Click_2(object sender, RoutedEventArgs e)
{

    if (textBox1.Text != "" && textBox2.Text != "" && textBox4.Text != "
        " && textBox5.Text != "")
    {
        if (textBox3.Text != "")
        {


            if (null == this.sensor)
            {
                this.statusBarText.Text = Properties.Resources.
                    ConnectDeviceFirst;
                return;
            }
```

```
// create a png bitmap encoder which knows how to save a .
    png file
BitmapEncoder encoder = new PngBitmapEncoder();



// create frame from the writable bitmap and add to encoder
encoder.Frames.Add(BitmapFrame.Create(this.colorBitmap));
//encoder.Frames.Add(BitmapFrame.Create(this.DrawBone)); //
    Prueba para guardar im gen de skelleton

//string time = System.DateTime.Now.ToString("hh'-'mm'-'ss",
     CultureInfo.CurrentUICulture.DateTimeFormat);
//string time = System.DateTime.Now.ToString("yyyy MM d HH '
    hr' mm 'min' ss 'seg'", CultureInfo.CurrentUICulture.
    DateTimeFormat);
string time = System.DateTime.Now.ToString("yyyy'-'MM'-'d HH
    '.'mm'.'ss", CultureInfo.CurrentUICulture.DateTimeFormat)
    ;


System.Drawing.Bitmap bmp = new System.Drawing.Bitmap(1, 1);
     // Mapa de bits auxiliar 1
System.Drawing.Bitmap aux = new System.Drawing.Bitmap(640,
    480); // Mapa de bits auxiliar 2

string myPhotos = Environment.GetFolderPath(Environment.
    SpecialFolder.MyPictures);

string path = Path.Combine(myPhotos, time + " Color.png");

// write the new file to disk
try
{
    using (FileStream fs = new FileStream(path, FileMode.
        Create))
    {
        encoder.Save(fs);
    }

    this.statusBarText.Text = string.Format("{0} {1}",
        Properties.Resources.ScreenshotWriteSuccess, path);
}
catch (IOException)
{
    this.statusBarText.Text = string.Format("{0} {1}",
        Properties.Resources.ScreenshotWriteFailed, path);
}

string[] lines = { textBlock2.Text, textBlock1.Text, "\n",
                "Fecha y hora de prueba: " + time, "
                    Operador: " + textBox2.Text, "\n",
                "Paciente: " + textBox1.Text,"Numero de
                    expediente"+ exp.Text,"Actividad lavoral
                    "+ actividad.Text,
                "Edad"+ eda.Text,"Peso: " + textBox4.Text +
```

```
                    " kg", "Estatura: " + textBox5.Text + "
                    m", "\n",
                textBox3.Text, "\n",
                "COORDENADAS", coordinates };

ScreenCapturer.CaptureAndSave("C:\\UAQKinectPostural\\
    Imagenes\\" + exp.Text + " Interfaz.png", CaptureMode.
    Window, System.Drawing.Imaging.ImageFormat.Png);
System.IO.File.WriteAllLines("C:\\UAQKinectPostural\\" + exp
    .Text + ".txt", lines);

bmp = ScreenCapturer.Capture(CaptureMode.Window);
System.Drawing.Rectangle cropArea = new System.Drawing.
    Rectangle(50, 50, 620, 500);
aux = bmp.Clone(cropArea, bmp.PixelFormat);
aux.Save("C:\\UAQKinectPostural\\Imagenes\\" + exp.Text + "
    Test.png");

//Crear archvo excel

Microsoft.Office.Interop.Excel.Application app = new
    Microsoft.Office.Interop.Excel.Application();
app.Visible = true;
app.WindowState = XlWindowState.xlMaximized;

Workbook wb = app.Workbooks.Add(XlWBATemplate.
    xlWBATWorksheet);
Worksheet ws = wb.Worksheets[1];
DateTime currentDate = DateTime.Now;

// OJO Lee la imagen de un archivo en la unidad C:
ws.Shapes.AddPicture(@"C:\\UAQKinectPostural\\Imagenes\\" +
    exp.Text + " Test.png", Microsoft.Office.Core.MsoTriState
    .msoFalse, Microsoft.Office.Core.MsoTriState.msoCTrue, 0,
     120, 300, 256); // posx, posy, ancho y alto



ws.Range["B1"].Value = "Fecha y hora de prueba";
ws.Range["C1"].Value = time;

ws.Range["B2"].Value = "Operador";
ws.Range["C2"].Value = textBox2.Text;

ws.Range["B3"].Value = "Pacinte";
ws.Range["C3"].Value = textBox1.Text;

ws.Range["B4"].Value = "Expediente";
ws.Range["C4"].Value = exp.Text;

ws.Range["B5"].Value = "Actividad";
ws.Range["C5"].Value = actividad.Text;

ws.Range["B6"].Value = "Edad";
ws.Range["C6"].Value = eda.Text+" a os";

ws.Range["B7"].Value = "Peso";
```

```
            ws.Range["C7"].Value = textBox4.Text+"kg";

            ws.Range["B8"].Value = "Estatura";
            ws.Range["C8"].Value = textBox5.Text + "m";


            ws.Range["A26"].Value = textBox3.Text;
            ws.Range["A27"].Value = anota.Text;



            wb.SaveAs("C:\\UAQKinectPostural\\ " + exp.Text+"--" +time +
                 ".xlsx");

            MessageBox.Show("Reporte guardado");
        }
        else
        {
            MessageBox.Show("El sensor no ha realizado ninguna lectura")
                ;
        }
    }
    else
    {

        MessageBox.Show("Es necesario ingresar todos los datos");
    }
}


private void MenuItem_Click_3(object sender, RoutedEventArgs e)
{
    Close();
}
private void MenuItem_Click_4(object sender, RoutedEventArgs e)
{

    if (this.sensor != null && !this.sensor.IsRunning)
    {
        this.sensor.Start();
    }
}
private void MenuItem_Click_5(object sender, RoutedEventArgs e)
{
    if (this.sensor != null && this.sensor.IsRunning)
    {
        this.sensor.Stop();
    }
}

private void iniciar_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Selecchine archivo nuevo");
}

private void MenuItem_Click_6(object sender, RoutedEventArgs e)
{
```

```
        l1.Visibility = Visibility.Visible;
        l2.Visibility = Visibility.Visible;
        l3.Visibility = Visibility.Visible;
        l4.Visibility = Visibility.Visible;
        l5.Visibility = Visibility.Visible;
        l6.Visibility = Visibility.Visible;
        l7.Visibility = Visibility.Visible;
        l8.Visibility = Visibility.Visible;
        l9.Visibility = Visibility.Visible;
        l10.Visibility = Visibility.Visible;
        l11.Visibility = Visibility.Visible;
        l12.Visibility = Visibility.Visible;
        l13.Visibility = Visibility.Visible;
        l14.Visibility = Visibility.Visible;
        l15.Visibility = Visibility.Visible;
        l16.Visibility = Visibility.Visible;
        l17.Visibility = Visibility.Visible;
        l18.Visibility = Visibility.Visible;
        l19.Visibility = Visibility.Visible;
        l20.Visibility = Visibility.Visible;
        l21.Visibility = Visibility.Visible;

        MessageBox.Show("Grid activado ");

    }
    private void MenuItem_Click_7(object sender, RoutedEventArgs e)
    {


        l1.Visibility = Visibility.Hidden;
        l2.Visibility = Visibility.Hidden;
        l3.Visibility = Visibility.Hidden;
        l4.Visibility = Visibility.Hidden;
        l5.Visibility = Visibility.Hidden;
        l6.Visibility = Visibility.Hidden;
        l7.Visibility = Visibility.Hidden;
        l8.Visibility = Visibility.Hidden;
        l9.Visibility = Visibility.Hidden;
        l10.Visibility = Visibility.Hidden;
        l11.Visibility = Visibility.Hidden;
        l12.Visibility = Visibility.Hidden;
        l13.Visibility = Visibility.Hidden;
        l14.Visibility = Visibility.Hidden;
        l15.Visibility = Visibility.Hidden;
        l16.Visibility = Visibility.Hidden;
        l17.Visibility = Visibility.Hidden;
        l18.Visibility = Visibility.Hidden;
        l19.Visibility = Visibility.Hidden;
        l20.Visibility = Visibility.Hidden;
        l21.Visibility = Visibility.Hidden;
        MessageBox.Show("Grid desactivado ");

    }

    private void MenuItem_Click_8(object sender, RoutedEventArgs e)
    {
        Process.Start("C:\\UAQKinectPostural\\manual.pdf");
```

```
        }
        private void MenuItem_Click_9(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Correo: djaramilllo5@gmail.com \n Telefono:
                4271201846");
        }


    }
}
```

3 ScreenCapturer.cs

```
 using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace KinectUAQv02
{

    public enum CaptureMode
    {
        Screen, Window
    }

    public static class ScreenCapturer
    {
        [DllImport("user32.dll")]
        private static extern IntPtr GetForegroundWindow();

        [DllImport("user32.dll")]
        private static extern IntPtr GetWindowRect(IntPtr hWnd, ref Rect rect);

        [StructLayout(LayoutKind.Sequential)]
        private struct Rect
        {
            public int Left;
            public int Top;
            public int Right;
            public int Bottom;
        }

        [DllImport("user32.dll", CharSet = CharSet.Auto, ExactSpelling = true)]
        public static extern IntPtr GetDesktopWindow();

        /// <summary> Capture Active Window, Desktop, Window or Control by hWnd
            or .NET Contro/Form and save it to a specified file.  </summary>
        /// <param name="filename">Filename.
        /// <para>* If extension is omitted, it's calculated from the type of
            file</para>
        /// <para>* If path is omitted, defaults to %TEMP%</para>
        /// <para>* Use %NOW% to put a timestamp in the filename</para></param>
        /// <param name="mode">Optional. The default value is CaptureMode.Window
            .</param>
        /// <param name="format">Optional file save mode.  Default is PNG</param
            >
        public static void CaptureAndSave(string filename, CaptureMode mode =
            CaptureMode.Window, ImageFormat format = null)
        {
            ImageSave(filename, format, Capture(mode));
        }

        /// <summary> Capture a specific window (or control) and save it to a
            specified file.  </summary>
        /// <param name="filename">Filename.
```

```
/// <para>* If extension is omitted, it's calculated from the type of
    file</para>
/// <para>* If path is omitted, defaults to %TEMP%</para>
/// <para>* Use %NOW% to put a timestamp in the filename</para></param>
/// <param name="handle">hWnd (handle) of the window to capture</param>
/// <param name="format">Optional file save mode.  Default is PNG</param
    >
public static void CaptureAndSave(string filename, IntPtr handle,
    ImageFormat format = null)
{
    ImageSave(filename, format, Capture(handle));
}

/// <summary> Capture a specific window (or control) and save it to a
    specified file.  </summary>
/// <param name="filename">Filename.
/// <para>* If extension is omitted, it's calculated from the type of
    file</para>
/// <para>* If path is omitted, defaults to %TEMP%</para>
/// <para>* Use %NOW% to put a timestamp in the filename</para></param>
/// <param name="c">Object to capture</param>
/// <param name="format">Optional file save mode.  Default is PNG</param
    >
public static void CaptureAndSave(string filename, Control c,
    ImageFormat format = null)
{
    ImageSave(filename, format, Capture(c));
}
/// <summary> Capture the active window (default) or the desktop and
    return it as a bitmap </summary>
/// <param name="mode">Optional. The default value is CaptureMode.Window
    .</param>
public static Bitmap Capture(CaptureMode mode = CaptureMode.Window)
{
    return Capture(mode == CaptureMode.Screen ? GetDesktopWindow() :
        GetForegroundWindow());
}

/// <summary> Capture a .NET Control, Form, UserControl, etc. </summary>
/// <param name="c">Object to capture</param>
/// <returns> Bitmap of control's area </returns>
public static Bitmap Capture(Control c)
{
    return Capture(c.Handle);
}


/// <summary> Capture a specific window and return it as a bitmap </
    summary>
/// <param name="handle">hWnd (handle) of the window to capture</param>
public static Bitmap Capture(IntPtr handle)
{
    Rectangle bounds;
    var rect = new Rect();
    GetWindowRect(handle, ref rect);
    bounds = new Rectangle(rect.Left, rect.Top, rect.Right - rect.Left,
        rect.Bottom - rect.Top);
```

```
            CursorPosition = new Point(Cursor.Position.X - rect.Left, Cursor.
                Position.Y - rect.Top);

            var result = new Bitmap(bounds.Width, bounds.Height);
            using (var g = Graphics.FromImage(result))
                g.CopyFromScreen(new Point(bounds.Left, bounds.Top), Point.Empty
                    , bounds.Size);

            return result;
        }

        /// <summary> Position of the cursor relative to the start of the
            capture </summary>
        public static Point CursorPosition;


        /// <summary> Save an image to a specific file </summary>
        /// <param name="filename">Filename.
        /// <para>* If extension is omitted, it's calculated from the type of
            file</para>
        /// <para>* If path is omitted, defaults to %TEMP%</para>
        /// <para>* Use %NOW% to put a timestamp in the filename</para></param>
        /// <param name="format">Optional file save mode.  Default is PNG</param
            >
        /// <param name="image">Image to save.  Usually a BitMap, but can be any
        /// Image.</param>
        static void ImageSave(string filename, ImageFormat format, Image image)
        {
          format = format ?? ImageFormat.Png;
          if (!filename.Contains("."))
            filename = filename.Trim() + "." + format.ToString().ToLower();

          if (!filename.Contains(@"\"))
            filename = Path.Combine(Environment.GetEnvironmentVariable("TEMP")
                ?? @"C:\Temp", filename);

          filename = filename.Replace("%NOW%", DateTime.Now.ToString("yyyy-MM-
              dd@hh.mm.ss"));
          image.Save(filename, format);
        }
    }
}
```