UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO – CAMPUS SUR

CARRERA DE INGENIERÍA ELECTRÓNICA

MENCIÓN SISTEMAS INDUSTRIALES

ANÁLISIS Y ESTUDIO DE LOS CÓDIGOS FUENTE SDK (KIT DE DESARROLLO DE SOFTWARE) E IMPLEMENTACIÓN DE UNA APLICACIÓN DEMOSTRATIVA QUE REGISTRE LA CAPTACIÓN DE MOVIMIENTOS DE MANOS Y BRAZOS DEL CUERPO HUMANO A TRAVÉS DE LED'S INDICADORES MEDIANTE LA UTILIZACIÓN DEL SENSOR KINECT DEL XBOX 360

TESIS PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO ELECTRÓNICO

LUIS ALEJANDRO HERNÁNDEZ TOALA

JUAN DAVID HERRERA RODRÍGUEZ

DIRECTOR: M.Sc. VINICIO TAPIA

Quito, Enero del 2013

DECLARACIÓN

Yo, Luis Alejandro Hernández Toala, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional, y, que he consultado las referencias bibliográficas que se incluyen en este documento. a través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la universidad politécnica salesiana, según lo establecido por la ley de propiedad intelectual por su reglamento y por la normatividad institucional vigente.

_____

Luis Alejandro Hernández Toala

DECLARACIÓN

Yo, Juan David Herrera Rodríguez, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional, y, que he consultado las referencias bibliográficas que se incluyen en este documento. a través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la universidad politécnica salesiana, según lo establecido por la ley de propiedad intelectual por su reglamento y por la normatividad institucional vigente.

_____

Juan David Herrera Rodríguez

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Luis Alejandro Hernández Toala bajo mi dirección.

_____

M.Sc. Vinicio Tapia
Director de Tesis

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Juan David Herrera Rodríguez bajo mi dirección.

_____

M.Sc. Vinicio Tapia
Director de Tesis

ÍNDICE GENERAL

Índice de Figuras

Índice Tablas

RESUMEN

Kinect cuenta con una cámara RGB, un sensor de profundidad, un sensor CMOS de infrarrojos, un micrófono de múltiples matrices, contiene una luz Led, un acelerómetro de tres ejes , un motor y en especial un procesador especializado PrimeSensor que capta el entorno en 3 dimensiones y esas captaciones traducirlas a una imagen sincronizada .

Para el desarrollo de la aplicación demostrativa se empleo el análisis y estudio de los códigos fuentes SDK de Kinect (Camera Fundamletal, Depth Data, Skeletal Tracking), los mismos que permitieron el diseño y control de una Pinza robótica además de un grupo de Led's indicadores que muestran el ángulo de giro de los servomotores en un rango de 0 a 180 grados.

La Pinza robótica consta con 3 grados de libertad ya que posee 3 servomotores, los mismos que van a permitir controlar los movimientos tanto en forma horizontal cuando el usuario de el barrido horizontal de la mano derecha (eje x), el movimiento vertical, cuando el usuario abra y cierre el brazo derecho (eje y), y la apertura y cierre, cuando el usuario abra o cierre la mano (eje z).

La interfaz y los datos que se adquieren de los distintos movimientos que realice el usuario al mover la Pinza robótica, en la aplicación demostrativa se creó en Visual Studio 2010 en el lenguaje de programación C#.

Los datos son enviados por medio de la comunicación serial de C# a la placa Arduino, la misma que se va a encargar de recibir esos datos y enviarlos a los distintos servomotores y Led's indicadores con los que consta la Pinza robótica.

La placa Arduino consta con conexión USB, lo que facilita conectar a cualquier dispositivo que tenga conexión vía USB, en nuestro caso, es conectada la placa al puerto USB en la Laptop que se creó el proyecto.

CAPITULO 1.

1.1Planteamiento Del Problema

En los tiempos actuales, ha existido una relación dependiente entre usuario (Ser Humano) y maquina (Dispositivos electrónicos), en donde para su manejo siempre ha sido necesario que el usuario realice un contacto físico con el dispositivo para que este funcione, situación que a veces puede resultar muy complicado ya que existe ciertos lugares de difícil acceso para el operador, y más aún si este tuviera alguna discapacidad que le impidiera moverse libremente a cualquier sitio.

Cabe recalcar que hoy en día existen tecnologías que abren una puerta hacia la innovación y la creatividad, las cuales nos permiten una gama de herramientas para desarrollar diversos sistemas complejos de control, y reconocimiento de comandos de voz y gestos, pero que en nuestro País no se está aplicando debido al desconocimiento o falta de acceso a los mismos.

En la Carrera de Ingeniería Electrónica es muy importante estar a la vanguardia de conocimientos tecnológicos y como estos pueden ser aplicado en distintas áreas ya sea, educación, salud, control y entretenimiento, cuyos avances tecnológicos no logran ser compartidos en las aulas de clases, por tal motivo se plantea realizar una aplicación demostrativa, la cual sirva de guía para futuras investigaciones.

1.2 Hipótesis.

Para supervisar y adquirir los datos de los sensores del Kinect, se utiliza el software de programación C#, también parte del paquete Microsoft Visual Studio 2010, en el cual se desarrolla el reconocimiento de los movimientos del cuerpo humano y para la implementación de una interfaz visual.

A través del análisis y estudio de los códigos fuente SDK (KIT DE DESARROLLO DE SOFTWARE) de Kinect se lograra comprender la programación y funcionamiento, de esta manera lograr comunicaciones con otros dispositivos con el USB, e implementar una aplicación demostrativa que capte el movimiento de brazos y manos del cuerpo humano.

1.3 Objetivos.

A continuación se describe el objetivo general y específico que tiene por finalidad el presente proyecto.

1.3.1 Objetivo General.

Analizar y Estudiar los códigos fuente SDK(Kit de Desarrollo de Software) del sensor Kinect del Xbox 360, e implementar una aplicación demostrativa que registre la captación de movimientos de manos y brazos del cuerpo humano a través de Led's indicadores

1.3.2 Objetivos Específico.

- Analizar el lenguaje de programación del SDK del sensor Kinect basado en C#.
- Elaborar un programa basado en los códigos fuentes SDK, en el cual está desarrollado el control del sensor Kinect.
- Realizar una aplicación que detecte el movimiento de brazos y manos del cuerpo humano.

- Implementar un sistema de indicadores Led's a través del puerto USB, que muestre la apertura y cierre de manos, barrido horizontal de la mano, apertura y cierre de brazos del cuerpo humano.

1.4 Justificación.

Aprovechando la disponibilidad y gran avance de las tecnologías de reconocimiento de voz y gestos humanos en tres dimensiones, se proyecta el desarrollo de una aplicación con el sensor Kinect. La interfaz del sensor de Kinect presenta grandes ventajas para el desarrollo de nuestro proyecto ya que reconoce gestos, comandos de voz, objetos e imágenes.

El proyecto es factible debido a que este dispositivo es asequible, presenta tecnología de punta, mismo que puede ser utilizado como herramienta, y ser aplicados en distintas áreas como: Educación, Entretenimiento, Médicas, Comercial, Industrial, permitiendo a los usuarios controlar e interactuar sin la necesidad de tener contacto físico con el dispositivo.

Por todas las características que tiene Kinect es útil analizar el código fuente SDK (Kit de Desarrollo de Software) del sensor y de esta manera poder utilizar toda su tecnología y aplicarla en el campo del control electrónico, como complemento didáctico a la carrera de Ingeniería Electrónica.

Capitulo 2

2.1 Estado Del Arte.

2.1.1 Situación Actual.

Kinect del Xbox 360, es un dispositivo que permite la interacción con el cuerpo humano, mediante una interfaz que capta y reconoce gestos humanos, comandos de voz e imágenes. Debido a sus utilidades y características Kinect es un claro oponente contra otras tecnologías como Wiimote de Nintendo y PlayStation Move de Sony.

FIGURA 1 De izquierda a derecha, PlayStation Move de Sony, Kinect Xbox 360 de Microsoft y Wiimote de Nintendo[1]

El sensor Kinect fue presentada por primera vez en junio del 2009 con el nombre de "Project Natal", pero su verdadero nombre se conoció el 13 de junio del 2010 que fue "Kinect" y en Noviembre del mismo año salió a la venta en EE.UU y México. Para su difusión la empresa Microsoft gastó un aproximado de 500 millones de dólares solo para su conocimiento en EE.UU, dicha suma de dinero fue mucho más grande que la inversión que utilizaron para el lanzamiento del Xbox 360.

[1] FIGURA 1 Tomada de: http://www.facilware.com/reflexion-sobre-las-nuevas-tecnologias-y-los-nuevos-controladores.html

En Diciembre del 2010 la empresa PrimeSense lanzo el primer SDK no oficial para Kinect.

Gracias al SDK no oficial de Kinect, numerosos, desarrolladores, investigadores, estudiantes, instituciones y aficionados, comenzaron la investigación y programación de diversas aplicaciones aprovechando todas las características que brinda Kinect más allá de los videojuegos.

Microsoft al contemplar el rotundo éxito de su dispositivo, desarrolló un SDK oficial y gratuito que es compatible con Windows 7. Actualmente existen tres versiones del SDK y cada una presenta una mejora respecto a la anterior, la primera versión lanzada para las personas interesadas en el desarrollo de aplicaciones con Kinect fue la versión 1.0, posteriormente se obtuvo la versión 1.0 Beta y hoy en día existe la última versión del SDK 1.5 v.

Microsoft ha facilitado los códigos fuente del sensor Kinect que sirven de base para aprender a utilizar las diferentes características del sensor como son:

- Cámara de video RGB (esto es, con componentes rojo, verde y azul).

- Un sensor de profundidad, para registrar la distancia de los objetos

- Un arreglo de micrófonos, que permite localizar la fuente de los sonidos y suprimir ruido de fondo.

En las dos primeras versiones que Microsoft lanzó se proporciono aplicaciones demo para entender y desarrollar nuevas aplicaciones. Entre los más relevantes se puede mencionar al Skeletal Tracking con el cual se reconoce el cuerpo humano y capta los movimiento realizados, Shape Game es un juego para 2 personas que utiliza de base el conocimiento del Skeletal Tracking, Kinect Audio Demo con el cual se reconoce los comando de voz para realizar una determinada acción, (esta opción esta solo disponible en idioma Inglés). En la última versión además de presentar una mejora tanto en la calidad grafica como en la interfaz de usuario, de

las aplicaciones anteriores, incorporó nuevas opciones para el programador en el cual puede incorporar Avatares (personajes virtuales que representan al usuario), el reconocimiento del rostro de personas en 3D, entre otros.

2.1.2 Referencias contemporáneas del proyecto.

TurtleBot.

TurtleBot es un robot asistente ideal para llevar o traer cosas, que fue diseñado gracias a las ventajas que presenta la tecnología del sensor Kinect , es un robot económico , que está equipado con componentes básicos, como un computador portátil, un robot iRobot Create y en especial el sensor 3D Microsoft Kinect

El sensor 3D Microsoft Kinect , le permite al TurtleBot que capta el ambiente circundante y que lo recorra eficazmente , las ordenes se ingresa en el computador portátil , y además tiene un sistema de software abierto , lo cual permite reprogramarle y presentar o diseñar mejores del mismo.

FIGURA 2 Robot Asistente TurtleBot Equipado Con El Sensor Kinect[2]

[2] FIGURA 2 Tomada de: http://www.ciencia-explicada.com/2011/05/turtlebot-el-camarero-robotico-basado.html

Nuevo asistente en el quirófano

Para minimizar las posibilidades de que los pacientes contraigan infecciones en el quirófano, dos investigadores españoles crearon TedCas, una aplicación basada en Kinect, que permite a los médicos que intervienen en la cirugía buscar en el computador toda la información del expediente del paciente, sin necesidad de tocar ningún objeto. La aplicación ganó el premio "únete al movimiento Kinect" y fue uno de los 10 proyectos que apoyó Wayra en España.

FIGURA 3 Consulta De Imágenes Radiológicas Mediante El Sensor Kinect[3]

Espejo virtual para tiendas de ropa

El "Magic Mirror" de Intel es una experiencia de compra virtual que utiliza a un avatar del cliente vestido con las prendas que desea probarse. Una pantalla muestra al avatar en 3D, el cual sigue los movimientos de la persona en tiempo real y cambia las dimensiones del cuerpo usando gestos. Por otro lado, la tienda Bloomingdale está utilizando Kinect, en su almacén de Los Ángeles, para hacer un mapa 3D del cuerpo y así producir prendas de vestir para cada uno de sus clientes.

[3] FIGURA 3 Tomada de: http://rehabilitacionymedicinafisica.blogspot.com/2011_11_01_archive.html

FIGURA 4 Microsoft Presumió Un Probador De Ropa Virtual Para El Sistema Kinect[4]

Estudian movimiento de los glaciares

El equipo requerido para realizar mediciones en 3D a superficies como la de un glaciar puede costar entre 10.000 y 200.000 dólares. Sin embargo, el investigador Marco Tedesco, de la Universidad City College de Nueva York, compró un Kinect de 120 dólares que acopló a un helicóptero a control remoto para tomar Imágenes de las lagunas que se forma sobre los glaciares cuando estos se derriten cuya agua drena por grietas y acelera el camino de estos hacia el mar y así pudo anticipar este fenómeno.

FIGURA 5 Estudio De Los Glaciares Con Kinect[5]

[4] FIGURA 4 Tomada de: http://contenidos.sonoraplaza.com/notas/7986
[5] FIGURA 5 Tomada de: http://www.juegoskinectxbox.com/page/10/

Anuncios inteligentes

Un programa que se desarrolla en los laboratorios de Intel, llamado Advertising Framework, reconoce la edad y sexo de las personas mediante las cámaras y el sensor de Kinect, con el fin de mostrarles de forma inteligente el anuncio publicitario que más se oriente hacia sus intereses. La idea es ubicarlos en lugares como centros comerciales, hoteles, aeropuertos y zonas comerciales, para que la publicidad sea dirigida a quien realmente corresponda.

FIGURA 6 Anuncios Inteligentes Con Kinect[6]

2.2 Sensor Kinect.

2.2.1 Componentes de Hardware.

El sensor cuenta con una cámara RGB, un sensor de profundidad, un sensor CMOS de infrarrojos, un micrófono de múltiples matrices, contiene una luz Led, un acelerómetro de tres ejes , un motor y en especial un procesador especializado PrimeSensor que capta el entorno en 3 dimensiones y esas captaciones traducirlas a una imagen sincronizada .

[6] FIGURA 6 Tomada de: http://pxnonline.wordpress.com/category/publicidad/

FIGURA 7 Componentes Del Sensor Kinect[7]

- **CAMARA RGB** Almacena los datos de los tres canales que envían los sensores en una sola resolución de 1280 x 960 , lo cual hace posible capturar una imagen en color , y envía datos a una frecuencia de actualización de 30 fps (Frame por Segundos)

- **MICRÓFONO DE MULTIPLES MATRICES** Conjunto de cuatro micrófonos que permite la localización de la fuente acústica y la eliminación del ruido ambiente , además permite la posibilidad de grabar audio

- **LUZ LED** Es un indicador para saber que el Kinect está listo para usarse.

- **MOTOR** Permite el movimiento del sensor en sentido vertical.

[7] FIGURA 7 Tomada de: http://blog.robotiq.com/bid/40428/Using-The-Kinect-For-Robotic-Manipulation

- **SENSORES**

- **SENSORES 3D DE PROFUNDIDAD** Es la combinación de un proyector de infrarrojos Laser y un sensor de imagen CMOS (sensor de profundidad).

  El proyector de infrarrojos Láser emite haces de luz infrarroja mientras que el sensor de imagen CMOS o sensor de profundidad se encarga de captar estos haces de luz infrarrojos reflejados hacia el sensor, los haces reflejados son convertidas en información de profundidad midiendo la distancia entre un objeto y el sensor, esto hace que la imagen en profundidad sean posibles.

- **SENSOR DE IMAGEN CMOS PARA RGB** Se emplea para capturar la resolución espacial , es decir captar las coordenadas de los ejes X e Y , se utiliza la entrada RGB (Rojo, Verde y Azul), y así para proporcionar color a las imágenes capturadas por el sensor de profundidad

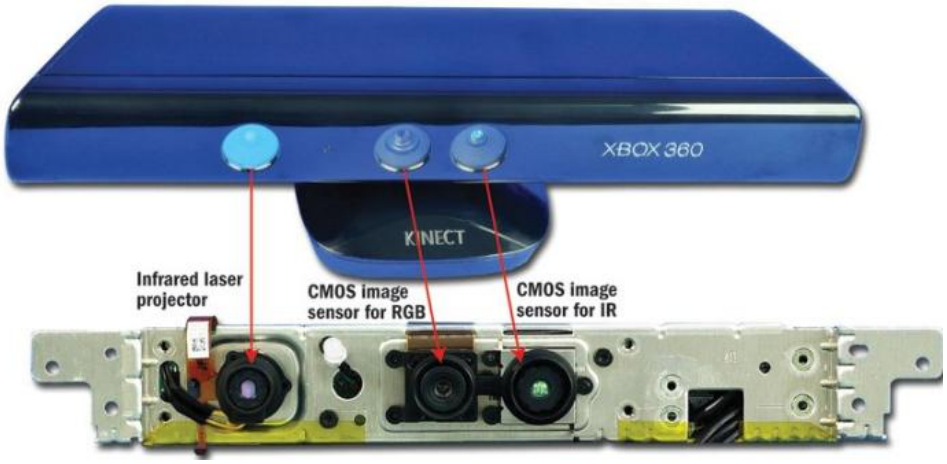


FIGURA 8 Sensores que conforman al Kinect[8]

[8] FIGURA 8 Tomada de: http://blog.robotiq.com/bid/40428/Using-The-Kinect-For-Robotic-Manipulation

- **CHIP PRIMESENSE PS1080**

El Chip PrimeSense 1080, sirve para reconstituir una captura de movimiento 3D de la escena que esté ubicada al frente del Kinect ya que este chip captura su entorno en tres dimensiones y transforma esas capturas a imágenes sincronizadas en 3D.

Este Chip fue desarrollado por la empresa israelita PrimeSense el cual fue su mayor logro ya que permite una infinidad de posibilidades.

Además los dos elementos que conforman el sensor 3D, el proyector de infrarrojos y el sensor de profundidad, trabajan en conjunto con el chip interno PrimeSensor

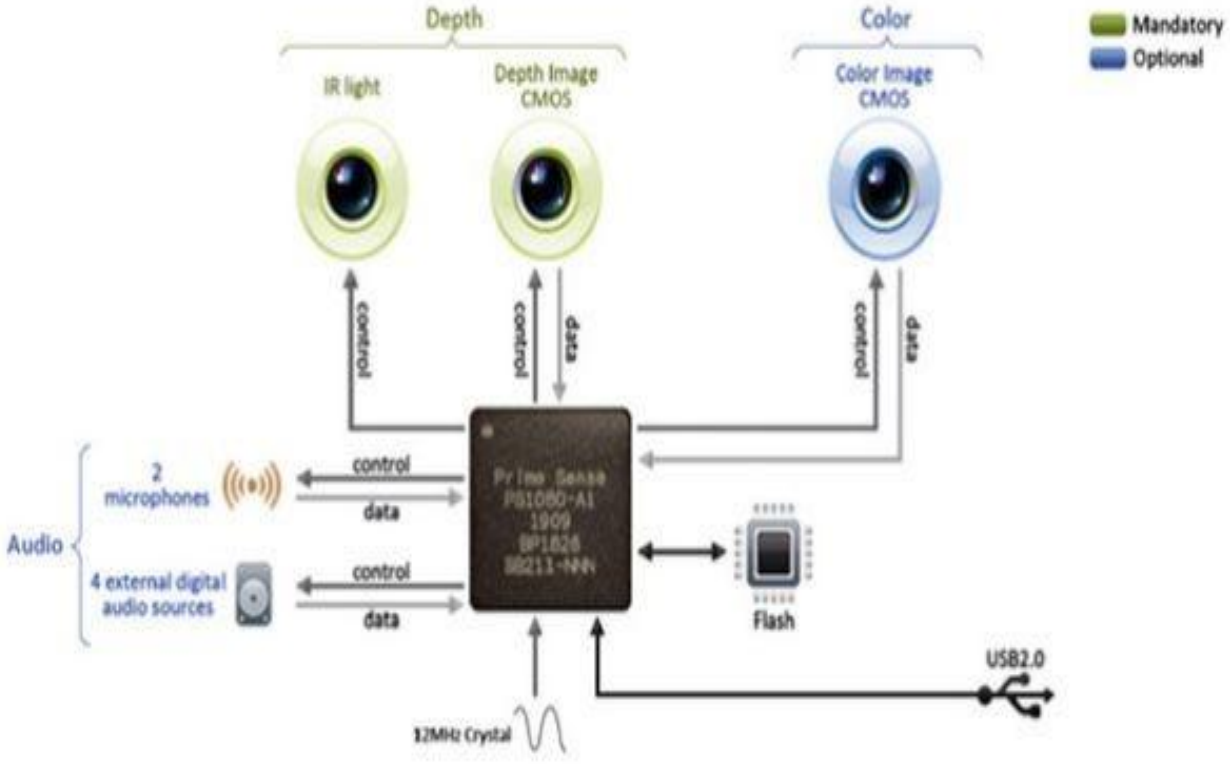


FIGURA 9 Sistema que conforma el chip PrimeSensor[9]

[9] FIGURA 9 Tomada de: http://www.eeworld.com.cn/xfdz/2010/1108/article_3829.html

El sensor Kinect también contiene aunque no estén a simple vista en su estructura de hardware:

- **ACELERÓMETRO** Permite tener estabilidad en las imágenes cuando se mueve el sensor Kinect

- **MEMORIA RAM 512 Mb**

- **VENTILADOR** Para el enfriamiento del dispositivo.

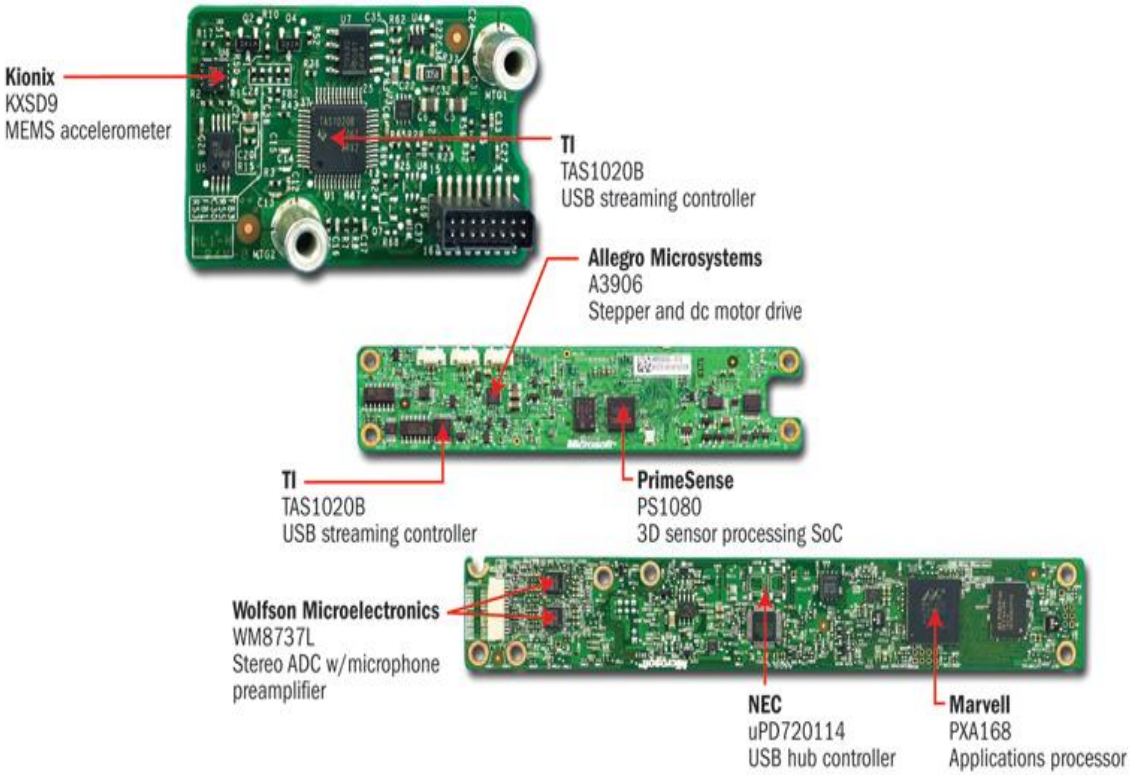


FIGURA 10 Placas principales y circuitos integrados destacados en el Kinect[10]

[10] FIGURA 10 Tomada de: http://www.eetimes.com/design/signal-processing-dsp/4211071/Inside-Xbox-360-s-Kinect-controller.

- **Adaptador de Kinect**

Kinect brinda una gran ventaja para conectarlo a nuestro computador ya que no se necesita de cables o adaptadores extras, solamente el adaptador KINECT el cual contiene un conector para la fuente de corriente, 2 dispositivos USB, uno para el sensor Kinect y otro para ser conectado a una computadora.

FIGURA 11 Adaptador USB de Kinect[11]

2.2.2 Funcionamiento de Kinect.

El funcionamiento del Kinect se divide en las siguientes partes:

- Reconocimiento de imágenes RGB
- Reconocimiento de imágenes en 3D
- Reconocimiento del esqueleto Humano
- Reconocimiento de Audio

[11] FIGURA 11 Tomada de: http://www.eetimes.com/design/signal-processing-dsp/4211071/Inside-Xbox-360-s-Kinect-controller

- Motor

2.2.2.1 Reconocimiento de imágenes RGB.

El sensor Kinect permite el reconocimiento de imágenes en tiempo real de una manera óptima, gracias a que Microsoft ha logrado efectos y funciones que antes tenían un elevado costo y que hoy en día gracias al Kinect se lo puede lograr a precios asequibles.

El reconocimiento de imágenes RGB permite añadir color a las imágenes capturadas por su sensor de 3D, el cual está conformado por el proyector y el sensor de profundidad

2.2.2.2 Reconocimiento de imágenes en 3D.

Por medio del proyector de infrarrojos se emiten unos haces de laser, los cuales rebotan en todo el campo de juego lo que permite que la cámara capte la profundidad de los diferentes objetos.

Al obtener estos datos Kinect emplea una serie de filtros con la finalidad de determinar que es una persona o que no lo es. El sistema emplea patrones para determinar si es una persona como por ejemplo, que una persona tiene extremidades superiores, inferiores, una cabeza, tiene su pelo largo o corto, para poder diferenciarlas de la mesa o de algún otro objeto que pueda existir en el campo de juego.

Esta información es ordenada para poder convertir la identificación de las partes del cuerpo humano en Joint o puntos, que sirven de referencia para que el procesador PrimeSensor grafique el esqueleto humano.

2.2.2.3 Reconocimiento del Esqueleto Humano.

Cuando la información se ha ordenado y se ha reconocido las partes del cuerpo humano, se crea un esqueleto, Kinect tiene por defecto cargadas en su sistema más de 200 posiciones comunes del ser humano, esto es para que Kinect llene los vacíos automáticamente en caso de que una acción tape alguna parte del esqueleto a la cámara, Kinect genera varios esqueletos pero solo se elige uno basándose en la experiencia.

2.2.2.4 Reconocimiento de Audio.

Kinect tiene ubicaciones específicas para los micrófonos que lo componen, uno a la izquierda y tres a la derecha, ya que de esta manera se logra un optimo reconocimiento de voz a la distancia, el ruido es anulado por la unidad de procesamiento y se utiliza su sistema de software para determinar de dónde proviene el sonido y de esta manera crear una especie de burbuja de sonido alrededor del jugador , logrando separar el sonido de la voz y omitir al resto de usuarios que se encuentre alrededor de los jugadores .

2.2.2.5 Motor.

El motor permite la movilidad del Kinect hacia arriba o hacia abajo más o menos unos 30ª, con el fin de calibrar cada espacio concreto por lo que la altura máxima está recomendada a uno o dos metros.

El motor también controla la cámara, activa el zoom y brinda la posibilidad de ampliar el espacio de juego.

2.2.3 SDK de Kinect.

El SDK de Kinect presenta una infinidad de posibilidades a los usuarios, científicos, programadores, estudiantes, y en fin a todo el público en general a desarrollar aplicaciones innovadoras aprovechando todo el potencial que este dispositivo presenta, además puede ser aplicado en una infinidad de áreas.

El SDK de Kinect proporciona Software sofisticado y herramientas para ayudar a los desarrolladores a aprovechar la forma más rica y natural del Sensor.

El SDK incluye:

- **Hardware de Kinect** Son los componentes de hardware, está conformado por el sensor Kinect, el cable USB y el adaptador USB.

- **Controladores Kinect** para usar en una computadora con Windows 7 como por ejemplo:

  - Conjunto de micrófonos Kinect

  - Controles de audio y video

  - Funciones de enumeración de dispositivos, que permiten a una aplicación que utilice más de un Kinect

- **Skeletal Tracking** de una o dos personas que estén en el rango de visión de Kinect

- **Cámara de Profundidad** que calculará la distancia del objeto al sensor Kinect

- **Procesamiento de audio** para sus 4 Micrófonos.

- **APIs**

- **Ejemplos de códigos fuente**

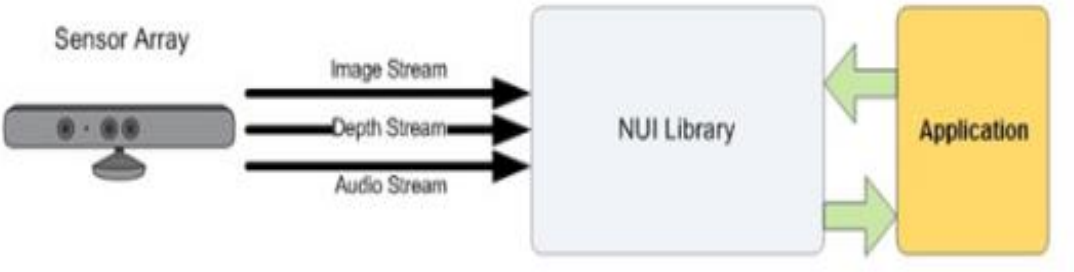


FIGURA 12 Interacción Hardware - Software Con La Aplicación[12]

2.2.4 Arquitectura.

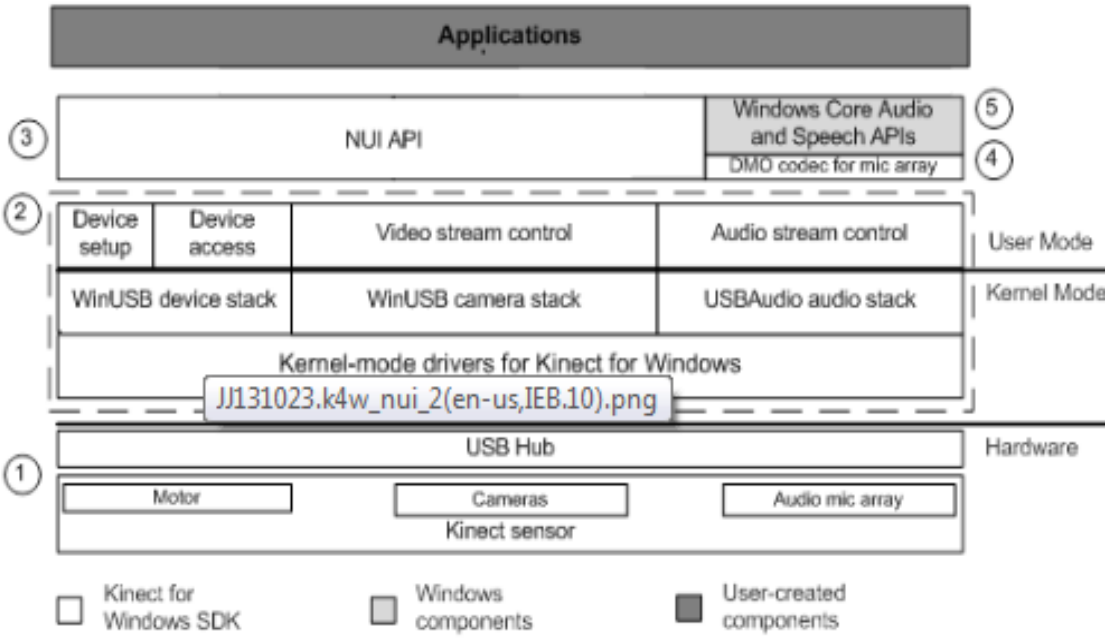


FIGURA 13 Arquitectura De Kinect[13]

[12] FIGURA 12 Tomada de: http://msdn.microsoft.com/en-us/library/jj131023

[13] FIGURA 13 Tomada de: http://msdn.microsoft.com/en-us/library/jj131023

En donde los números corresponden a lo siguiente:

1. El Hardware Kinect

2. Los drivers de Kinect desarrollado para Windows 7 , que se instalan al ejecutar el SDK y brindan soporte para :

   - El sistema de micrófonos como un dispositivo kernel-mode al que podemos tener acceso con los APIs estándares de Windows

   - Streaming de datos e imágenes de profundidad

   - Funciones de enumeración de dispositivos que permitirán que una aplicación puede ocupar más de un sensor Kinect conectado a la misma computadora.

3. NUI API, es un conjunto de APIs que recopilan los datos capturados por los sensores de imagen y además controlan el dispositivo

4. Kinect Audio DMO amplia el soporte de micrófonos de Windows 7 para exponer la localización de la formación de la fuente acústica.

5. APIs estándares de Windows.

2.2.5 Nui Api.

El NUI API (Natural User Interface) o (Interfaz Natural de Usuario) es el núcleo principal de Kinect para la API de Windows, que brinda soporte al manejo de datos y controla algunas de las propiedades del sensor como:

- Brindar soporte al Skeletal Tracking por medio de las imágenes procesadas y datos de profundidad.
- Permite el acceso a las imágenes de profundidad captadas por el sensor Kinect
- Accesos a los distintos Kinect que estén conectados al computador

El NUI API cuenta con Software para reconocer y realizar seguimiento de un cuerpo humano, permite que se reconozca hasta dos personas en frente de la cámara, la integración de APIs de Windows para permitir la ejecución de comandos de voz. Además cuenta con una amplia integración con el SDK para realizar el seguimiento del rostro Humano.

El NUI API se divide en los siguientes subsistemas que son:

2.2.5.1 Data Stream.

Permite la captura de datos como color, audio, datos de profundidad y permite procesar los datos de profundidad para generar datos sobre el esqueleto.

2.2.5.1.1 Datos de la imagen a Color.

Los datos de la imagen a color tienen 2 niveles de calidad y dos formatos diferentes. El nivel de calidad determina la velocidad en que los datos son transferidos desde el sensor y el formato determina si los datos de la imagen de color se codifican como RGB o YUV.

Ambos formatos usan datos tomados por la misma cámara, por lo que representan la misma imagen. Kinect envía los datos al PC a través de un cable USB, que ofrece un ancho de banda determinado

Formatos RGB y YUV:

- **RGB,** proporciona mapas de bits a color de 32 bits lineares con formato X8R8G8B8, dentro del espacio de colores RGB. Para usarlo es necesario especificar Color o Color_YUV cuando se abre el flujo de datos.

- **YUV**, proporciona mapas de bits de 16 bit con corrección gamma (la corrección gama añade transparencia a la imagen). Como este flujo usa 16 bits por cada pixel, este formato usa menos memoria y asigna menos buffer de memoria cuando se abre el flujo de datos. Para trabajar con YUV es necesario especificar YUV COLOR cuando se abre el flujo de datos. Esta opción está disponible sólo para resolución 640 x 480 y con una frecuencia de 15 fps.

La imagen que es captada por el sensor a 1280 x 1024 es comprimida y transformada a RGB antes de la transmisión al Runtime, el Runtime descomprime estos datos antes de entregarlos a la aplicación. El uso de la compresión permite el envío de datos a una tasa de 30fps, pero el algoritmo usado implica una pérdida de fidelidad de la imagen.

2.2.5.1.1.1 Funcionamiento de la cámara RGB.

Cada imagen está formada por un conjunto de píxeles. Cada pixel de la imagen está compuesto por cuatro componentes que representan los valores rojo, verde, azul y otro componente que es el valor de transparencia (alfa), en el caso de imágenes de formato tipo RGBa, o un valor vacio si es del tipo RGB.

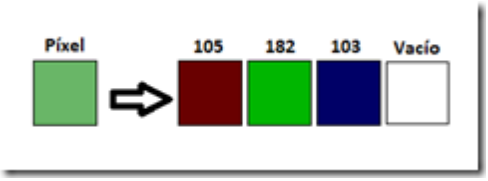


FIGURA 14 Representación de un Pixel[14]

Cada componente del pixel tiene un valor entre 0 y 254, que corresponde a un byte. El sensor Kinect codifica las imágenes que obtiene en un vector de bytes, donde cada byte representa un componente de cada pixel.

La organización de los pixeles es de arriba a abajo y de izquierda a derecha donde los cuatro primeros elementos del vector son los valores rojo, verde, azul y alfa del pixel de arriba a la izquierda mientras que los cuatro últimos serán del pixel de abajo a la derecha.

FIGURA 15 Disposición de los pixeles RGB en el array de bytes[15]

2.2.5.1.2 Datos de la cámara de Profundidad.

La cámara de profundidad en si se encarga de generar un vector de bytes que representa la distancia del objeto hacia el sensor.

[14] FIGURA 14 tomada de http://blogs.msdn.com/b/esmsdn/archive/2011/07/20/reto-kinect-usar-las-c-225-maras-del-sensor.aspx
[15] FIGURA 15 tomada de http://blogs.msdn.com/b/esmsdn/archive/2011/07/20/reto-kinect-usar-las-c-225-maras-del-sensor.aspx

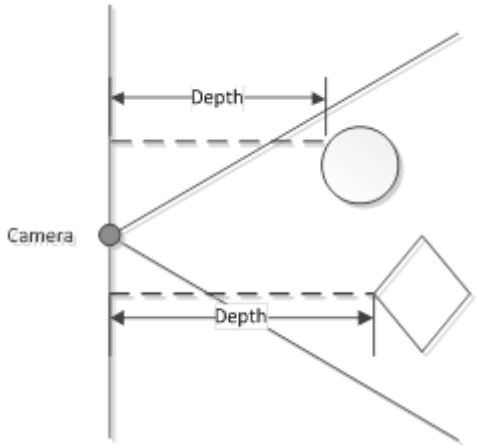


FIGURA 16 Campo de visión del sensor de Profundidad[16]

El sensor de profundidad almacena una imagen en escala de grises de todo el campo visible del sensor de profundidad, en donde cada pixel representa la distancia cartesiana más cercana desde la cámara al objeto (La coordenadas cartesianas X, Y o Z son medidas en milímetros). Las coordenadas cartesianas del sensor de profundidad solo representan la ubicación de un pixel en el marco de profundidad.

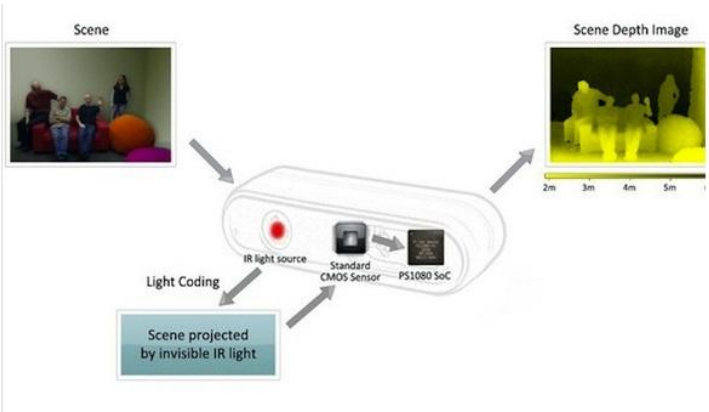


FIGURA 17 Almacenamiento de una imagen en escala de Grises[17]

El Rango del sensor de profundidad son 2 , el rango por defecto (Default Range) y el rango de cerca(Near Range) , el rango por defecto está habilitado tanto para

[16] FIGURA 16 Tomada de http://msdn.microsoft.com/en-us/library/hh973078#CommunityContent
[17] FIGURA 17 Tomada de http://animusproject.wix.com/web/apps/blog/month/2012/5/page/1

Windows como para Xbox 360 , y el rango de cerca solo está habilitado para el sensor que se ocupe en Windows

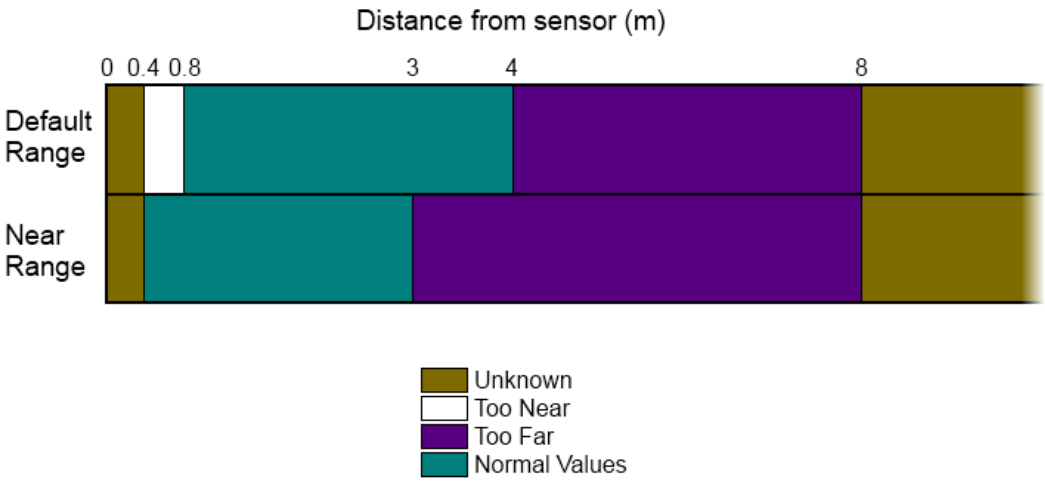


FIGURA 18 Rangos de distancias permitidas para Kinect[18]

Cada pixel es guardado en 2 bytes del vector, esto se debe a que como Kinect tiene dos cámaras de infrarrojos cada pixel guarda su distancia a cada cámara.

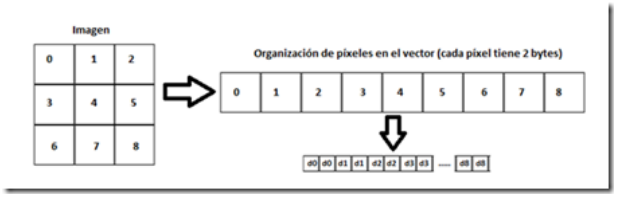


FIGURA 19 Disposición de los pixeles de profundidad en el vector de bytes[19]

Un valor de profundidad igual a cero indica que no hay datos de profundidad disponibles para esa posición, puede ser debido a que todos los objetos están o muy cerca de la cámara o muy lejos de ella.

De esta manera todo el campo de visión del Kinect es almacenado en dos mapas, uno de color y el otro de profundidad, y con el empleo del proyector de rayos

[18] FIGURA 18 Tomada de http://msdn.microsoft.com/en-us/library/hh973078#CommunityContent
[19] FIGURA 19 Tomada de http://blogs.msdn.com/b/esmsdn/archive/2011/07/20/reto-kinect-usar-las-c-225-maras-del-sensor.aspx

infrarrojos se logra eliminar parcialmente la luz ambiente, ya que no registra la luz visible y por ende, no tendrá muchas medidas erróneas.

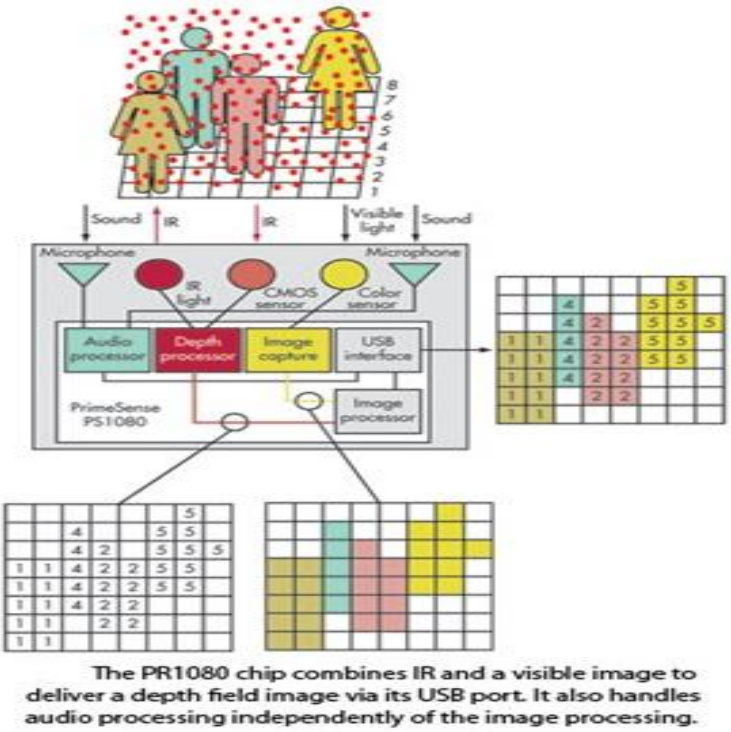


FIGURA 20 Proceso de adquisición de datos de la cámara RGB y sensor de profundidad[20]

2.2.5.2 Skeletal Tracking.

Es la función más prometedora de Kinect, porque permite detectar la figura humana (esqueleto) cuando se está moviendo, estos movimientos son asociados a través de unos puntos o Joints los cuales grafican el esqueleto humano.

Para la creación del cuerpo Kinect coge las siguientes 20 articulaciones

[20] FIGURA 20 Tomada de http://animusproject.wix.com/web/apps/blog/month/2012/5/page/1

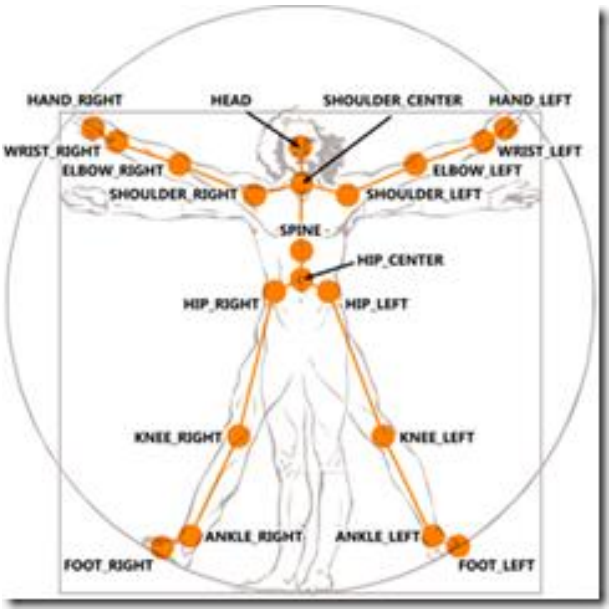


FIGURA 21 Forma de Kinect para reconoce al esqueleto humano a través de puntos o Joints[21]

Todo esto se logra debido a que el Skeletal Tracking tiene almacenado los datos de la imagen a color y los datos de la cámara de profundidad, lo que permite reconocer entre objetos y seres humanos ya que asocia los parámetros del cuerpo humano como extremidades superiores , articulaciones e incluso gestos ,para que de esta forma el cuerpo humano sea reconocido y detecte el movimiento de todas las partes que conforman el esqueleto humano(brazos , manos , muñecas , brazos rodillas codos , cadera , etc. .)

FIGURA 22 Detección de movimiento [22]

El Skeletal Tracking consta de un SkeletonFrame y este a su vez está compuesto por un vector (array) de SkeletonData (clase que contiene los datos de cada esqueleto),

[21] FIGURA 21 Tomada de http://blogs.msdn.com/b/esmsdn/archive/2011/08/09/reto-sdk-de-kinect-detectar poses-con-skeletal-tracking.aspx
[22] FIGURA 22 Tomada de http://animusproject.wix.com/web/apps/blog/month/2012/5/page/1

uno por cada esqueleto que el Skeletal Tracking reconoce. No todos los Skeleton Frame contienen SkeletonData.

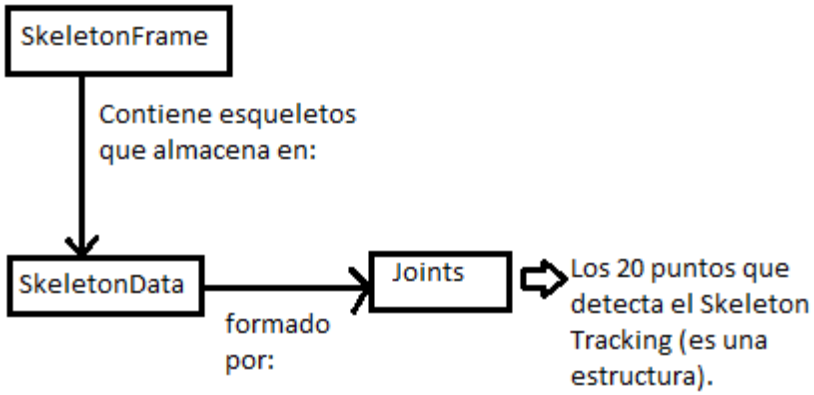


FIGURA 23 Esquema Skeletal Tracking[23]

De esta manera cuando la persona realice un movimiento como mover los brazos arriba o abajo, saltar, mover las manos , el sensor capturara ese movimiento y lo trasladara al Kinect para que el asocie esta información y lo muestre en la aplicación .

Este proceso es el método por el que Kinect es capaz de detectar la figura humana cuando se sitúa delante. Consta de 6 pasos:

1- El sensor lanza una serie de puntos

2- Kinect crea el mapa de profundidad a partir de los puntos detectados

3- Detecta el suelo y separa los objetos del fondo para encontrar el contorno humano.

4- Asocia las partes detectadas para hace una clasificación de las partes humanas.

5- Identifica las articulaciones.

[23] FIGURA 23 Tomada de http://animusproject.wix.com/web/apps/blog/month/2012/5/page/1

6- Simula el esqueleto humano.

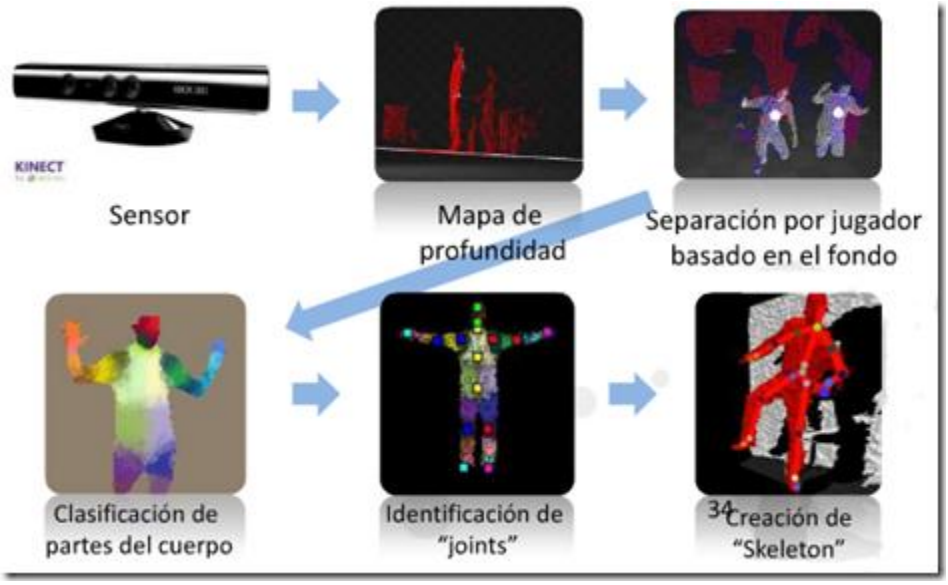


FIGURA 24 Proceso para el reconocimiento del Esqueleto[24]

Para todos los esqueletos detectados, tenemos los siguientes datos:

-A cada jugador le corresponde un único ID mientras se mueva dentro del rango de visión del sensor. Si sale del campo de visión de Kinect es posible que pierda su ID.

- El valor de la posición del jugador (el centro de masa del jugador), este valor es el único valor que tienen los jugadores con tracking pasivo.

-La posición de cada Joint viene determinada por coordenadas X, Y y Z, a diferencia de los datos de la imagen de profundidad, las medidas están expresadas en metros.

[24] FIGURA 24 Tomada de http://blogs.msdn.com/b/esmsdn/archive/2011/08/22/reto-sdk-kinect-reconocer-gestos-con-skeletal-tracking.aspx

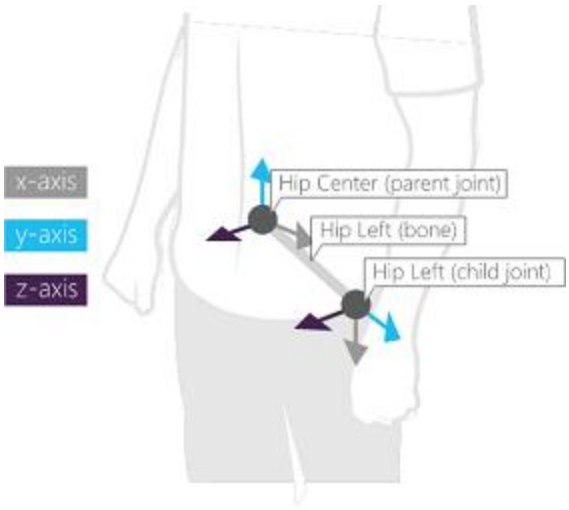


FIGURA 25 Eje de coordenadas del Esqueleto[25]

Como se puede ver, se trata de un sistema diestro de coordenadas que coloca el conjunto de sensores en el punto de origen con el lado positivo del eje z, extendiéndose en la dirección en que apunta el conjunto de sensores, el lado positivo del eje y extendiéndose hacia arriba y el lado positivo del eje x extendiéndose hacia la izquierda (respecto el Kinect). Cada eje indica:

- Eje X, Posición horizontal, viene dada por la distancia en metros del Kinect al Joint a lo largo del eje X.
- Eje Y, Posición vertical, viene dada por la distancia en metros del Kinect al Joint a lo largo del eje Y.

- Eje Z, Distancia desde el Kinect medida en metros.

[25] FIGURA 25 Tomada de http://msdn.microsoft.com/en-us/library/hh973073

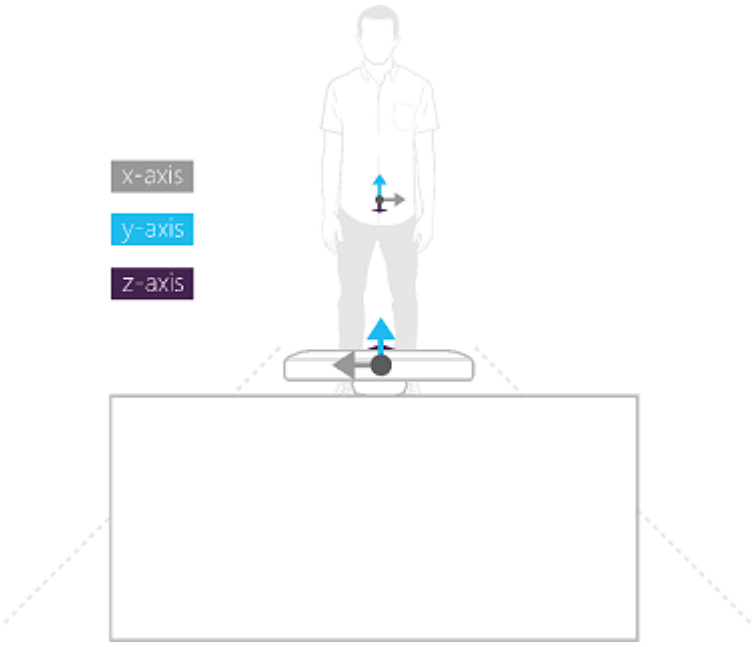


FIGURA 26 Orientación absoluta cundo el usuario esta de frente a la cámara[26]

2.2.5.3 Speach.

El sistema de micrófonos consiste en un conjunto de cuatro micrófonos dispuestos en forma de L. El tener el conjunto de micrófonos en esta posición da diversas ventajas respecto de un micrófono sólo:

-Mejora de la calidad de audio

-Como el audio llega en diferente tiempo a cada micrófono, se puede determinar la dirección donde está el origen del sonido y usar el conjunto de micrófonos como un micrófono direccional orientable.

Las aplicaciones que usen el SDK podrán utilizar los micrófonos para lo siguiente:

- Captura de audio de alta calidad

[26] FIGURA 26 Tomada de http://msdn.microsoft.com/en-us/library/hh973073

- Formación de haz y localización de la fuente

- Reconocimiento de voz

FIGURA 27 Reconocimiento de Voz[27]

2.2.6 Técnica utilizada por Kinect: Luz Estructurada.

La técnica utilizada por el sistema de imágenes en 3D de PrimeSense en el Kinect se llama Luz Estructurada en 3D. Esta técnica se utiliza en muchas aplicaciones industriales, tales como el control de la producción y medición de volumen, e implica escáneres de alta precisión y son muy costosos. Kinect es el primer dispositivo que aplicar esta técnica en un producto de consumo.

2.2.6.1 Escáner de Luz Estructurada en 3D.

"El escáner 3D consta de una fuente de luz (que proyectará el haz) y una cámara (que captará los puntos/líneas de las superficies) separados entre sí. Para escanear el objeto se define un sistema de coordenadas esféricas para determinar cada punto del espacio tridimensional que se está capturando."[1]

La mayoría de los escáneres de luz estructurada se basan en la proyección de un haz de luz que impacta sobre una superficie 3D, utilizando la deformación del haz de luz, los cuales se pueden observar desde un punto distinto al de la fuente, dichas deformaciones permiten generar puntos (Coordenadas) en las

[27] FIGURA 27 Tomada de http://animusproject.wix.com/web/apps/blog/month/2012/5/page/1
[1] Tomado de http://es.wikipedia.org/wiki/Esc%C3%A1ner_de_luz_estructurada

superficies que impactan, lo que permite medir la distancia desde cada punto a la cámara y así reconstituir el volumen de la superficie en 3D. Este método puede extenderse a la proyección de muchos haz de luz al mismo tiempo, que proporciona un alto número de muestras simultáneamente

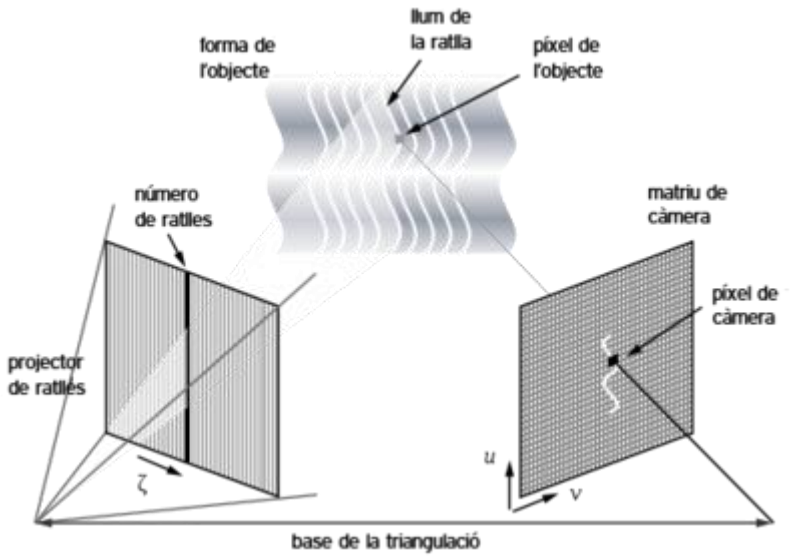


FIGURA 28 Técnica escáner de luz estructurada[28]

El sistema Kinect es algo diferente. En lugar de proyectar las franjas de la luz visible, el proyector de infrarrojos del Kinect envía un patrón de haces de luz infrarroja (llamado una codificación imagen IR por PrimeSense), que rebota en los objetos y es capturado por el sensor de imagen CMOS estándar, esta imagen capturada se pasa al chip de PrimeSense para que se traduzca a la imagen de profundidad.

[28] FIGURA 28 Tomada de http://es.wikipedia.org/wiki/Esc%C3%A1ner_de_luz_estructurada

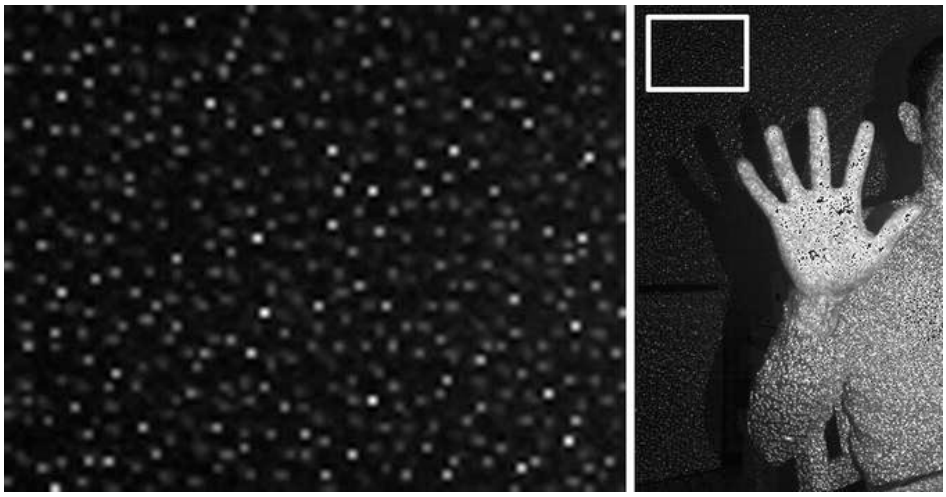


FIGURA 29 Código de imagen IR [29]

FIGURA 30 Mapa de profundidad reconstruida desde los patrones de códigos de luz infrarrojo. [30]

2.2.6.2 Transformación de los códigos de imagen de luz a un mapa de profundidad.

Una vez que el patrón de la luz infrarroja de codificación se reciba, PS1080 PrimeSense chip, compara la imagen recibida con una imagen de referencia almacenados en la memoria del chip como el resultado de una rutina de calibración realizada en cada dispositivo durante el proceso de producción. La comparación de la imagen "superficie" de referencia y el patrón de infrarrojos de entrada se traduce por

[29] FIGURA 29 Tomada de Autores Tesis
[30] FIGURA 30 Tomada de Autores Tesis

el chip en una imagen de tamaño VGA de profundidad de la escena que puede acceder a través de la API de Windows.

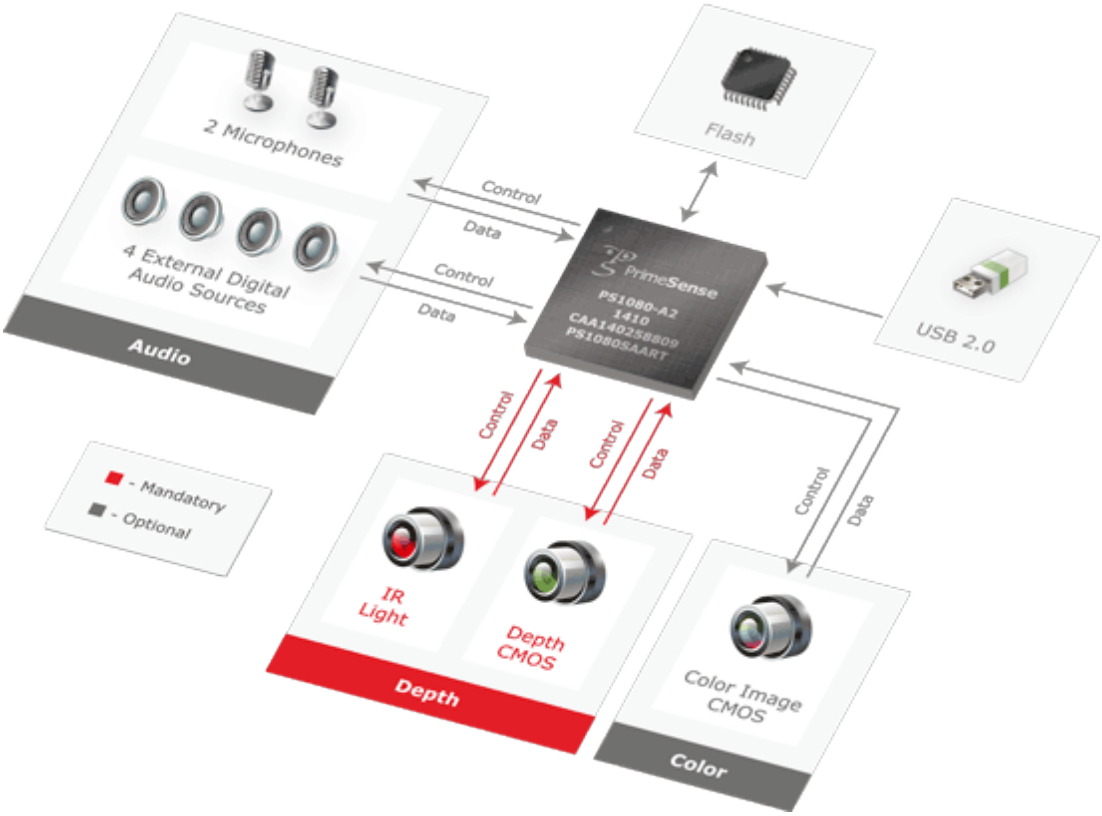


FIGURA 31 Sistema Del Chip PrimeSense Ps1080[31]

2.2.7 Requerimiento de Hardware y Software

2.2.7.1 Hardware.

- Kinect Para Xbox-360
- Computadora Con Procesador Dual – Core 2.66 Hz O Superior
- Tarjeta Grafica Compatible Con Windows 7 Que Soporte Directx9.0
- 2GB De RAM (4 GB Recomendado).

2.2.7.2 Software.

- Windows 7 (x86 o x64)

[31] FIGURA 31 Tomada de http://www.abclinuxu.cz/clanky/kinect-pro-xbox-360-a-gnu-linux-openni#!/-1/

- Visual Studio 2010 cualquier edición
- Microsoft .NET Framework 4.0

2.3 Arduino

2.3.1. Introducción

Arduino es una plataforma electrónica de código abierto compuesta por un microcontrolador, un lenguaje de programación sencillo y un IDE (Integrated Development Environment). Es una herramienta para crear aplicaciones interactivas y está diseñado para simplificar el trabajo requerido para programación, pero es lo suficientemente flexible para desarrollar proyectos complejos.

Desde su origen en 2005, han sido vendidas más de 200000 placas Arduino, y hay un gran creciente número de proyectos usándolo como su principal núcleo de procesamiento. Como Arduino es una plataforma de código abierto se tiene a disposición varios tutoriales, libros de referencia y librerías disponibles para los usuarios. Los programadores de Arduino están mejorando constantemente la tecnología para que las tarjetas puedan especializarse en diferentes tareas.

La simplicidad intencional que se tiene con la plataforma Arduino ha permitido tener acceso a la programación para personas que nunca antes pensaron usar o programar un microcontrolador.

2.3.2. Hardware

La tarjeta Arduino está construida con un microcontrolador AVR de 8-bit Atmel. Dependiendo de la tarjeta se puede encontrar diferentes chips que incluyen: ATmega8, ATmega168, ATmega328, ATmega1280, y ATmega2560. La tarjeta Arduino permite utilizar los pines de entrada y salida hacia otros circuitos.

Arduino es un hardware de libre circulación y cada diagrama esquemático está disponible como un archivo tipo EAGLE (EAGLE es un diseño de PCB, autorouter y CAM software de Cadsoft). De esta manera es posible crear por nuestra cuenta una tarjeta Arduino. Todo esto está explicado con detalle en http://arduino.cc/en/Main/Policy.

2.3.3. Descripción De La Tarjeta Arduino

El diagrama de la figura muestra un esquema básico de los componentes utilizados en la mayoría de tarjetas Arduino.

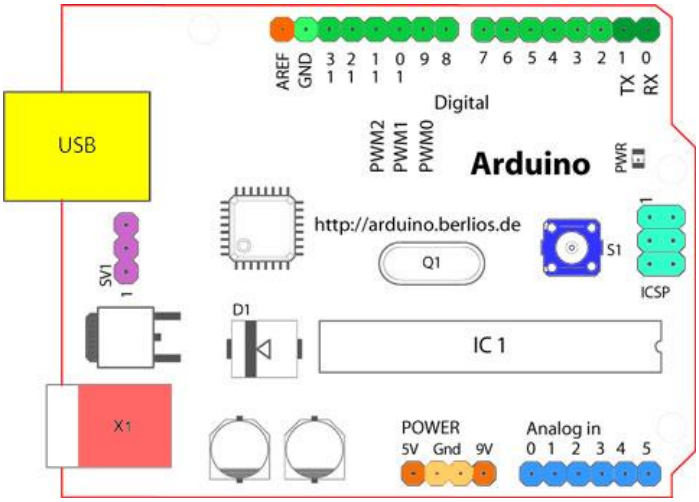


FIGURA 3232 Diagrama Arduino UNO[32]

[32] FIGURA 32 Tomada de http://arduino.cc/es/Reference/Board

2.3.3.1. Componentes

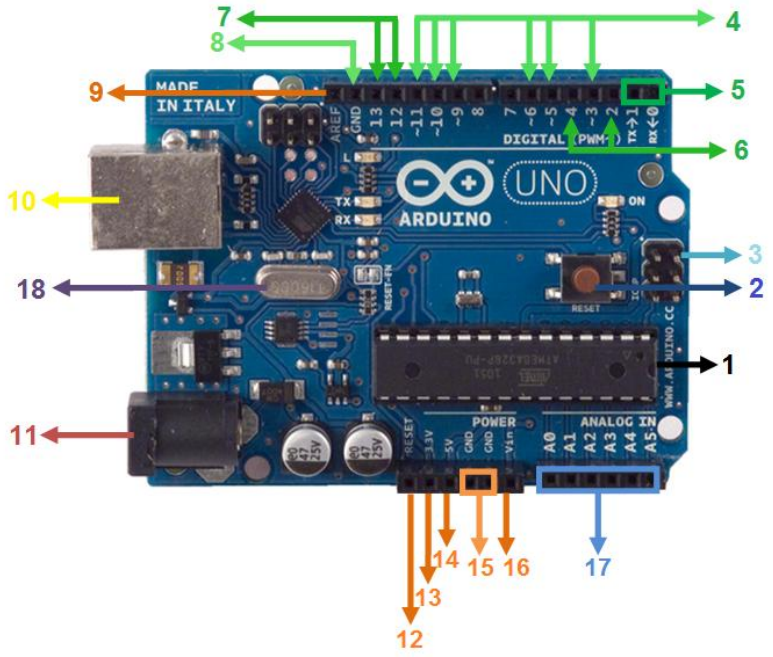


FIGURA 33 Descripción de partes de Arduino UNO[33]

1. Microcontrolador ATmega328

Es un microcontrolador de la compañía Atmel que cuenta con 32KB de memoria flash, 2KB de memoria RAM y 1KB de memoria EEPROM.

Características

- Voltaje de Operación: 5V
- Memoria Flash: 32 KB de los cuales 512 bytes son utilizados por el bootloader
- Numero de Pines : 32
- CPU: AVR 8 Bit
- SRAM 2 KB
- EEPROM 1 KB
- Velocidad del Reloj 16 MHz
- Bootloader preinstalado

2. Botón Reset

Utilizado para suministrar un valor LOW (0V) y reiniciar el microcontrolador.

[33] FIGURA 33 Tomada de http://arduino.cc/en/Main/ArduinoBoardUno imagen editada

3. ICSP
Es un conector para la programación ICSP (In Circuit Serial Programming, o Programación Serial en circuito). El ICSP es el sistema utilizado en los dispositivos PIC para programarlos sin necesidad de tener que retirar el chip del circuito del que forma parte.

4. PWM
Los pines 3, 5, 6, 9, 10 y 11 proporcionan 8 bits de salida PWM con la función analogWrite ().

5. Serial: 0 (RX) y 1 (TX).
Utilizado para recibir (RX) y transmitir (TX) datos serie TTL.

6. Interrupciones externas: 2 y 3.
Estas terminales pueden ser configuradas para disparar una interrupción con un valor bajo, un pulso de subida o bajada, o un cambio de valor.

7. SPI (Serial Peripheral Interface)

Los pines 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK); Soportan comunicación SPI. El Bus SPI es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.
Esta funcionalidad viene con el hardware pero no está incluida en el lenguaje Arduino.

8. GND
Pines de Tierra

9. AREF
Referencia de voltaje para las entradas analógicas. Utilizada con la función analogReference().

10. USB
Arduino utiliza el puerto USB para comunicarse con la computadora, es reconocido como un puerto COM y debe ser configurado al momento de instalar Arduino.
Proporciona una alimentación de 5V a la tarjeta.

11. Conector de Alimentación

Conector que proporciona alimentación a la tarjeta.

12. Reset.

Usado para añadir un botón de reset a los Shields que no dejan Acceso a la tarjeta.

Un “Shield” es una tarjeta adicional compatible con Arduino que nos permite ampliar sus capacidades.

13. 3.3V

Una fuente de 3.3 voltios generada por el chip FTDI de la placa.

14. 5V

La alimentación regulada utilizada para alimentar el microcontrolador y otros componentes de la placa. Esta puede venir de VIN a través de un regulador en placa o ser proporcionada por USB u otra fuente regulada de 5V.

15. GND

Pines de Tierra.

16. VIN

Es el voltaje de entrada a la placa Arduino cuando se está utilizando una fuente de alimentación externa (En comparación con los 5 voltios de la conexión USB o de otra fuente de alimentación regulada). Puede proporcionar voltaje a través de este pin.

17. Analog IN

Los pines de entrada analógicos soportan conversiones analógico-digital (ADC) de 10 bit utilizando la función analogRead(). Las entradas analógicas pueden ser también usadas como pines digitales: entrada analógica 0 como pin digital 14 hasta la entrada analógica 5 como pin digital 19.

18. Cristal

Un cristal oscilador a 16Mhz la frecuencia es estable frente a variaciones de la tensión de alimentación.

2.3.3.2. Pines de entrada y salida

Arduino uno consta con 14 pines de entrada/salida digitales y 6 pines de entrada analógicos, como se muestra en la figura. , estos pines corresponden a los pines de entrada/salida del microcontrolador Atmega. La arquitectura de una tarjeta Arduino muestra estos pines en la superficie, para que sean fácilmente conectados a circuitos externos.

Los pines de Arduino se pueden establecer de dos maneras: entradas y salidas. Los pines de Atmega son puestos como pines de entrada por defecto, por lo que no se necesita especificar ningún pin como entrada en Arduino, aunque a menudo se lo especifica en el código para tener mayor claridad.

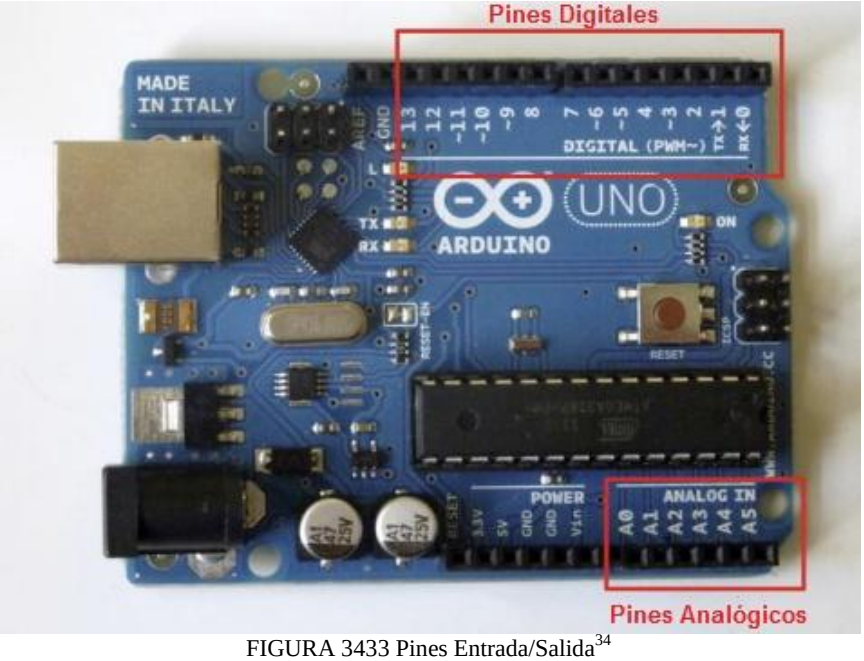


FIGURA 3433 Pines Entrada/Salida[34]

2.3.3.3. Pines Digitales

Arduino tiene 14 pines digitales, numerados del 0 al 13. Los pines digitales pueden ser configurados como Entradas (INPUT) o como Salidas (OUTPUT), usando la función pinMode(). En los dos modos los pines solo puedes enviar o recibir señales digitales, lo que consiste en dos estados diferentes: ON (HIGH, o 5V) y OFF(LOW, o 0V).

Los pines configurados como Salidas (OUTPUT), pueden proveer corriente y voltaje hacia dispositivos externos. Estos pines son usados para encender o apagar Diodos LED, control de motores, sistemas de control, además pueden ser usados para modulación por ancho de pulso (PWM).

Los pines configurados como Entradas (INPUT) están listos para leer datos enviados por dispositivos conectados a ellos, en estos pines se pueden leer señales digitales como, el estado de un switch, pulsos eléctricos entre otros.

[34] FIGURA 34 Tomada de Autores Tesis

2.3.3.4. Pines Analógicos

Los microcontroladores Atmega que se usan en las tarjetas Arduino contienen seis canales con conversores ADC (Análogo Convertido a Digital). La función de este dispositivo es convertir un voltaje de entrada analógico en un número digital proporcional a la magnitud de voltaje de entrada en relación al voltaje de referencia (5V).

El conversor Atmega en las tarjetas Arduino tiene una resolución de 10 bit, lo que significa que retornara números enteros de 0 a 1023 basados en los valores de potencial aplicados y comparados con la referencia de 5V. una entrada de 0V produce un valor de 0, un potencial de 5V a la entrada un valor de 1023 y el valor medio 2,5V retornara el número 512.

Estos pines pueden ser establecidos como entradas o salidas exactamente igual que los digitales. Llamándolos A0, A1, etc.

2.3.4. Arduino IDE

La IDE de Arduino es una multiplataforma escrita en Java y está derivada de la IDE de Processing y Wiring. Una IDE es básicamente un programa diseñado para facilitar el desarrollo de software, la IDE de Arduino incluye un editor de códigos (un editor de texto con algunas características especiales para la programación, tales como sintaxis, resaltado y subrayado automático) y un compilador que transforma el código escrito en el programa en instrucciones entendibles para la máquina.

Lo único que se necesita saber sobre el IDE de Arduino es que es una parte del software la cual nos permite escribir fácilmente los códigos de programación y descargarlos a nuestra tarjeta, la siguiente figura muestra como se ve la pantalla:

FIGURA 3534 IDE de Arduino[35]

Como se ve en la parte inferior izquierda el IDE de Arduino muestra el tipo de tarjeta Arduino y el puerto serial donde está conectado.

La siguiente tabla describe los diferentes símbolos que encontramos en la barra de herramientas

Símbolo	Descripción
	Verify Verifica si existen errores en el código.
	Upload Compila el código y lo carga en la tarjeta Arduino
	New Crea una nueva pestaña
	Open Abre un archivo creado en Arduino
	Save Guarda el archivo

[35] FIGURA 35 Tomada de Autores Tesis

	Serial Monitor Abre un monitor del puerto serial ocupado

Tabla 1 Símbolos de la barra de Herramientas del IDE de Arduino

CAPITULO 3

3.1 Estudio de los códigos fuentes SDK de Kinect Beta 2.

El siguiente estudio está realizado de los códigos fuentes de coding4fun la versión Beta 2 del SDK de Kinect, los cuales se pueden encontrar en

http://files.ch9.ms/coding4fun/KinectForWindowsSDKQuickstarts.zip.

- ***Los códigos fuente que se van a analizar son los siguientes:***

- Skeletal Tracking.
- Camera Fundamentals.
- Depth Data.

3.1.1. Skeletal Tracking

- Código fuente:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
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 Microsoft.Research.Kinect.Nui;

using Coding4Fun.Kinect.Wpf;
```

Microsoft.Research.Kinect.Nui Es la librería necesaria para empezar a utilizar y programar el paquete de controles disponibles que tiene el sensor Kinect. Esta librería siempre debe estar incluida en el desarrollo de cualquier proyecto ya que si no es incluida no se obtendrá el enlace de datos entre la interfaz Kinect con Visual Studio C#.

Coding4Fun.Kinect.Wpf Es la librería que hace referencia al paquete de controles de Coding4Fun Toolkit. Contiene métodos para trabajar con el sensor de profundidad y también con los datos del Skeletal Tracking como por ejemplo, hacer referencias a escalas, realizar una carga de imágenes más sencilla.

```
namespace SkeletalTracking

{

   public partial class MainWindow : Window

   {

      public MainWindow()

      {

         InitializeComponent();

      }

          Runtime nui = new Runtime();
```

Mediante la función ***"Runtime"*** Se inicializa el sensor Kinect a la cual se le puede asignar cualquier nombre de variable, para este ejemplo se utiliza como nombre de variable "nui".

Runtime <Variable> = new Runtime();

```
private void Window_Loaded(object sender, RoutedEventArgs e)
```

private void Window_Loaded.- Evento creado para que se ejecuten todos los eventos, objetos, métodos del sensor Kinect.

```
      {

         nui.Initialize(RuntimeOptions.UseSkeletalTracking);
```

nui.Initialize.- Sirve para inicializar o empezar a usar el SkeletalTracking del Kinect.

RuntimeOptions.- Es una instrucción que nos permite utilizar las distintas funciones que presenta Kinect, para usar el Skeletal tracking se utiliza "UseSkeletalTracking

```
#region TransformSmooth

nui.SkeletonEngine.TransformSmooth = true;
```

nui.SkeletonEngine.TransformSmooth = true.- Es una instrucción que sirve para inicializar la corrección de los datos, es decir, que tengan un mayor grado de precisión, esta instrucción debe de estar establecida en "true" para que pueda ejecutarse.

```
var parameters = new TransformSmoothParameters

{

   Smoothing = 0.75f,

   Correction = 0.0f,

   Prediction = 0.0f,

   JitterRadius = 0.05f,

   MaxDeviationRadius = 0.04f
```

var <Nombre de Variable> = new TransformSmoothParameters.- Va a ser una variable que va a almacenar la configuración de los diferentes parámetros para lograr una mayor precisión en los datos enviados del Skeletal Tracking, como por ejemplo la fluidez de datos, la corrección, predicción, salto de radio, desviación máxima de los datos.

```
};

nui.SkeletonEngine.SmoothParameters = parameters;
```

nui.SkeletonEngine.SmoothParameters = <Nombre de Variable>.- En esta parte se va a ejecutar "nui.SkeletonEngine.SmoothParameters" con el nombre de la variable que se haya utilizado para almacenar los distintos parámetros configurados.

```
#endregion

nui.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(nui_SkeletonFrameReady);
```

nui.SkeletonFrameReady.- Es una instrucción que permite crear un evento para recibir los datos enviados del Skeletal Tracking.

new EventHandler<SkeletonFrameReadyEventArgs>(<Nombre_Evento>).- Es un comando que permite dar el nombre de cómo se va a llamar el método.

```
}
void nui_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
```

Void <Nombre de Metodo>. - Método en el cual se va a ejecutar todos los eventos que tiene SkeletonFrameReady.

```
    SkeletonFrame allSkeletons = e.SkeletonFrame;
```

Variable que contiene todas las tramas de los movimientos del usuario

```
    SkeletonData skeleton = (from s in allSkeletons.Skeletons
                    where s.TrackingState ==
                                        SkeletonTrackingState.Tracked
                    select s).FirstOrDefault();
```

Se crea una variable llamada Skeleton donde se especifica los parámetros para determinar si el usuario está en movimiento con la opción **"s.TrackingState == SkeletonTrackingState.Tracked"** la cual va a determinar si existe movimiento y va a capturar los datos.

```
    if(skeleton != null)
```

Se verifica si la variable skeleton contiene datos o está vacía.

```
    {
      SetEllipsePosition(headEllipse, skeleton.Joints[JointID.Head]);
      SetEllipsePosition(leftEllipse,
      skeleton.Joints[JointID.HandLeft]);
      SetEllipsePosition(rightEllipse,
      skeleton.Joints[JointID.HandRight]);
    }
```

Captura los datos de los distintos puntos del cuerpo humano y se enlaza al objeto ellipse los datos captados los cuales serán enviados al método SetEllipsePosition donde consta las escala para mover los diferentes partes del cuerpo humano. (En este caso solo se está enlazando la cabeza, mano derecha y mano izquierda).

```
    }

    private void SetEllipsePosition(FrameworkElement ellipse, Joint joint)
    {
```

Se crea un evento llamado SetEllipsePosition en el cual se va a especificar la escala y se va a enlazar los objetos ellipse a los movimientos de los puntos en los ejes x e y.

```
        var scaledJoint = joint.ScaleTo(640, 480, .5f, .5f);
```

Se crea la escala de 640 x 480 pixeles el cual es el rango para que los datos captados de los puntos de las distintas partes del cuerpo humano tengan la posibilidad de moverse acorde a la escala dada, ".5f, .5f" son dos parámetros que permiten ajustar la precisión del movimiento tanto en altura como en anchura estos valores se ajustan de acuerdo a las necesidades de cada proyecto.

```
        Canvas.SetLeft(ellipse, scaledJoint.Position.X);
        Canvas.SetTop(ellipse, scaledJoint.Position.Y);
```

Configura la posición de la ellipse para poder moverlos en la ventana principal de a acuerdo a la posición de los puntos del Skeletal Tracking, utilizando los métodos **Canvas.SetLeft** y **Canvas.SetTop** los cuales contienen los valores de las posiciones X y Y de los datos de los puntos captados.

```
    }

    private void Window_Closed(object sender, EventArgs e)
```

private void Window_Closed.- Evento creado para finalizar el uso del sensor Kinect.

```
    {
        nui.Uninitialize();
```

nui.Uninitialize.- Es una función que se encarga de cerrar todos los eventos, métodos, objetos del sensor Kinect.

```
    }
}
```

```
}
```

3.1.2. Camera Fundamentals

```
namespace CameraFundamentals
{
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
      }

      Runtime nui = new Runtime();
```

Mediante la función Runtime Se inicializa el sensor Kinect a la cual se le puede asignar cualquier nombre de variable, para este ejemplo se utiliza como nombre de variable "nui".

Runtime <Variable> = new Runtime();

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

private void Window_Loaded.- Evento creado para que se ejecuten todos los eventos, objetos, métodos del sensor Kinect.

```
         nui.VideoFrameReady += new EventHandler<ImageFrameReadyEventArgs>(nui_VideoFrameReady);
```

nui.VideoFrameReady.- Es una instrucción que permite crear un evento que habilita la camara RGB de kinect

EventHandler<ImageFrameReadyEventArgs>(Nombre_Evento>).- Es un comando que permite dar el nombre de cómo se va a llamar el método.

```
         nui.DepthFrameReady += new
EventHandler<ImageFrameReadyEventArgs>(nui_DepthFrameReady);
```

nui.DepthFrameReady.- Es una instrucción que permite crear un evento para recibir los datos de profundidad del Skeletal Tracking

EventHandler<ImageFrameReadyEventArgs>(<Nombre_Evento>).- Es un comando que permite dar el nombre de cómo se va a llamar el método.

```
nui.Initialize(RuntimeOptions.UseColor | RuntimeOptions.UseDepth);
```

nui.Initialize.- Sirve para inicializar o empezar a usar el Skeletal Tracking del Kinect.

RuntimeOptions.- Es una instrucción que nos permite utilizar las distintas funciones que presenta Kinect, para usar la cámara RGB y el sensor de profundidad se utiliza las opciones "UseColor" y "UseDepth" respectivamente.

```
nui.VideoStream.Open(ImageStreamType.Video, 2, ImageResolution.Resolution640x480,
ImageType.Color);
```

nui.VideoStream.Open.- Es una instrucción que permite abrir el flujo de datos de la cámara de video, es decir, que la cámara RGB va a captar los datos y los va a enviar a la aplicación que se esté ocupando, se especifica el tipo de flujo de imagen, la resolución que se quiera ocupar, y el tipo de Imagen debe ser especificado como "Color" para empezar a usar la cámara RGB.

```
nui.DepthStream.Open(ImageStreamType.Depth, 2, ImageResolution.Resolution320x240,
ImageType.Depth);

}
```

nui.DepthStream.Open.- Es una instrucción que permite abrir el flujo de datos del sensor de profundidad, capta todos los puntos que estén en el eje z y esos datos serán enviados a cualquier aplicación que la necesite, en este caso el tipo de flujo de imágenes debe ser especificado como profundidad "Depth", la resolución que se quiera ocupar y el tipo de imagen debe ser especificado como profundidad "Depth" para empezar a usar el sensor de profundidad.

```
void nui_DepthFrameReady(object sender, ImageFrameReadyEventArgs e)
```

void nui_DepthFrameReady.- Es un evento creado en el cual se vas a captar los datos de profundidad del cuerpo humano.

```
{

image2.Source = e.ImageFrame.ToBitmapSource();
```

image2.Source = e.ImageFrame.ToBitmapSource().- Esta es una instrucción utilizada de los métodos de extensión de los datos de profundidad de la librería Coding4Fun, en la cual se va a crear un objeto tipo Bitmap, el que será almacenado en el objeto image2.source y en este se va a desplegar los datos de profundidad del cuerpo humano.

```
    }

    void nui_VideoFrameReady(object sender, ImageFrameReadyEventArgs e)
```

void nui_VideoFrameReady.- Es un evento creado en el cual se vas a habilitar la camara RGB del sensor Kinect.

```
{
        PlanarImage imageData = e.ImageFrame.Image;
```

PlanarImage imageData = e.ImageFrame.Image.- la instrucción PlanarImage<Nombre de Variable> Crea una variable "imageData" la cual va a almacenar los datos capturados por la cámara RGB.

```
        image1.Source = BitmapSource.Create(imageData.Width, imageData.Height, 96, 96, PixelFormats.Bgr32,
null, imageData.Bits, imageData.Width * imageData.BytesPerPixel);
```

image1.Source = BitmapSource.Create().- Se crea una imagen Bitmap especificando el ancho y largo de la imagen que contiene los datos capturados, en este caso "imageData" se especifica el formato de pixeles "Bgr32" que es el formato de pixeles que se va a almacenar y luego se realiza una mezcla entre el ancho de la imagen "imageData.Width" y la paleta de Pixeles de la imagen "imageData.BytesPerPixel" para asignar sus respectivos colores a los diferentes elemento que estén en el campo de visión de la cámara.

```
    }

    private void Window_Closed(object sender, EventArgs e)
```

private void Window_Closed.- Evento creado para finalizar el uso del sensor Kinect.

```
    {
        nui.Uninitialize();
```

nui.Uninitialize.- Es una función que se encarga de cerrar todos los eventos, métodos, objetos del sensor Kinect.

```
    }
  }
}
```

3.1.3. Depth Data

```
namespace WorkingWithDepthData
{
  public partial class MainWindow : Window
  {
    public MainWindow()
    {
      InitializeComponent();
    }
    Runtime nui = new Runtime();
```

Mediante la función Runtime se inicializa el sensor Kinect a la cual se le puede asignar cualquier nombre de variable, para este ejemplo se utiliza como nombre de variable "nui".

Runtime <Variable> = new Runtime();

```
    private void Window_Loaded(object sender, RoutedEventArgs e)
```

private void Window_Loaded.- Evento creado para que se ejecuten todos los eventos, objetos, métodos del sensor Kinect.

```
    {
      nui.Initialize(RuntimeOptions.UseDepthAndPlayerIndex | RuntimeOptions.UseSkeletalTracking);
```

nui.Initialize.- Sirve para inicializar o empezar a usar el Skeletal Tracking del Kinect.

RuntimeOptions.- Es una instrucción que nos permite utilizar las distintas funciones que presenta Kinect, en este caso se utiliza las opciones "UseDepthAndPlayerIndex" las cuales van a captar los datos de profundidad y a determinar cuántas personas se encuentran en el rango de visión del sensor de profundidad asignando un índice de jugador "PlayerIndex".

```
      nui.DepthFrameReady += new EventHandler<ImageFrameReadyEventArgs>(nui_DepthFrameReady);
```

nui.DepthFrameReady.- Es una instrucción que permite crear un evento para recibir los datos de profundidad del Skeletal Tracking

EventHandler<ImageFrameReadyEventArgs>(<Nombre_Evento>).- Es un comando que permite dar el nombre de cómo se va a llamar el método.

```
      nui.DepthStream.Open(ImageStreamType.Depth, 2, ImageResolution.Resolution320x240,
        ImageType.DepthAndPlayerIndex);
```

***nui.DepthStream.Open*.-** Es una instrucción que permite abrir el flujo de datos del sensor de profundidad, capta todos los puntos que estén en el eje z y esos datos serán enviados a cualquier aplicación que la necesite, en este caso el tipo de flujo de imágenes debe ser especificado como profundidad "Depth", la resolución que se quiera ocupar y el tipo de imagen debe ser especificado como profundidad *"DepthAndPlayerIndex"* para empezar a usar el sensor de profundidad y asignarle un número de Índice para identificar a las personas que esté dentro del campo de visión del sensor de profundidad..

```
}

void nui_DepthFrameReady(object sender, ImageFrameReadyEventArgs e)
```

void nui_DepthFrameReady.- Es un evento creado en el cual se vas a captar los datos de profundidad del cuerpo humano.

```
{

    byte[] ColoredBytes = GenerateColoredBytes(e.ImageFrame);
```

byte[] <Nombre de Variable> = GenerateColoredBytes().- Se crea una vector tipo byte en el cual se va a convierte la información de los datos de profundidad de un pixel, en la información de los datos de color RGB en el evento "GenerateColoredBytes".

```
    PlanarImage image = e.ImageFrame.Image;
```

PlanarImage image = e.ImageFrame.Image.- La instrucción PlanarImage<Nombre de Variable> crea una variable "image" la cual va a almacenar los datos capturados por la cámara RGB.

```
    image1.Source = BitmapSource.Create(image.Width, image.Height, 96, 96, PixelFormats.Bgr32, null,

      ColoredBytes, image.Width * PixelFormats.Bgr32.BitsPerPixel/ 8);
```

image1.Source = BitmapSource.Create().- Se crea una imagen Bitmap especificando el ancho y largo de la imagen que contiene los datos capturados, en este caso "image" se especifica el formato de pixeles **"Bgr32"** que es el formato de pixeles que se va a almacenar y luego se realiza una mezcla entre el ancho de la imagen **"image.Width"** y la paleta de Pixeles de la imagen **"PixelFormats.Bgr32.BitsPerPixel/ 8"** la cual es dividida para 8 ya que cada pixel tiene un valor decimal de 0 a 254 , lo que corresponde a un byte y así asignar sus respectivos colores a los diferentes elemento que estén en el campo de visión de la cámara.

```
}

private byte[] GenerateColoredBytes(ImageFrame imageFrame)
```

private byte[] GenerateColoredBytes(ImageFrame imageFrame).-Es un evento creado vector tipo byte en el cual se va a transformar los datos de profundidad a datos de color RGB.

```
{
    int height = imageFrame.Image.Height;
    int width = imageFrame.Image.Width;
```

Se crean las variables **"height"** y **"width",** que van a contener los valores tanto de ancho como de largo de la trama de imágenes del objeto Image .

```
    Byte[] depthData = imageFrame.Image.Bits;
```

Se crea un vector tipo byte, en este caso **"dephData",** al tener 2 cámaras de infrarrojos cada pixel se corresponde con 2 bytes, en el vector siendo estos la distancia de ese pixel a cada cámara.

```
    Byte[] colorFrame = new byte[imageFrame.Image.Height * imageFrame.Image.Width * 4];
```

Se crea un vector tipo byte , llamado **"colorFrame",** el que va almacenar la información de los colores para todos los pixeles en la imagen , estos dato almacenados se obtienen de la multiplicación del alto por el ancho y por 4 , este cuatro significa la configuración **"bgr32"** , que es referente la configuración de pixeles (Azul, Verde, Rojo, y un byte vacio).Cabe recalcar que existe ora configuración de pixeles que es (Bgra32)que es referente a la configuración (Azul, Verde, Rojo y un byte de transparencia).

```
    const int BlueIndex = 0;
    const int GreenIndex = 1;
    const int RedIndex = 2;
```

Se crean unas constantes enteras llamadas **"BlueIndex, GreenIndex, RedIndex",** las cuales van a contener sus respectivos índices identificadas en el orden de 0,1,2, estos valores van a ser los que nos van a indicar la posición de los códigos de colores correspondientes a (Azul, Verde y Rojo)

```
    var depthIndex = 0;
```

Se crean una variable de índice de profundidad llamada **"dephIndex"**, el cual nos va a indicar la distancia de cada pixel a la cámara de infrarrojos.

```
    for (var y = 0; y < height; y++)
```

Se emplea una sentencia **"for"**, en la cual, la variable **"y"** va empezar con el valor en cero y va a seguir sumando 1 a su valor actual (y = y+1), mientras sea menor que la altura del objeto Image, para que de esta manera se ajuste el campo de visión de las cámaras infrarrojas a la correspondiente altura que tenga el objeto Image. El valor de la variable **"y"** empieza en cero, para que el campo de visión de las cámaras infrarrojas encajen de una manera correcta en el objeto Image , ya que si se cambia este valor , el campo de visión no podría encajar correctamente haciendo que este suba o baje y no podría ajustarse en la altura correcta del objeto Image.

```
{
  var heightOffset = y * width;
```

Se crea una variable **"heightOffset",** en la cual el valor de la variable **"y"** es multiplicado por el ancho del objeto Image "width", para obtener una compensación y de esta manera hacer más preciso el ajuste del campo de visión de las cámaras infrarrojas a la altura del objeto Image.

```
for (var x = 0; x < width; x++)
```

Se emplea una sentencia **"for",** en la cual, la variable **"x"** va empezar con el valor en cero y va a seguir sumando 1 a su valor actual (x = x+1), mientras sea menor que el ancho **"width"** del objeto Image, para que de esta manera se ajuste el campo de visión de las cámaras infrarrojas al correspondiente ancho que tenga el objeto Image. El valor de la variable **"x"** empieza en cero, para que el campo de visión de las cámaras infrarrojas encajen de una manera correcta en el objeto Image, ya que si se cambia este valor, el campo de visión no podría encajar correctamente ,haciendo que este tenga una distorsión y no podría ajustarse en la anchura correcta del objeto Image.

```
{
  var index = ((width - x - 1) + heightOffset) * 4;
```

La variable **"index",** almacena todos los colores de los objetos que estén en el campo de visión de las cámaras infrarrojas, los cuales se obtiene mediante un proceso matemático, en donde se resta el ancho del objeto imagen menos el valor de x disminuido en 1 y sumado con la variable **"heightOffset"**, ese resultado es multiplicado por 4, para obtener la configuración bgr32 que es referente la configuración de pixeles (Azul, Verde, Rojo, y un byte vacio).

```
  var distance = GetDistanceWithPlayerIndex(depthData[depthIndex], depthData[depthIndex + 1]);
```

La variable **"distance"** se encarga de almacenar la distancia que existe entre los distintos objetos que se encuentren en el campo de visión del sensor, con el método **"GetDistanceWithPlayerIndex()"** que se encarga de enlazar los datos de profundidad **"depthData"** y los datos de índice de profundidad **"depthIndex"** para obtener la distancia del pixel de los objetos al sensor.

```
if (distance <= 900)
{
    colorFrame[index + BlueIndex] = 255;
    colorFrame[index + GreenIndex] = 0;
    colorFrame[index + RedIndex] = 0;
```

Se utiliza la sentencia **"if"** para determinar la distancia de los objetos, en este caso si la distancia de cualquier objeto es menor o igual que 900, ese objeto será pintado del color azul, lo que va a indicar que el objeto está muy cerca al sensor.

```
}
else if (distance > 900 && distance < 2000)
{
    colorFrame[index + BlueIndex] = 0;
    colorFrame[index + GreenIndex] = 255;
    colorFrame[index + RedIndex] = 0;
```

Se utiliza la sentencia **"else if"** para determinar la distancia de los objetos , en este caso si la distancia de cualquier objeto es mayor que 900 y menor que 2000 , ese objeto que se encuentre en ese rango , será pintada del color verde , lo que va a indicar que el objeto está un poco alejado del sensor.

```
}
else if (distance > 2000)
{
    colorFrame[index + BlueIndex] = 0;
    colorFrame[index + GreenIndex] = 0;
    colorFrame[index + RedIndex] = 255;
```

Se utiliza la sentencia **"else if"** para determinar la distancia de los objetos, en este caso si la distancia de cualquier objeto es mayor que 2000, es objeto que se encuentre en ese rango , será pintada del color rojo , lo que va a indicar que el objeto está muy lejos del sensor.

```
}
if (GetPlayerIndex(depthData[depthIndex]) > 0)
```

Se enlaza los datos de profundidad y los datos de índice de profundidad al método **"`GetPlayerIndex`"**, el cual se va a ejecutar siempre y cuando los datos recibidos sean mayores que cero, para que la condición de la sentencia "if" se cumpla.

```
{
```

```
colorFrame[index + BlueIndex] = 0;
colorFrame[index + GreenIndex] = 255;
colorFrame[index + RedIndex] = 255;
```

En esta sección se encarga de pintar de color amarillo al o a los jugadores que estén muy lejos del sensor, es decir, que se encuentren a una distancia superior a 2000 , se les pinta de amarillo , para que se los pueda diferenciar de los objetos que se pintan de rojo al estar a distancias mayores de 2000 con respecto al sensor.

```
}

depthIndex += 2;
```

La variable "depthIndex " va a ser siempre 2 para poder emplear la configuración de colores Azul ,Verde y Rojo(correspondientes a sus índices 0,1,2) ,ya que si solo se pone 0 los objeto que se encuentren dentro del campo de visión de las cámaras infrarrojas va a tornarse de color Azul, si se pone 1 los objeto que se encuentren dentro del campo de visión de las cámaras infrarrojas va a tornarse de color Azul y Verde la mezcla de los 2 colores , si se pone 2 los objeto que se encuentren dentro del campo de visión de las cámaras infrarrojas va a tornarse de color Azul ,Verde , Rojo la mezcla de los 3 colores , los cuales van a ser mostrados en el objeto Image.

```
    }
  }
  return colorFrame;
```

Retorna la variable **"colorFrame",** que contiene los colores capturados del entorno del campo de visión de las cámaras infrarrojas.

```
}
private static int GetPlayerIndex(byte firstFrame)
{
  return (int)firstFrame & 7;
```

Se crea un método llamado **"GetPlayerIndex(byte firstFrame)"**, el cual va a retornarnos un valor correspondiente al número de jugadores que estén en el campo de visión del sensor, va a retornar 0 cuando no exista ningún jugador , 1 cuando este un primer jugador , 2 cuando este un segundo jugadores , así sucesivamente hasta ubicarse máximo 7 jugadores respectivamente y cada uno le corresponde un bit.

```
}
private int GetDistanceWithPlayerIndex(byte firstFrame, byte secondFrame)
{
  int distance = (int)(firstFrame >> 3 | secondFrame << 5);
```

```
return distance;
```

El método llamado **"`GetDistanceWithPlayerIndex`",** va a devolvernos la distancia a las que los objetos se van a encontrar con respecto al primero y a la segunda trama y lograr una compensación para los 3 primeros bytes obtenidos después del índice del jugador.

```
}

const float MaxDepthDistance = 4000; // max value returned

const float MinDepthDistance = 850; // min value returned

const float MaxDepthDistanceOffset = MaxDepthDistance - MinDepthDistance;
```

Se crean 3 constantes flotantes , la primera llamada **"`MaxDepthDistance`**", que va a contener la distancia de profundidad máxima que va a ser igual a 4000, la segunda llamada **"`MinDepthDistance` "** que va a almacenar la distancia de profundidad mínima que va a ser igual a 850 , y luego se logra calcular la compensación de la distancia de profundidad a través de la resta de **"`MaxDepthDistance`"** menos **"`MinDepthDistance` ",** para obtener una mayor precisión en la distancia de profundidad.

```
private void Window_Closed(object sender, EventArgs e)
```

private void Window_Closed.- Evento creado para finalizar el uso del sensor Kinect.

```
{

nui.Uninitialize();
```

***nui.Uninitialize*.-** Es una función que se encarga de cerrar todos los eventos, métodos, objetos del sensor Kinect.

```
}

}

}
```

3.2. Diseño de la Aplicación Demostrativa

3.3. Diseño Software

3.3.1. Diseño de la Aplicación Demostrativa en C#

La aplicación demostrativa que se creó utilizando los SDK de Kinect mediante Led's indicadores, fue el diseño y control de una Pinza robótica mediante la

aplicación de los estudios y análisis de los SDK de Kinect y por medio de los Led's indicadores que van a mostrar en que rango de grados de 0º a 180º se encuentra el brazo y un último Led indicador que se va a encender o apagar cuando se produzca la apertura o cierre de la Pinza robótica. La Pinza robótica consta con 3 grados de libertad ya que posee 3 servomotores, los mismos que van a permitir controlar los movimientos tanto en forma horizontal cuando el usuario de el barrido horizontal de la mano derecha (eje x), el movimiento vertical de la Pinza robótica, cuando el usuario abra y cierre el brazo derecho (eje y), y la apertura y cierre de la Pinza robótica cuando el usuario abra o cierre la mano (eje z).

La interfaz de la aplicación demostrativa se creó en Visual Studio 2010 en el lenguaje de programación C# , en los formularios WPF, en donde el usuario va a poder tener el control de ajustar el ángulo del motor del sensor Kinect y el movimiento de la Pinza robótica en las distintas ventanas que se le presentaran.

Los datos que se adquieren de los distintos movimientos que realice el usuario al mover la Pinza robótica por medio del sensor Kinect, van a ser programados en el lenguaje C# y van a ser enviados por medio de la comunicación serial a la placa Arduino, la misma que se va a encargar de recibir esos datos y enviarlos a los distintos servomotores y Led's indicadores con los que consta la aplicación demostrativa, en este caso, la Pinza robótica. La placa Arduino consta con conexión USB, lo que facilita conectar a cualquier dispositivo que tenga conexión vía USB, en nuestro caso, es conectada la placa al puerto USB en la Laptop que se creó el proyecto.

3.3.1.1. Formulario WPF Caratula

3.3.1.1.1. Análisis del formulario WPF Caratula

En este formulario esta básicamente destinado a la presentación del trabajo practico , es decir, El análisis de los códigos fuentes SDK de Kinect y diseño de una aplicación demostrativa, donde se menciona el tema antes señalado, los integrantes que desarrollaron este trabajo ,el nombre del tutor de tesis , entre otras cosas, además va a contar un botón llamado “Empezar Aplicación “ para que promedio de este el usuario de un clic y el formulario “Caratula “se va a ocultar y va a aparecer el formulario “Tesis_Kinect_Aplicacion” para empezar a utilizar la aplicación demostrativa .

3.3.1.1.2. Diseño del formulario WPF Caratula

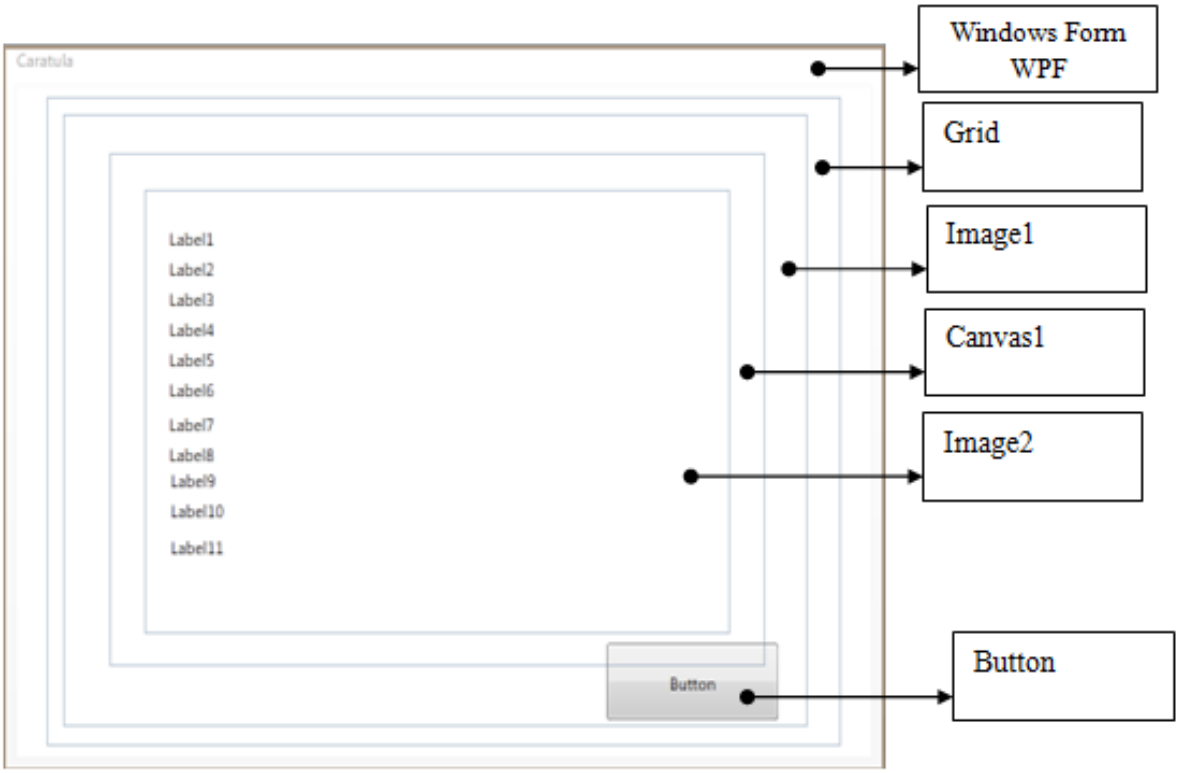


FIGURA 36 Diseño del formulario WPF Caratula[36]

En la siguiente tabla se van a especificar los objetos ó elementos utilizados del lenguaje de programación C# para el desarrollo del Software de la Pinza Robótica, los nombres que se les asignaron para su programación y sus respectivas descripciones para las que fueron empleados.

Nombre del Objeto	Nombre Asignado al Objeto	Descripción
Windows Form WPF	Caratula	Es una ventana WPF, en esta ventana se va a proceder a insertar los distintos objetos Grid para poder realizar la

[36] FIGURA 36 Tomada de Aplicación Demostrativa, Autores Tesis

		presentación grafica de la misma.
Grid	Grid	Este elemento siempre está por defecto al abrir o crear ventanas tipo WPF, este panel permite insertar todos los objetos ya sean estos “Image”, “Canavas”, “Ellipse” , etc., para poder realizar la interfaz grafica , este Grid no se le nombra de ninguna manera , debido a que en esta ventana WPF solo se ocupa un Grid y no hay la necesidad de asignarle algún nombre.
Image1	Fondo_1	Este objeto “Image, va a permitir ingresar una imagen de formato “.bmp” , que será colocada como un primer fondo de pantalla en el formulario “Caratula”
Canvas1	C_Tras_Fondo	Sirve como un panel, donde podemos ingresas distintos

		objetos, en este caso se le da la función de brindar un efecto de contraste, para tener un mayor impacto visual en la implementación en el formulario “Caratula”.
Image2	Fondo_2	Se le va a cargar una imagen de formato “.bmp”, que va a ser implementada como un segundo fondo en nuestro formulario WPF.
Button	Bt_Inicio	Este objeto va a estar programado para cuando el usuario con el cursor de un click, se cierre la venta actual y llame al segundo formulario WPF “Tesis_Kinect_Aplicacion”.
Label1	L_Un	Va a mostrar el nombre de la Universidad Politécnica Salesiana
Label2	L_Cmps	Va a mostrar el Campus correspondiente donde se realizó la Tesis

Label3	L_Carrera	Va a mostrar el nombre de la carrera de Ingeniería Electrónica
Label4	L_Mencion	Va a mostrar el nombre de la mención , que los desarrolladores de esta aplicación siguen , que es Sistemas Industriales
Label5	L_Tuno	Va a contener la primera parte del título de la tesis en cuestión
Label6	L_TDos	Va a contener la segunda parte del título de la tesis en cuestión
Label7	L_TTres	Va a contener la tercera parte del título de la tesis en cuestión
Label8	L_Tesis	Va a desplegar el título que se va a lograr obtener una vez finalizado el proyecto

Label9	L_NUno	Va a mostrar el nombre del primer integrante que conforma el desarrollo de la Tesis.
Label10	L_NDos	Va a mostrar el nombre del segundo integrante que conforma el desarrollo de la Tesis.

Tabla 2 Nombres, asignaciones y descripción de los objetos empleados en el diseño del formulario carátula

3.3.1.1.3. Resultado del formulario WPF Caratula

FIGURA 37 Formulario Caratula resultado final[37]

[37] FIGURA 37 Tomada de Aplicación Demostrativa, Autores Tesis

3.3.1.2. Formulario WPF Tesis_Kinect_Apliacacion

3.3.1.2.1. Análisis del formulario WPF Tesis_Kinect_Aplicacion

En este formulario se va a diseñar toda la interfaz grafica de la aplicación demostrativa, va a contener 3 elementos Grid, que son paneles que permiten crear interfaces tanto básicas como complejas, permitiendo colocar varios elementos en distintos Grid, el Grid llamado “grid1”va a ser el Grid principal dentro del formulario WPF que va a contener a los Grid restantes llamados “gr1” y “gr2”.

El Grid “gr1”, se desempeña a manera de una ventana , en donde se va a proceder a ajustar el motor de la cámara , a las distintas estaturas de los diversos usuarios que puedan emplear esta aplicación , si así el usuario lo desea, mediante los botones de “Ajustar ” ó “Predeterminado” también va a contar con etiquetas o “Label” en donde se van a mostrar mensajes para indicar al usuario la distancia correcta que debe de tener hacia el sensor , y por último se incluye un botón “siguiente”, el que va a permitir habilitar el siguiente Grid “gr2”

El Grid “gr2”,va a contener en si, el diseño de la interfaz grafica de la aplicación demostrativa , en donde va a contar con 2 objetos “Image” , en los cuales se va a desplegar las distintas utilidades que contiene el SDK de Kinect, en el primer objeto Image “im_SC” se va a desplegar en conjunto el Skeletal Tracking con la cámara de video RGB , en el segundo objeto Image “im_C” se va a desplegar las imágenes que capture la cámara RGB del sensor Kinect y se va a contar con un objeto Canvas “Canvas 11”donde se va a mostrar cómo funciona el Skeletal Tracking del SDK de Kinect , además van a ver objetos elipses que van a estar enlazados a los “Joint” o puntos del esqueleto humano que Kinect reconoce, para que estos objetos se muevan a través del empleo de Skeletal Tracking y además se cuenta con objetos Elipses

adicionales que van a simular el encendido de los Led's indicadores cada vez que se encuentren los servomotores en determinados grados entre el rango de 0º a 180º .

3.3.1.2.2. Diseño del formulario WPF Tesis_Kinect_Aplicacion

El diseño del formulario "Tesis_Kinect_Aplicacion", está realizado en 2 Grid , el primer Grid llamado "gr1" , y el segundo Grid llamado "gr2",que va a contener en si las distintas funciones que tiene el sensor Kinect y en donde el usuario va a poder controlar al Brazo robótico mediante los movimientos del cuerpo humano , específicamente el movimiento de sus extremidades superiores como la apertura y cierre de brazos, barrido horizontal de la mano y apertura y cierre de la mano, a continuación se detalla el diseño de cada Grid .

3.3.1.2.2.1. Diseño GRID "gr1" del formulario WPF Tesis_Kinect_Aplicacion

Este Grid como se menciona anteriormente se va a encargar en general de ajustar el motor de la cámara y determinar la correcta distancia que debe de tener el usuario hacia el sensor mediante un cuadro de mensaje conformados por "Label" que van a estar programados y mostraran los distintos contenidos de acorde a la distancia, el usuario va a poder ajustar el Angulo del motor del sensor Kinect mediante un objeto "Slider" en el rango de -27º a 27º y además va a existir 3 botones llamados , "Ajuste" que va a proceder a colocar el motor del sensor Kinect según el ángulo seleccionado por el usuario , un botón "Predeterminado", que se va a emplear para posicionar el motor de la cámara en el ángulo 0º y un botón "Siguiente", que va a cumplir la función de cerrar el Grid y llamar al segundo Grid llamado"gr2" en donde el usuario va a empezar la aplicación.

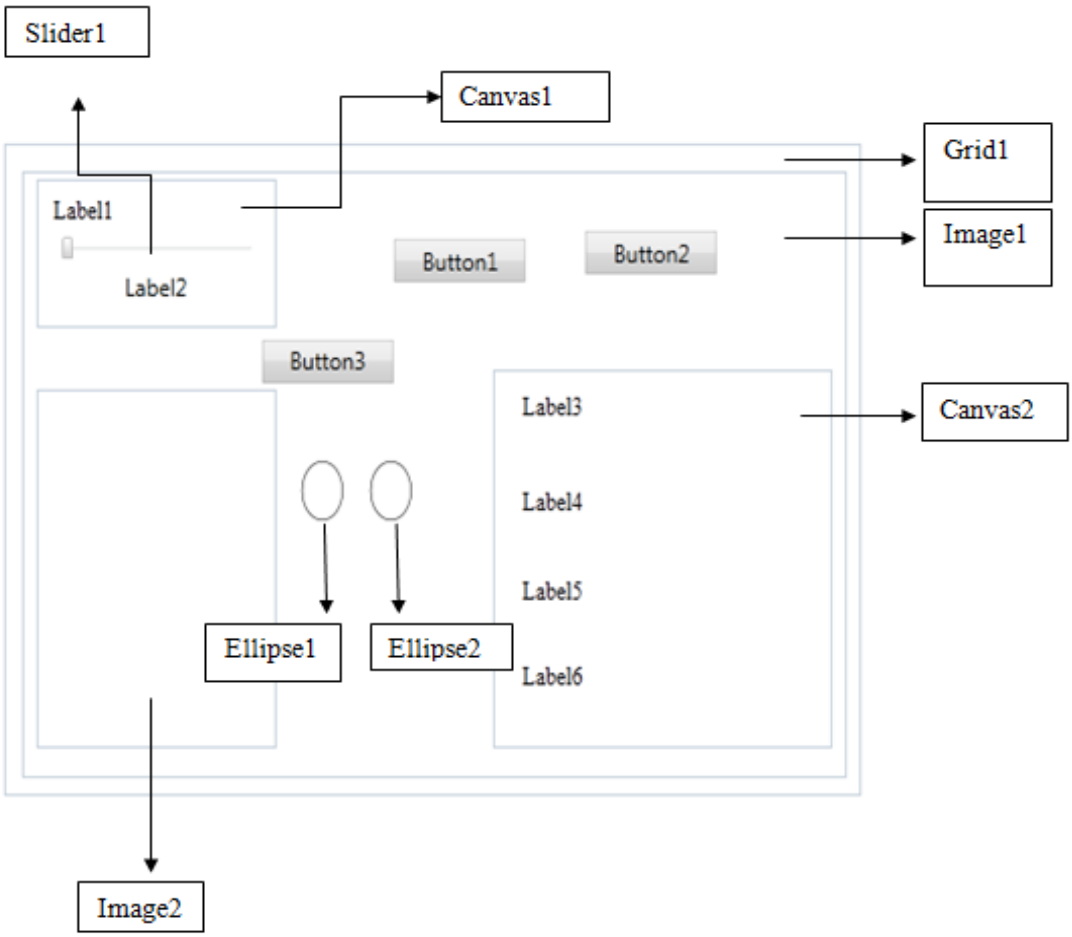


FIGURA 38 Diseño del formulario WPF Grid "gr1"[38]

Nombre del Objeto	Nombre Asignado al Objeto	Descripción
Grid1	gr1	Se va a proceder a insertar los distintos objetos para poder realizar nuestra primera ventana, donde el usuario va a proceder a ajustar el ángulo del motor de la cámara del sensor Kinect

[38] FIGURA 38 Tomada de Aplicación Demostrativa, Autores Tesis

Image1	Fondo_1	Va a permitir ingresar una imagen de formato “.bmp” , que será colocada como fondo de pantalla en el formulario “Tesis_Kinect_Aplicacion”
Image2	im_camaraKinect	Se van a enlazar los datos de la cámara RGB del sensor Kinect al objeto “Image”, para que se desplieguen los datos captados por la cámara, en dicho objeto.
Slider1	Slider 1	Va a permitir seleccionar cuantos grados deseamos que se mueva el motor de la cámara del sensor Kinect entre -27 a 27 grados (º), mediante el movimiento de la mano derecha al situarse en la misma posición del objeto, así poder desplazar la barra del slider y ubicar el ángulo que le parezca mas conveniente al usuario al momento de utilizar la aplicación.

Canvas1	Canvas1	Va a ser utilizado como un panel, donde servirá para colocar los objeto “Slider1” y “Label1”
Canavas2	Canavas2	Se va a proceder a colocar los distintos objetos “Label”.
Ellipse1	E_Mizq	Se van a, enlazar el Joint o punto de la mano izquierda correspondiente del skeletalTracking con la mano izquierda del usuario , y cuyos datos serán almacenados en el objeto “E_Mizq”, para que posteriormente ,el usuario al mover su mano izquierda al mismo tiempo va a mover a dicho objeto.
Ellipse2	E_Mder	Se van a, enlazar el Joint o punto de la mano derecha correspondiente del Skeletal Tracking con la mano

		derecha del usuario, y cuyos datos serán almacenados en el objeto “E_Mder”, para que posteriormente, el usuario al mover su mano derecha ,al mismo tiempo va a mover al objeto enlazado a ese movimiento.
Label1	lb_AjusteCamara	Va a estar colocado en el objeto canvas denominado “Canavas1”, este Label llamado “lb_AjusteCamara”va a mostrar de título: Ajuste el motor de la cámara.
Label2	lb_Grados	Va a mostrar al usuario, cuantos grados está moviendo el motor de la cámara del sensor Kinect.
Label3	lb_mensaje	va a contener la palabra mensaje , la cual será mostrado al momento de la aplicación

Label4	lb_mostrar_mensaje	Va a contener un texto a manera informativa, para que el usuario se ubique en la posición correcta hacia el sensor.
Label5	lb_MCalculo_Distancia	Va a ser programado, para que acorde a la distancia que se encuentre el usuario, en ese Label se muestre un mensaje en donde le informe al usuario si está muy cerca , si esta correcta o si está alejada su posición con respecto al sensor.
Label6	lb_mDistancia	Se le va a programar, para que le muestre al usuario un mensaje de texto ,indicando que de un paso para a tras si se encuentra muy cerca del sensor , que su posición es la correcta , o que dé un paso al frente si está alejado en el sensor.

Button1	bt_ajustar	El usuario al momento de ubicar su mano derecha sobre el botón, este procederá a ajustar el motor de la cámara acorde al ángulo seleccionado.
Button2	bt_defecto	al momento que el usuario lo seleccione, va a colocar al motor de la cámara del sensor Kinect en una posición por defecto , en este caso , 0 grados (º).
Button3	bt_siguiente	Va ser programado para que cuando sea seleccionado, deshabilite y oculta al “gr1”, y sea llamado el segundo Grid “gr2” para que sea mostrado, habilitado y el usuario empiece a utilizar el segundo grid “gr2”.

Tabla 3 Nombres, asignaciones y descripción de los objetos empleados en el diseño del Grid “gr1”

3.3.1.2.2.1.1. Resultado Grid “gr1”

FIGURA 39 Resultado final del diseño del Grid “gr1”[39]

3.3.1.2.2.2. Diseño Grid “gr2” del formulario WPF Tesis_Kinect_Aplicacion

El Grid llamado “gr2”, va a ser la parte esencial de la aplicación demostrativa, ya que en este Grid se va a programar y se va a enlazar los distintos datos que se va a obtener del sensor Kinect con el lenguaje de programación C# , para posteriormente enviarlos por medio del protocolo serial hacia la placa Arduino, en donde están conectados los distintos elementos electrónicos para hacer mover al brazo robótico y encender los Led’s indicadores.

[39] FIGURA 39 Tomada de Aplicación Demostrativa, Autores Tesis

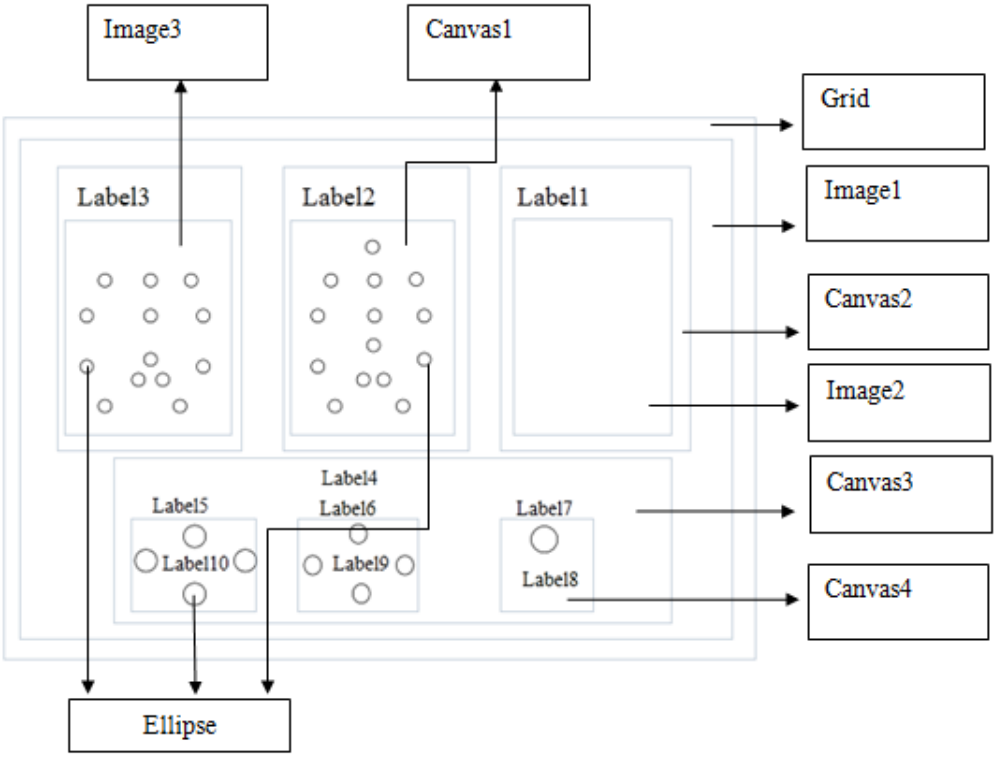


FIGURA 40 Diseño del Grid "gr2"[40]

Nombre del Objeto	Nombre Asignado al Objeto	Descripción
Grid1	gr2	En este elemento se va a proceder a insertar los distintos objetos para poder realizar una ventana gráfica, donde el usuario va a controlar al brazo robótico mediante el movimiento de sus extremidades superiores.

[40] FIGURA 40 Tomada de Aplicación Demostrativa, Autores Tesis

Image1	Fondo_2	Va a permitir ingresar una imagen de formato “.bmp” , que será colocada como fondo de pantalla en el Grid “gr2”
Image2	im_C	Se van a enlazar los datos de la cámara RGB del sensor Kinect al objeto “Image”, para que se despejen los datos captados por la cámara, en dicho objeto.
Image3	im_SC	Se van a enlazar los datos de la cámara RGB del sensor Kinect al objeto “Image”, para que se desplieguen los datos captados por la cámara, en dicho objeto, además se va a colocar objetos “ellipses” para que cuando el usuario realice un movimiento en el objeto “image” , también se muevan los objetos “Ellipses” en sincronía con los

		movimientos captados en el objeto “”Image
Canvas1	Can_1	Este objeto va a servir como un panel, donde se va a insertar los objetos llamados “im_SC”, objetos Ellipses y “Can_SC”, ademas de su respectiva etiqueta Label llamada “lb_SkeCam”.
Canvas2	Can_2	Este objeto va a servir como un panel, donde se va a insertar el objeto llamado “Canvas_Skeletal”, objetos Ellipses, además de su respectiva etiqueta Label llamada “lb_Skeleton”.
Canvas3	Can_3	Este objeto va a servir como un panel, donde se va a insertar los objetos llamados “Can_C” y”im_C”, además de su respectiva etiqueta Label llamada “lb_Camara”.
Canvas4	Can_4	Este objeto va a servir como un panel ,donde se va a insertar los distintos objetos

		Canvas , Etiquetas Label y objetos Ellipses ,
Canvas5	Can_SC	Servirá como un panal en donde se colocaran objetos “ellipses”, que van a estar enlazados con los distintos puntos del Esqueleto Humano, los cuales se van a desplazar solamente dentro de este objeto Canvas.
Canvas6	Canvas_Skeletal	Servirá como un panel en donde se colocaran objetos “ellipses”, que van a estar enlazados con los distintos puntos del Esqueleto Humano, los cuales se van a desplazar solamente dentro de este objeto Canvas.
Canvas7	Can_C	Servirá como un panel en donde se va a colocar el objeto Image llamado “im_C”
Canvas8	Can_H	Va a ser empleado como un panel de fondo de pantalla para dar un mejor contraste a la aplicación Grafica, además

		de contener los objetos Ellipses y etiquetas Label
Canvas9	Can_V	Va a ser empleado como un panel de fondo de pantalla para dar un mejor contraste a la aplicación Grafica además de contener los objetos Ellipses y etiquetas Label
Canvas10	Can_Pinza	Va a ser empleado como un panel de fondo de pantalla para dar un mejor contraste a la aplicación Grafica además de contener los objetos Ellipses y etiquetas Label
Label1	lb_SkeCam	Sera empleado a manera de titulo que va a mostrar en su contenido “Skeletal Tracking en conjunto con la Cámara del sensor Kinect”
Label2	lb_Skeleton	Sera empleado a manera de titulo que va a mostrar en su contenido “Skeletal Tracking del sensor Kinect”
Label3	lb_Camara	Sera empleado a manera de titulo que va a mostrar en su contenido “Cámara del

		sensor Kinect”
Label4	lb_titulo_Leds	Sera empleado a manera de titulo que va a mostrar en su contenido “Led’s Indicadores de Grados”
Label5	lb_Barrido_Hori_Mano	Sera empleado a manera de titulo que va a mostrar en su contenido “Grados desplazamiento horizontal”
Label6	lb_Ap_Cie_Brazo	Sera empleado a manera de titulo que va a mostrar en su contenido “Grados desplazamiento vertical”
Label7	lb_Ap_Cie_Pinza	Sera empleado a manera de titulo que va a mostrar en su contenido “Apertura o Cierre de la Pinza ”
Label8	lb_Gra_V	Este objeto Label va a desplegar ó va a mostrar al usuario cuantos grados en el eje Y se esta movimiento ,este valor va a estar comprendido entre 0-180 Grados
Label9	lb_Gra_H	Este objeto Label va a desplegar ó va a mostrar al

		usuario cuantos grados en el eje X se está movimiento ,este valor va a estar comprendido entre 0-180 Grados
Label10	Lb_Gra_Pinza	Va a mostrar un mensaje indicador , para saber si la pinza esta “Abierta” ó “Cerrada”
Ellipse1	e_ei1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joint`s “JointID. ShoulderLeft” , que representa al hombro izquierdo del cuerpo humano
Ellipse2	e_ei2	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joint`s “JointID.ElbowLeft” , que representa al codo izquierdo del cuerpo humano
Ellipse3	e_ei3	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joint`s

		“JointID.WristLeft” , que representa a la muñeca izquierda del ser humano
Ellipse4	e_ei4	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joint`s “JointID.HandLeft” , que representa a la mano izquierda del cuerpo humano
Ellipse5	e_e1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joint`s “JointID.ShoulderCenter” , que representa al tórax del cuerpo humano
Ellipse6	e_ec1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joint`s “JointID.Spine” , que representa a la espina del cuerpo humano
Ellipse7	e_ec2	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y

		va a estar enlazado al Joint`s "JointID.HipCenter" , que representa a la pelvis del cuerpo humano
Ellipse8	e_eci	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Can_SC" y va a estar enlazado al Joint`s "JointID.HipLeft" , que representa al punto de la cadera izquierda del cuerpo humano
Ellipse9	e_ecd	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Can_SC" y va a estar enlazado al Joint`s "JointID.HipRight" , que representa al punto de la cadera derecha del cuerpo humano
Ellipse10	e_ed1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Can_SC" y va a estar enlazado al Joint`s "JointID.ShoulderRight" , que representa al hombro derecho del cuerpo humano

Ellipse11	e_ed2	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joints “JointID.ElbowRight” , que representa al codo derecho del cuerpo humano
Ellipse12	e_ed3	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joints “JointID.WristRight” , que representa a la muñeca derecha del cuerpo humano
Ellipse13	e_ed4	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Can_SC” y va a estar enlazado al Joints “JointID.HandRight” , que representa a la mano derecha del cuerpo humano
Ellipse14	e_cabeza	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Canvas_Skeletal” y va a estar enlazado al Joint “JointID.Head” , que

		representa a la cabeza del cuerpo humano
Ellipse15	e_pi1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID. ShoulderLeft" , que representa al hombro izquierdo del cuerpo humano
Ellipse16	e_pi2	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.ElbowLeft" , que representa al codo izquierdo del cuerpo humano
Ellipse17	e_pi3	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.WristLeft" , que representa a la muñeca izquierda del ser humano

Ellipse18	e_pi4	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.HandLeft" , que representa a la mano izquierda del cuerpo humano
Ellipse19	e_p1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.ShoulderCenter" , que representa al tórax del cuerpo humano
Ellipse20	e_pc1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.Spine" , que representa a la espina del cuerpo humano
Ellipse21	e_pc2	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado

		“Canvas_Skeletal” y va a estar enlazado al Joint “JointID.HipCenter” , que representa a la pelvis del cuerpo humano
Ellipse22	e_pc3	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Canvas_Skeletal” y va a estar enlazado al Joint “JointID.HipLeft” , que representa al punto de la cadera izquierda del cuerpo humano
Ellipse23	e_pc4	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Canvas_Skeletal” y va a estar enlazado al Joint “JointID.HipRight” , que representa al punto de la cadera derecha del cuerpo humano
Ellipse24	e_pd1	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado “Canvas_Skeletal” y va a

		estar enlazado al Joint "JointID.ShoulderRight" , que representa al hombro derecho del cuerpo humano
Ellipse25	e_pd2	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.ElbowRight" , que representa al codo derecho del cuerpo humano
Ellipse26	e_pd3	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.WristRight" , que representa a la muñeca derecha del cuerpo humano
Ellipse27	e_pd4	Este objeto Ellipse va estar colocado dentro del objeto Canvas llamado "Canvas_Skeletal" y va a estar enlazado al Joint "JointID.HandRight" , que representa a la mano derecha

		del cuerpo humano
Ellipse28	e_dh_45	Este objeto Ellipse se va a pintar de un color determinado cuando los valores obtenidos en el eje X, estén comprendidos entre 0-45 Grados
Ellipse29	e_dh_90	Este objeto Ellipse se va a pintar de un color determinado cuando los valores obtenidos en el eje X, estén comprendidos entre 46-90 Grados
Ellipse30	e_dh_135	Este objeto Ellipse se va a pintar de un color determinado cuando los valores obtenidos en el eje X, estén comprendidos entre 91-135 Grados
Ellipse31	e_dh_180	Este objeto Ellipse se va a pintar de un color determinado cuando los valores obtenidos en el eje X, estén comprendidos entre 136-180 Grados
Ellipse32	e_dv_45	Este objeto Ellipse se va a

		pintar de un color determinado cuando los valores obtenidos en el eje Y, estén comprendidos entre 0-45 Grados
Ellipse33	e_dv_90	Este objeto Ellipse se va a pintar de un color determinado cuando los valores obtenidos en el eje Y, estén comprendidos entre 46-90 Grados
Ellipse34	e_dv_135	Este objeto Ellipse se va a pintar de un color determinado cuando los valores obtenidos en el eje Y, estén comprendidos entre 91-135 Grados
Ellipse35	e_dv_180	Este objeto Ellipse se va a pintar de un color determinado cuando los valores obtenidos en el eje Y, estén comprendidos entre 136-180 Grados
Ellipse36	e_pinza	Este objeto Ellipse se va a pintar de un color determinado cuando los

		valores obtenidos de la distancia en el eje Z de la mano izquierda, estén comprendidos entre 1.8m-2m

Tabla 4 Nombres, asignaciones y descripción de los objetos empleados en el diseño del Grid "gr2"

3.3.1.2.2.2.1. Resultado Grid "gr2"

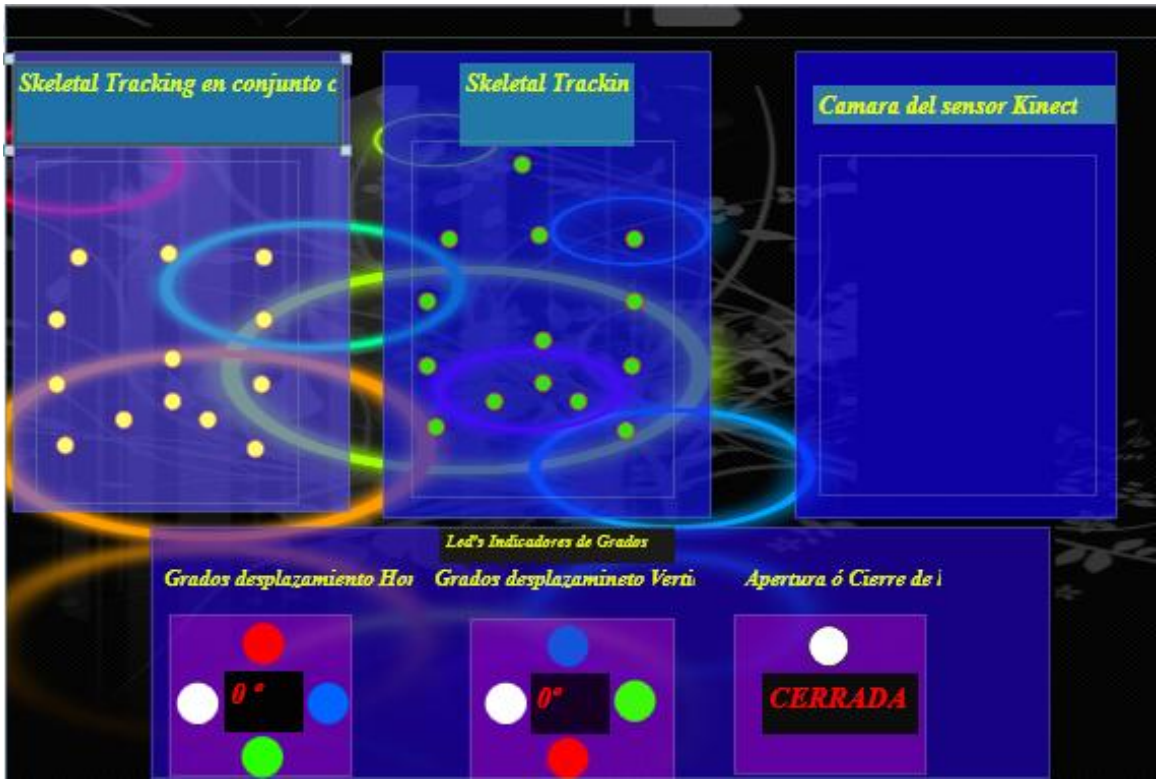


FIGURA 41 Resultado final Grid "gr2" [41]

3.3.2. Programación C#

Los presentes programas fueron desarrollados en el lenguaje de programación C#, de Microsoft Visual Studio 2010

3.3.2.1. Codificación formulario WPF Caratula

```
using System;

using System.Collections.Generic;
```

[41] FIGURA 41 Tomada de Aplicación Demostrativa, Autores Tesis

```
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace Tesis_Kinect
{
   /// <summary>
   /// Lógica de interacción para MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      // Se crea el evento  Conf_Tamano_Elementos() , en el cual se va a          establecer el ancho y el alto
de los distintos elementos
      // empleados en el diseño  del formulario "Caratula"
      public void Conf_Tamano_Elementos()
      {
         // Se establece el tamaño del formulario "Caratula", Ancho= 680 y Alto= 480
         this.Height = 480;
         this.Width = 680;
         // Se establece el tamaño del Objeto Image "Fondo_1", Ancho= 658 y Alto= 441
         Fondo_1.Width = 658;
         Fondo_1.Height = 441;
         // Se establece el tamaño del Objeto Image "Fondo_2", Ancho= 568 y Alto= 342
         Fondo_2.Height = 342;
         Fondo_2.Width = 568;
         // Se establece el tamaño del Objeto Button "Bt_Inico", Ancho= Se ajusta de manera automática y
Alto= 40
         Bt_Inicio.Height = 40;
         // Se establece el tamaño de letra número 12 del Objeto Button "Bt_Inico".
         Bt_Inicio.FontSize = 12;
      }
      //Se crea un método donde se establecen los diferentes títulos que van a contener los objetos "Label".
      public void Etiquetas()
      {
         L_Un.Content = "UNIVERSIDAD POLITECNICA SALESIANA";
         L_Cmps.Content = "SEDE QUITO – CAMPUS SUR";
         L_Carrera.Content = "CARRERA DE INGENIERÍA ELECTRÓNICA";
         L_Mencion.Content = "MENCIÓN SISTEMAS INDUSTRIALES";
         L_TUno.Content = "APLICACIÓN DEMOSTRATIVA QUE CAPTE LOS MOVIMIENTOS ";
         L_TDos.Content = "DE BRAZOS Y MANOS DEL CUERPO HUMANO";
         L_TTres.Content = "MEDIANTE EL SENSOR KINECT";
         L_Tesis.Content = "TESIS PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO
ELECTRÓNICO";
         L_NUno.Content = "LUIS ALEJANDRO HERNÁNDEZ TOALA";
         L_NDos.Content = "JUAN DAVID HERRERA RODRÍGUEZ ";
         L_Dir.Content = "DIRECTOR: Msc. Vinicio Tapia ";
         Bt_Inicio.Content = "CLICK Para Empezar Aplicación";

      }

      //Se crea un Evento, en el cual se va a proceder a hacer un llamado a las distintas imágenes que serán
cargadas a los
      // Distintos objetos Image
      public void cargar_Imagenes_fondos()
      {
         // Se crea una variable , que va a ser tipo BitmapImage llamada "imagen1", "imagen2".
         BitmapImage imagen1;
```

```
            BitmapImage imagen2;
            // Se inicializa una nueva instancia de la clase BitmapImage().
            imagen1 = new BitmapImage();
            imagen2 = new BitmapImage();
// Señala el inicio de la inicialización de BitmapImage con la función BeginInit().
            imagen1.BeginInit();
            imagen2.BeginInit();
            // Se especifica la ruta de donde se encuentra la Imagen de formato .bmp que va
            // a ser cargada en el objeto Image.
            imagen1.UriSource = new Uri(@"/Images/1.bmp", UriKind.Relative);
            imagen2.UriSource = new Uri(@"/Images/2.bmp", UriKind.Relative);
            // Señala la finalización de la inicialización de BitmapImage con la función EndInit().
            imagen1.EndInit();
            imagen2.EndInit();
            // Se le carga al objeto Image "Fondo_1", la variable Bitmap "imagen1" que contiene
            // la Imagen de formato .bmp para ser mostrada en el formulario "Caratula".
            Fondo_1.Source = imagen1;
            Fondo_2.Source = imagen2;
        }
        public MainWindow()
        {
            InitializeComponent();
            Conf_Tamano_Elementos();
            cargar_Imagenes_fondos();
            Etiquetas();

        }

        private void Bt_Inicio_Click(object sender, RoutedEventArgs e)
        {
            // Se inicializa una nueva instancia para que sea llamado al siguiente formulario
"Tesis_Kinect_Aplicacion", a través de
            // la variable "Aplicacion".
            Tesis_Kinect_Aplicacion Aplicacion = new Tesis_Kinect_Aplicacion();
            //La variable "Aplicacion", hace el llamado del formulario "Tesis_Kinect_Aplicacion", mediante la
función
            // Show() , que muestra el siguiente formulario al que la variable este enlazada.
            Aplicacion.Show();
            //La función Close() , permite cerrar el formulario actual en el cual se esta trabajando , en este caso
            //el formulario "Caratula".
            this.Close();
        }
    }
}
```

3.3.2.2. Codificación formulario WPF Tesis_Kinect_Aplicacion

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
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.Shapes;
// using Microsoft.Research.Kinect.Nui; es la librería correspondiente del sensor Kinect que va a permitir
// ejecutar todas sus funciones y programarlo
using Microsoft.Research.Kinect.Nui;
```

```
// using Coding4Fun.Kinect.Wpf; es la librería correspondiente para poder utilizar las extensiones o
métodos
// de  Coding4Fun.
using Coding4Fun.Kinect.Wpf;
// using System.IO.Ports; es la libreria correspondiente, que permite inicializar o habilitar
// al puerto serial para que pueda ser utilizado.
using System.IO.Ports;

namespace Tesis_Kinect
{
   /// <summary>
   /// Lógica de interacción para Tesis_Kinect_Aplicacion.xaml
   /// </summary>
   ///


   public partial class Tesis_Kinect_Aplicacion : Window
   {
      // Se inicializa al sensor Kinect con la función Runtime  a  través de la variable "S_Kinect".
      public Runtime S_kinect;
      // Se inicializa al puerto serial, a través de la variable "puerto".
      SerialPort puerto;

      public int i;
      public int j;
      //Se crean variables que van a ser utilizadas para determinar la escala en Grados de 0 a 180
      //en los ejes tanto en X como en Y
     //Variables empleadas para la extremidad superior derecha
      private int eje_x = 0;
      private int eje_y = 0;
      private int eje_Z = 0;
      private int eje_xs = 0;
      //Variables empleadas para la extremidad superior izquierda
      private int eje_xi = 0;
      private int eje_yi = 0;
      private int eje_Zi = 0;
      private int eje_xis = 0;
      private int eje_yis = 0;
      private int eje_Zis = 0;

      // Se crea el evento  Conf_Tamano_Elementos() , en el cual se va a establecer el ancho y el alto de los
distintos elementos
      // empleados en el diseño de las distintas ventanas del software de la aplicación demostrativa.
      public void Conf_Tamanos_Elementos_Motor() {
         // Se establece el tamaño del Objeto Grid "gr1", Ancho= 946 y Alto= 640.
         gr1.Height = 640;
         gr1.Width = 946;
         // Se establece el tamaño del Objeto Image "Fondo_1", Ancho= 940 y Alto= 630.
         Fondo_1.Height = 630;
         Fondo_1.Width = 940;
         //Al objeto Slider se le colocan en los valores Máximos=27 y Mínimos =-27 , que van a ser los
grados correspondientes , que
         //el  motor de cámara del sensor Kinect podrá desplazarse en el sentido vertical.
         slider1.Maximum = 27;
         slider1.Minimum = -27;
      }
      public void Conf_Tamanos_Elementos_Aplicacion()
      {
         // Se establece el tamaño del Objeto Grid "gr2", Ancho= 948 y Alto= 640.
         gr2.Height = 640;
         gr2.Width = 946;
         // Se establece el tamaño del Objeto Image "Fondo_2", Ancho= 940 y Alto= 630.
         Fondo_2.Height = 630;
         Fondo_2.Width = 940;
```

```
            //Al objeto Slider se le colocan en los valores Máximos=27 y Mínimos =-27, que van a ser los
grados correspondientes, que
            //el  motor de cámara del sensor Kinect podrá desplazarse en el sentido vertical.

        }

        public void cargar_Imagenes_fondos_Motor()
        {
            // Se crea una variable, que va a ser tipo BitmapImage llamada "imagen1", "imagen2".
            BitmapImage imagen1;
            // Se inicializa una nueva instancia de la clase BitmapImage().
            imagen1 = new BitmapImage();
            // Señala el inicio de la inicialización de BitmapImage con la función BeginInit().
            imagen1.BeginInit();
            // Se especifica la ruta de donde se encuentra la Imagen de formato .bmp que va
            // Ser cargada en el objeto Image.
            imagen1.UriSource = new Uri(@"/Images/3.bmp", UriKind.Relative);
            // Señala la finalización de la inicialización de BitmapImage con la función EndInit().
            imagen1.EndInit();
            // Se le carga al objeto Image "Fondo_1", la variable Bitmap "imagen1" que contiene
            // La Imagen de formato .bmp para ser mostrada en el formulario "Caratula".
            Fondo_1.Source = imagen1;
        }
        //Se crea un método donde se establecen los diferentes títulos que van a contener los objetos "Label".
        public void Etiquetas_Motor() {
            lb_AjusteCamara.Content = "Ajuste del Motor de la Cámara ";
            lb_mostrar_mensaje.Content = "¡Atención! Antes de empezar sitúate en la \r distancia correcta hacia
el sensor ...";
        }
        //Se crea un método donde se establecen los diferentes títulos que van a contener los objetos "Label"
en el
        //objeto Grid llamado "gr2".
        public void Etiquetas_Apli_Kinect()
        {
            lb_SkeCam.Content = "Skeletal Tracking en conjunto \r con la Cámara del sensor Kinect";
            lb_Skeleton.Content = "Skeletal Tracking\rdel sensor Kinect";
            lb_Camara.Content = "Cámara del sensor Kinect";
            lb_Barrido_Hori_Mano.Content = "Grados desplazamiento\r       Horizontal ";
            lb_Ap_Cie_Brazo.Content = "Grados desplazamiento\r      Vertical";
            lb_Ap_Cie_Pinza.Content = "Apertura ó Cierre\r       de la Pinza";
        }
        //Se crea un método para proceder a limpiar el contenido de los label's al iniciar el programa.
        public void Limpiar_Motor()
        {
            lb_MCalculo_Distancia.Content = " ";
            lb_mDistancia.Content = " ";
        }
        public Tesis_Kinect_Aplicacion()
        {
            InitializeComponent();
            //se ejecuta el método Etiquetas_Apli_Kinect(); al momento que se inicializa al formulario WPF.
            Etiquetas_Apli_Kinect();
            //Se especifica el ancho =946 y el largo=640 de la venta WPF Tesis_Kinect_Aplicacion  llamada
"Aplicación Kinect".
            Aplicacion_Kinect.Height = 640;
            Aplicacion_Kinect.Width = 946;
            //Se hace al Grid "gr2" invisible a través de la propiedad Visibility y la desabilitamos a través de la
propiedad Enable =False.
            gr2.Visibility = System.Windows.Visibility.Collapsed;
            gr2.IsEnabled = false;
            // Se especifica cuantos Sensores Kinect son utilizados o están conectados a través de la función
            //Runtime.Kinects[0];en donde [0]= 1 sensor Kinect, [1]=2 sensores Kinect,[2]=3 sensores Kinect,
etc.
            S_kinect = Runtime.Kinects[0];
```

```
            //Se crea un evento en donde van a ser cargadas las distintas funciones del sensor Kinect
            this.Loaded += new RoutedEventHandler(Window_Loaded_Kinect);
            //Se configura al puerto serial en el COM4.
            puerto = new SerialPort("COM4", 9600, Parity.None, 8, StopBits.One);
            //Se emplea un método llamado VideoFrameReady, este método nos va a servir para obtener el
frame desde el
            //que podremos extraer el objeto tipo PlanarImage que es, con el que trabajaremos en la aplicación.
            S_kinect.VideoFrameReady += camara_video_kinect;
        }
        //Se crea un método para configurar la camara RGB de Kinect.
        void camara_video_kinect(object sender,
Microsoft.Research.Kinect.Nui.ImageFrameReadyEventArgs e)
        {
            //Al PlanarIamge se le va a utilizar para crear un BitmapSource con el método Create, en el cual se
van a especificar
            //los parámetros de anchura , altura de la imagen , el dpi, el formato de píxeles de la imagen (brg32,
brga32, blackWhite…),
            //la representación en bytes de la imágen y por último el stride.
            PlanarImage im_video = e.ImageFrame.Image;
            BitmapSource im_configuracion = BitmapSource.Create(im_video.Width, im_video.Height, 96, 96,
                                    PixelFormats.Bgr32, null, im_video.Bits, im_video.Width *
im_video.BytesPerPixel);
            //Los resultados de Configuración serán enlazados a los distintos objetos Image's ,que se han
empleado en el desarrollo de la
            //aplicación, así que cada vez que el sensor capture una imagen se activara el evento
,camara_video_kinect que obtendrá la imagen,
            //creará un BitmapSource por medio del PlanarImage y actualizará la imagen que se tiene que
mostrar en la ventana de nuestra aplicación.
            im_camaraKinect.Source = im_configuracion;
            im_SC.Source = im_configuracion;
            im_C.Source = im_configuracion;

        }
        //Se crea un método  llamado Bloqueo en el cual se va a bloquear a los objetos Slider y al botón
"bt_defecto"
        public void Bloqueo()
        {
            //Se bloque al objeto Slider1 y bt_defecto, a través de la propiedad IsEnable=False
            slider1.IsEnabled = false;
            bt_defecto.IsEnabled = false;
        }
        //Este método nos va a permitir crear las instancias para controlar al sensor Kinect e inicializaremos
con las opciones que se desee usar
        private void Window_Loaded_Kinect(object sender, RoutedEventArgs e)
        {
            //Abrimos al puerto serial con la función Open.
            puerto.Open();
            //Ejecutamos los distintos métodos creados.
            Conf_Tamanos_Elementos_Motor();
            Etiquetas_Motor();
            cargar_Imagenes_fondos_Motor();
            Limpiar_Motor();
            Bloqueo();
            Conf_Tamanos_Elementos_Aplicacion();
            //Se hace al Grid "gr2" invisible a través de la propiedad Visibility y la deshabilitamos a través de la
propiedad Enable =False.
            gr2.Visibility = System.Windows.Visibility.Collapsed;
            gr2.IsEnabled = false;
            //Inicializamos y especificamos el uso del Skeletal Tracking
            S_kinect.Initialize(Microsoft.Research.Kinect.Nui.RuntimeOptions.UseColor |
RuntimeOptions.UseSkeletalTracking);
            //Inicializamos y especificamos el uso de la cámara RGB del sensor Kinect, configuramos la
resolución ,tipo de imagen , tipo de captura de imágenes.
```

```
            S_kinect.VideoStream.Open(ImageStreamType.Video, 2, ImageResolution.Resolution640x480,
ImageType.Color);
            // Se ubica el ángulo del motor de la cámara en 0 , cada vez que se ejecute el programa.
            S_kinect.NuiCamera.ElevationAngle = 0;
            //Se emplea el evento de Codi4Fun, llamado SkeletonEngine, el cual nos permite ajustar  los
distintos parámetros
            //para realizar un poco mas preciso los datos adquiridos
            S_kinect.SkeletonEngine.TransformSmooth = true;
            TransformSmoothParameters parameters = new TransformSmoothParameters();
            parameters.Smoothing = 0.7f;
            parameters.Correction = 0.3f;
            parameters.Prediction = 0.4f;
            parameters.JitterRadius = 1.0f;
            parameters.MaxDeviationRadius = 0.5f;
            S_kinect.SkeletonEngine.SmoothParameters = parameters;
            // Se crea el evento Skeleton_Kinect, que se va a activar cada vez que los distintos puntos o Joint se
encuentren en movimiento.
            S_kinect.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(Skeleton_Kinect);
        }
        // Se crea un método en el cual se va a crear una escala para  los movimientos de la mano derecha.
        private void Pos_ini_Elipses_Derecha(Ellipse elipse, Joint puntos)
        {

            //Se crea la escala de 680 x 480 pixeles por esta escala la mano derecha tenga la posibilidad
            //de moverse por toda la pantalla, 0.5f y 0.2f son dos parámetros que permiten ajustar la precisión
del movimiento tanto en altura como en anchura
            //estos valores se ajustan de acuerdo a las necesidades de cada proyecto.
            var escala_puntos_Mder = puntos.ScaleTo(680, 480, 0.5f, 0.2f);
            //Se crea una variable en donde se van a enlazar los puntos llamada " Act_Pts"
            Joint Act_Pts = new Joint();
            //Se almacena que puntos del skeleton del usuario se está moviendo
            Act_Pts.ID = puntos.ID;
            //Se habilita la opción tracked en la variable para que se alanceen los puntos en movimiento del
skeleton Tracking
            Act_Pts.TrackingState = JointTrackingState.Tracked;
            //Configura la posición de la ellipse para poder moverlos en la ventana principal de a acuerdo a la
posición de los
            //puntos del skeleton utilizando los métodos Canvas.SetLeft y Canvas.SetTop los cuales contienen
los valores de las
            //posiciones (X,Y).
            Canvas.SetLeft(elipse, escala_puntos_Mder.Position.X);
            Canvas.SetTop(elipse, escala_puntos_Mder.Position.Y);
        }


        private void Pos_ini_Elipses_Izquierda(Ellipse elipse, Joint puntos)
        {
            //Se crea la escala de 200 x 50 pixeles el cual es el rango para que la mano izquierda tenga la
posibilidad
            //de moverse para justar la posición del ángulo de la cámara del sensor Kinect
            //0.5f y 0.2f son dos parámetros que permiten ajustar la precisión del movimiento tanto en altura
como en anchura
            //estos valores se ajustan de acuerdo a las necesidades de cada proyecto.
            var escala_puntos_Mizq = puntos.ScaleTo(200, 50, 0.5f, 0.2f);
            //Se crea una variable la cual va a almacenar los puntos que Kinect identifica de los movimientos
que hace el usuario
            Joint Act_Pts = new Joint();
            //Se almacena que puntos del skeleton del usuario se está moviendo
            Act_Pts.ID = puntos.ID;
            //se habilita la opción tracked en la variable para que se almacenen los puntos en movimiento del
skeleto
            Act_Pts.TrackingState = JointTrackingState.Tracked;
```

```
            //configura la posición de la ellipse para poder moverlos en la ventana principal de a acuerdo a la
posición de los
            //puntos del skeleton utilizando los métodos Canvas.SetLeft y Canvas.SetTop los cuales contienen
los valores de las
            // posiciones (X,Y)
            Canvas.SetLeft(elipse, escala_puntos_Mizq.Position.X);
            Canvas.SetTop(elipse, escala_puntos_Mizq.Position.Y);
        }
        private void Posicion_Elipses(Ellipse ellipse, Joint joint)
        {
            //Se crea la escala de 217 x 275 pixeles el cual es el rango para que los objetos ellipses tengan la
posibilidad
            //de moverse por toda la escala de los objetos Canvas
            //0.5f y 0.2f son dos parámetros que permiten ajustar la precisión del movimiento tanto en altura
como en anchura
            //estos valores se ajustan de acuerdo a las necesidades de cada proyecto.
            var scaledJoint = joint.ScaleTo(217, 275, 1f, 0.7f);
            //Se crea una variable la cual va a almacenar los puntos que Kinect identifica de los movimientos
que hace el usuario
            Joint updatedJoint = new Joint();
            //Se almacena que puntos del skeleton del usuario se está moviendo
            updatedJoint.ID = joint.ID;
            //se habilita la opción tracked en la variable para que se almacenen los puntos en movimiento del
skeleton
            updatedJoint.TrackingState = JointTrackingState.Tracked;
            //configura la posición de la ellipse para poder moverlos en la ventana principal de a acuerdo a la
posición de los
            //puntos del Skeleton utilizando los métodos Canvas.SetLeft y Canvas.SetTop los cuales contienen
los valores de las posiciones (X,Y)
            Canvas.SetLeft(ellipse, scaledJoint.Position.X);
            Canvas.SetTop(ellipse, scaledJoint.Position.Y);
        }

        //Se crea un método, que va a permitir ajustar el ángulo del motor de la cámara del sensor Kinect
        private void Proceso_Ajuste_Motor(Joint Mizquierda, Joint Mderecha)
        {
            //Se crea la variable angu, la cual contiene el valor del ángulo de la cámara, este tiene que ser entre -
27 a 27 grados
            int angu = 0;
            //Se delimita el rango de movimiento que puede tener la mano izquierda
            if (Mizquierda.Position.Y >= 0.01 && Mizquierda.Position.Y <= 0.4 && slider1.IsEnabled == true)
            {
                //multiplicamos el valor de la posición de la mano izquierda en el eje x por 100 para tener un
valor apreciable,
                //se obtiene el entero del valor con la función (int) y serán almacenados en la variable angu
                angu = (int)(Mizquierda.Position.X * 100);
                //Slider1 va a moverse dependiendo de los valores que tenga la variable angu (de -27 a 27)
                slider1.Value = angu;
                //Se realiza un control ,para cuando el valor de la variable angu sea mayor que 27 , la variable
angu va a ser igual a 27
                //para que no sobrepase a un valor mayor que 27 .
                if (angu > 27)
                {
                //La variable angu va a ser igual a 27
                    angu = 27;
                }
                //Se crea un control, para determinar que el valor de la variable angu este entre los rangos de +27
Y -27 grados
                else if (angu >= -27 && angu <= 27)
                {
                    //multiplicamos el valor de la posición de la mano izquierda en el eje x por 100 para tener un
valor apreciable,
                    //se obtiene el entero del valor con la función (int) y serán almacenados en la variable angu
                    angu = (int)(Mizquierda.Position.X * 100);
```

```
            }
            //Se realiza un control, para cuando el valor de la variable angu sea menor que -27, la variable
angu va a ser igual a -27
            //para que no sobrepase a un valor mayor que -27.
            else if (angu <= -27)
            {
            //La variable angu va a ser igual a -27
               angu = -27;
            }
            //Se va a transformar la variable "angu", que es un a variable entera a una variable String a través
de la función ToString(), para
            // que en el objeto Label llamado " lb_grados", se despliegue o se muestre al usuario cuantos
grados se esta moviendo al objeto Slider1.
            lb_grados.Content = angu.ToString() + "º";
         }
         // Se realiza un control, determinando las coordenadas en los ejes (X,Y) en las cuales está situado el
botón Ajustar para que cuando el usuario
         //deslice la mano derecha por el botón ,este simule que fue pulsado , además se realiza unos
controles adicionales para determinar si el botón
         //"bt_ajustar" y el objeto Slider1 están habilitados entonces, si cumple con estas condiciones realice
la acción programada
         if ((Mderecha.Position.X >= -0.010 && Mderecha.Position.X <= 0.16) && (Mderecha.Position.Y
>= 0.11 && Mderecha.Position.Y <= 0.14) && (bt_ajustar.IsEnabled == true) && (slider1.IsEnabled ==
true))
         {
            //Se realiza unas condiciones, en donde se determina si el valor del objeto Slider1 es un valor
entero.
            if (S_kinect.NuiCamera.ElevationAngle != (int)slider1.Value)
            {
               //Se procede a almacenar el valor entre +27 o -27 que contenga el objeto Slider1,dentro del
evento ElevationAngle , y de esta forma
               // ajustar el ángulo del motor de la cámara del sensor , según los grados de inclinación o
elevación que haya elegido el usuario.
               S_kinect.NuiCamera.ElevationAngle = (int)slider1.Value;
               //El objeto button "bt_ajustar" se va a deshabilitar a través de la funcion "IsEnable" y
"bt_defecto" se va a proceder a habilitar poniéndolos en las condiciones de falso y verdadero
               // según sea el caso.
               bt_ajustar.IsEnabled = false;
               bt_defecto.IsEnabled = true;
             }
         }
         // Se realiza un control , determinando las coordenadas en los ejes (X,Y) en las cuales esta situado el
botón llamado "bt_defecto" para que cuando el usuario
         //deslice la mano derecha por el botón ,este simule que fue pulsado , además se realiza unos
controles adicionales para determinar si el botón
         //"bt_ajustar" esta deshabilitado a través de la función IsEnable poniéndole en la condición false,  el
objeto Slider1 esta habilitado y el botón "bt_defecto" este habilitado , entonces, si cumplen con estas
condiciones realice la acción programada
         if ((Mderecha.Position.X >= 0.21 && Mderecha.Position.X <= 0.44) && (Mderecha.Position.Y >=
0.11 && Mderecha.Position.Y <= 0.14) && (bt_ajustar.IsEnabled == false) && (slider1.IsEnabled ==
true) && (bt_defecto.IsEnabled == true))
         {
            //Se realiza una condiciones , en donde se determina si el valor del objeto Slider1  es un valor
entero.
            if (S_kinect.NuiCamera.ElevationAngle != (int)slider1.Value)
            {
               //Se procede a almacenar el valor predeterminado de la aplicación que es cero grados (0º),
dentro del evento ElevationAngle , y de esta forma
               // ajustar el ángulo del motor de la cámara del sensor en dicho valor ,independientemente de
cualquier valor seleccionado.
               S_kinect.NuiCamera.ElevationAngle = 0;
               //El objeto button "bt_ajustar" se va a habilitar a través de la función "IsEnable" y "bt_defecto"
se va a proceder a deshabilitar poniéndolos en las condiciones de verdadero y falso
```

```
                // según sea el caso.
                bt_ajustar.IsEnabled = true;
                bt_defecto.IsEnabled = false;
            }
        }
    }
    //Se crea un método en el cual se ajusta la distancia entre el usuario y el sensor Kinect,
    //la variable Posición es enlazada a los datos que contenga el punto o Joint del Skeletal Tracking
llamado "ShoulderCenter".
    private void Proceso_Ajuste_Distancia(Joint Posicion)
    {
        //Se realiza varias condiciones If para determinar la distancia correcta que debe de tener el usuario
con respecto al sensor Kinect,
        //tomando los valores del Joint ShoulderCenter que se mueve en el eje z, y cuando el usuario se
encuentre a la correcta distancia que está comprendida entre
        //1.8m-2m se desplegara en Label a manera de mensajes que su posición es la correcta, caso
contrario se le indicara que está muy cerca o muy alejado de la aplicación.
        //Para el control de distancia se ocupa al eje Z , el cual mide o captura los datos de profundidad de
los distintos puntos con los que cuenta el Esqueleto humano,
        //y de esa manera permite obtener datos de distancia en la aplicación realizada.
        if (Posicion.Position.Z >= 1.8 && Posicion.Position.Z <= 2.0)
        {
            lb_MCalculo_Distancia.Content = "    Tu Distancia hacia el sensor \r                es la
CORRECTA...";
            lb_mDistancia.Content = "Puedes seleccionar el botón Siguiente  \r para empezar la aplicación,
caso contrario\r" +
                " procede a ajustar el motor de la cámara";
            slider1.IsEnabled = true;
        }
        else if (Posicion.Position.Z <= 1.79)
        {
            lb_MCalculo_Distancia.Content = "     Te encuentras MUY CERCA  \r          hacia el sensor...  ";
            lb_mDistancia.Content = "Retrocede un poco para obtener una distancia  \r correcta  y empezar la
aplicación\r" +
                "ó proceder a ajustar el motor de la cámara";
            slider1.IsEnabled = false;
        }
        else if (Posicion.Position.Z >= 2.01)
        {
            lb_MCalculo_Distancia.Content = "     Te encuentras LEJOS  \r          con respecto al sensor...  ";
            lb_mDistancia.Content = "Adelantate un poco para obtener una \r  distancia correcta y empezar la
aplicación \r" +
                " ó proceder a ajustar el motor de la camara";
            slider1.IsEnabled = false;
        }
    }
    //Se crea un método ,para que cuando el usuario sitúe su mano derecha sobre el bt_siguiente y este
habilitado se ejecute este evento
    private void Proceso_Dem_Principal(Joint Mderecha)
    {
        //Se emplea un control para que si el usuario esta situado acorde a las posiciones del botón
"bt_Siguiente" y este habilitado, si cumple esas condiciones
        //que realice la acción programada.
        if ((Mderecha.Position.X >= -0.12 && Mderecha.Position.X <= 0.014) && (Mderecha.Position.Y
>= 0.043 && Mderecha.Position.Y <= 0.079) && (bt_siguiente.IsEnabled == true))
        {
        //El Grid "gr1", va a ponerse invisible y se le va a deshabilitar
            gr1.Visibility = System.Windows.Visibility.Collapsed;
            gr1.IsEnabled = false;
            //Al Grid "gr2" se le vuelve visible y se le habilita.
            gr2.Visibility = System.Windows.Visibility.Visible;
            gr2.IsEnabled = true;
        }
    }
```

```
    // Método en el cual se va a ejecutar todos los eventos que tiene SkeletonFrameReady
    internal void Skeleton_Kinect(Object sender, SkeletonFrameReadyEventArgs e)
    {
      // La variable skeletonSet contiene todas las tramas de los movimientos del usuario
      SkeletonFrame skeletonSet = e.SkeletonFrame;
      // Se crea una variable llamada skeleton donde se especifica los parámetros para determinar si el
usuario está en movimiento con la opción s.TrackingState == SkeletonTrackingState.Tracked
      // la cual va a determinar si existe movimiento y va a capturar los datos
        SkeletonData skeleton = (from s in skeletonSet.Skeletons
                                 where s.TrackingState == SkeletonTrackingState.Tracked
                                 select s).FirstOrDefault();
      //Se verifica si la variable skeleton contiene datos o esta vacía.
      if (skeleton != null)
      {
        //Se captura los datos del punto de la mano derecha y lo enlazamos al ellipse "E_Mder" los cuales
serán enviados al
        //método "Pos_ini_Elipses_Derecha", donde consta las escala para mover la mano derecha.
        Pos_ini_Elipses_Derecha(E_Mder, skeleton.Joints[JointID.HandRight]);
        //Se captura los datos del punto de la mano izquierda y lo enlazamos al ellipse "E_Mizq" los
cuales serán enviados al
        //método "Pos_ini_Elipses_Izquierda", donde consta las escala para mover la mano derecha.
        Pos_ini_Elipses_Izquierda(E_Mizq, skeleton.Joints[JointID.HandLeft]);
        //Se captura los datos de los punto de la mano derecha e izquierda y lo enlazamos al ellipse
"E_Mizq" y "E_Mizq" los cuales serán enviados al
        //método " Proceso_Ajuste_Motor", donde se podrá seleccionar los distintos botones y el objeto
Slider1
        Proceso_Ajuste_Motor(skeleton.Joints[JointID.HandLeft],skeleton.Joints[JointID.HandRight]);
        //Se captura los datos del punto "Centro" entre los hombros llamado "ShoulderCenter" ,los cuales
serán enviados al
        //método "Proceso_Ajuste_Distancia" donde se determina la correcta distancia con respecto al
sensor que debe de tener el usuario.
        Proceso_Ajuste_Distancia(skeleton.Joints[JointID.ShoulderCenter]);
        //Se captura los datos del punto de la mano derecha y lo enlazamos al ellipse "E_Mder" los cuales
serán enviados al
        //método "Proceso_Dem_Principal".
        Proceso_Dem_Principal(skeleton.Joints[JointID.HandRight]);
        ///////////////
        //Se enlaza los objetos Ellipses a los distintos Joints del Skeletal Tracking, en donde se van a
capturar sus datos y van a ser enviados
        //al método "Posicion_Elipses", para que se muevan de acuerdo a la escala seleccionada.
        // Estos objetos Ellipses se van a mover dentro del objeto Canvas llamado "Canvas_Skeletal".
        Posicion_Elipses(e_cabeza, skeleton.Joints[JointID.Head]);
        Posicion_Elipses(e_p1, skeleton.Joints[JointID.ShoulderCenter]);
        Posicion_Elipses(e_pi1, skeleton.Joints[JointID.ShoulderLeft]);
        Posicion_Elipses(e_pi2, skeleton.Joints[JointID.ElbowLeft]);
        Posicion_Elipses(e_pi3, skeleton.Joints[JointID.WristLeft]);
        Posicion_Elipses(e_pi4, skeleton.Joints[JointID.HandLeft]);
        Posicion_Elipses(e_pd1, skeleton.Joints[JointID.ShoulderRight]);
        Posicion_Elipses(e_pd2, skeleton.Joints[JointID.ElbowRight]);
        Posicion_Elipses(e_pd3, skeleton.Joints[JointID.WristRight]);
        Posicion_Elipses(e_pd4, skeleton.Joints[JointID.HandRight]);
        Posicion_Elipses(e_pc1, skeleton.Joints[JointID.Spine]);
        Posicion_Elipses(e_pc2, skeleton.Joints[JointID.HipCenter]);
        Posicion_Elipses(e_pc3, skeleton.Joints[JointID.HipLeft]);
        Posicion_Elipses(e_pc4, skeleton.Joints[JointID.HipRight]);
        ////////////////////////
        // Estos objetos Ellipses se van a mover dentro del objeto Canvas llamado "Can_SC".
        Posicion_Elipses(e_e1, skeleton.Joints[JointID.ShoulderCenter]);
        Posicion_Elipses(e_ei1, skeleton.Joints[JointID.ShoulderLeft]);
        Posicion_Elipses(e_ei2, skeleton.Joints[JointID.ElbowLeft]);
        Posicion_Elipses(e_ei3, skeleton.Joints[JointID.WristLeft]);
        Posicion_Elipses(e_ei4, skeleton.Joints[JointID.HandLeft]);
        Posicion_Elipses(e_ed1, skeleton.Joints[JointID.ShoulderRight]);
```

```
            Posicion_Elipses(e_ed2, skeleton.Joints[JointID.ElbowRight]);
            Posicion_Elipses(e_ed3, skeleton.Joints[JointID.WristRight]);
            Posicion_Elipses(e_ed4, skeleton.Joints[JointID.HandRight]);
            Posicion_Elipses(e_ec1, skeleton.Joints[JointID.Spine]);
            Posicion_Elipses(e_ec2, skeleton.Joints[JointID.HipCenter]);
            Posicion_Elipses(e_eci, skeleton.Joints[JointID.HipLeft]);
            Posicion_Elipses(e_ecd, skeleton.Joints[JointID.HipRight]);

            //Se captura los datos de los Joint`s de la mano derecha y de la mano izquierda,
            // que serán enviados al método llamado "Pro_Envio_Datos"
            Pro_Envio_Datos(skeleton.Joints[JointID.HandLeft], skeleton.Joints[JointID.HandRight]);
        }
    }
    //Se crea este método en el cual se va a realizar las respectivas escalas para obtener los datos obtenidos
    //en grados de 0-180Grados y además se realiza el envió de datos a través del puerto serial
    private void Pro_Envio_Datos(Joint handleft, Joint handrigth)
    {
        //Debido a que los datos obtenidos varían entre 0 y 80
        //los valores de eje_x y eje_y se multiplican por 225 para tener una escala de 0 a 180
        eje_x = (int)(handrigth.Position.X * 225);
        eje_y = (int)(handrigth.Position.Y * 225);
        //Los valores de eje_z se multiplica por 10 para tener un rango más apreciable.
        eje_Z = (int)(handrigth.Position.Z * 10);
        eje_xi = (int)(handleft.Position.X * 225);
        eje_yi = (int)(handrigth.Position.Y * 225); // OJO MOVIMIENTO BRAZO DERECHO EJE Y
        eje_Zi = (int)(handleft.Position.Z * 10);

        // Se delimita los valores en eje_x y eje_y para que estén en el rango de 0 a 180 grados
        //además se realiza un control que se ejecute la aplicación que contiene el Grid "gr2",solamente
        //si el Grid "gr1" este invisible.
        if (gr2.IsVisible == true && gr1.IsVisible == false)
        {
            if (eje_x >= 0 && eje_x <= 179)
            {

                eje_xs = eje_x;
            }
            else if (eje_x <= 0)
            {

                eje_xs = 0;
            }
            else if (eje_x >= 180)
            {

                eje_xs = 179;
            }
            // brazo izquierdo x
            if (eje_xi >= -179 && eje_xi <= 0)
            {

                eje_xis = eje_xi;
            }
            else if (eje_xi >= 0)
            {

                eje_xis = 0;
            }
            else if (eje_xi <= -180)
            {

                eje_xis = -179;
            }
            /// brazo derecho eje y
```

```
            /// Se delimita los valores del eje y para que estén en un rango de 0 a 120 grados, y así la pinza
robótica no choque con
            /// La base cuando llegue a la parte inferior
            if (eje_yi >= 0 && eje_yi <= 120)
            {

                eje_yis = 120 - eje_yi;

            }
            else if (eje_yi <= 0)
            {

                eje_yis = 120;
            }
            else if (eje_yi >= 121)
            {

                eje_yis = 0;
            }

            // Brazo izquierdo eje z, se delimita los valores para que estén en el rango de 90 a 147 grados
            if (eje_Zi <= 12)
            {

                eje_Zis = 90;
                Lb_Gra_Pinza.Content = "Apertura";
                e_pinza.Fill = System.Windows.Media.Brushes.White;

            }
            else if (eje_Zi >= 16)
            {
                eje_Zis = 147;
                Lb_Gra_Pinza.Content = "Cierre";
                e_pinza.Fill = System.Windows.Media.Brushes.Transparent;
            }
        }
        //A los valores obtenidos de los ejes se les transforma a una cadena de caracteres a través de la
función ToString(),
        //para que esos valores sean desplegados en los Objetos Label
        lb_Gra_H.Content = eje_xs.ToString() + " º";
        lb_Gra_V.Content = eje_yis.ToString() + " º";


        //Con la función Math.Abs se obtiene el valor absoluto del eje_xis
        eje_xis = Math.Abs(eje_xis);
        // Se crea un vector llamado servo de tipo byte de 3 posiciones en los cuales se almacenan los datos
        // transformados a tipo byte de eje_xs, eje_yis y eje_Zis
        byte[] servo = new byte[] { (byte)(eje_xs), (byte)(eje_yis), (byte)(eje_Zis) };
        //Se especifica la variable que se va a escribir en el puerto desde la posición 0 calculando su
longitud
        puerto.Write(servo, 0, servo.Length);

        //////////////
        //Este proceso se emplea solamente para que los objetos Ellipses se pinten en determinados colores
cada vez
        //que se encuentren en los rangos programados
        if (eje_xs >= 0 && eje_xs <= 44)
        {
            e_dv_45.Fill = System.Windows.Media.Brushes.Yellow;
            e_dv_90.Fill = System.Windows.Media.Brushes.Purple;
        }
        else if (eje_xs >= 45 && eje_xs <= 89)
        {
            e_dv_90.Fill = System.Windows.Media.Brushes.Red;
```

```
            e_dv_45.Fill = System.Windows.Media.Brushes.Purple;
            e_dv_135.Fill = System.Windows.Media.Brushes.Purple;
        }
        else if (eje_xs >= 90 && eje_xs <= 134)
        {
            e_dv_135.Fill = System.Windows.Media.Brushes.Blue;
            e_dv_90.Fill = System.Windows.Media.Brushes.Purple;
            e_dv_180.Fill = System.Windows.Media.Brushes.Purple;
        }
        else if (eje_xs >= 135 && eje_xs <= 179)
        {
            e_dv_180.Fill = System.Windows.Media.Brushes.Green;
            e_dv_135.Fill = System.Windows.Media.Brushes.Purple;
        }
        ////////////////////
        if (eje_yis >= 0 && eje_yis <= 44)
        {
            e_dh_45.Fill = System.Windows.Media.Brushes.Yellow;
            e_dh_90.Fill = System.Windows.Media.Brushes.Purple;
        }
        else if (eje_yis >= 45 && eje_yis <= 89)
        {
            e_dh_90.Fill = System.Windows.Media.Brushes.Red;
            e_dh_45.Fill = System.Windows.Media.Brushes.Purple;
            e_dh_135.Fill = System.Windows.Media.Brushes.Purple;
        }
        else if (eje_yis >= 90 && eje_yis <= 134)
        {
            e_dh_135.Fill = System.Windows.Media.Brushes.Blue;
            e_dh_90.Fill = System.Windows.Media.Brushes.Purple;
            e_dh_180.Fill = System.Windows.Media.Brushes.Purple;
        }
        else if (eje_yis >= 135 && eje_yis <= 179)
        {
            e_dh_180.Fill = System.Windows.Media.Brushes.Green;
            e_dh_135.Fill = System.Windows.Media.Brushes.Purple;
        }

    }


    private void im_SC_ImageFailed(object sender, ExceptionRoutedEventArgs e)
    {

    }

  }
}
```

3.3.2.3. Codificación programa Arduino uno

```
#include <Servo.h> // agrega la librería de Arduino para el control de servomotores
//leds
int Led8 = 8;
int Led10 = 10;
int Led11 = 11;
int Led12 = 12;
int Led13 = 13;

int Led2 = 2;
int Led4 = 4;
int Led5 = 5;
int Led7 = 7;

//Servo servo;
Servo  d_ejex_servo;
Servo  i_ejey_servo;
Servo  i_ejez_servo;
void setup()
{
  pinMode(Led8, OUTPUT);
  pinMode(Led10, OUTPUT);
  pinMode(Led11, OUTPUT);
  pinMode(Led12 , OUTPUT);
  pinMode(Led13, OUTPUT);


  pinMode(Led2, OUTPUT);
  pinMode(Led4 , OUTPUT);
  pinMode(Led5 , OUTPUT);
  pinMode(Led7, OUTPUT);
  // Inicializamos el puerto serial a 9600 bps:
  Serial.begin(9600);

  d_ejex_servo.attach(3);
  i_ejey_servo.attach(6);
  i_ejez_servo.attach(9);

   d_ejex_servo.write(0);
   i_ejey_servo.write(0);
```

```
  i_ejez_servo.write(0);

}

unsigned char dx,iy,iz =0;
int val = 0;

void loop()
{

 if (Serial.available() >= 3) {
   dx = Serial.read();
   iy = Serial.read();
   iz = Serial.read();


      d_ejex_servo.write(dx);
      i_ejey_servo.write(iy);
      i_ejez_servo.write(iz);
   if ( dx >= 0 && dx <=36 ){
     digitalWrite(Led8, HIGH);
     digitalWrite(Led10, LOW);
     digitalWrite(Led11, LOW);
     digitalWrite(Led12, LOW);
     digitalWrite(Led13, LOW);
   // delay(1000);
   }
    if(dx >= 37 && dx <=72) {

     digitalWrite(Led8, LOW);
     digitalWrite(Led10, HIGH);
     digitalWrite(Led11, LOW);
     digitalWrite(Led12, LOW);
     digitalWrite(Led13, LOW);
     //delay(1000);
   }
   if(dx >= 73 && dx <=108) {

     digitalWrite(Led8, LOW);
     digitalWrite(Led10, LOW);
     digitalWrite(Led11, HIGH);
```

```
    digitalWrite(Led12, LOW);
    digitalWrite(Led13, LOW);
    //delay(1000);
   }
   if(dx >= 109 && dx <=144) {

    digitalWrite(Led8, LOW);
    digitalWrite(Led10, LOW);
    digitalWrite(Led11, LOW);
    digitalWrite(Led12, HIGH);
    digitalWrite(Led13, LOW);
    //delay(1000);
   }
   if(dx >= 145 && dx <=180) {

    digitalWrite(Led8, LOW);
    digitalWrite(Led10, LOW);
    digitalWrite(Led11, LOW);
    digitalWrite(Led12, LOW);
    digitalWrite(Led13, HIGH);
    //delay(1000);
   }

   if ( iy >= 0 && iy <=60 ){


    digitalWrite(Led4, HIGH);
    digitalWrite(Led5, LOW);
    digitalWrite(Led7, LOW);
}
  if ( iy >= 61 && iy <=120 ){


    digitalWrite(Led4, LOW);
    digitalWrite(Led5, HIGH);
    digitalWrite(Led7, LOW);
}
  if ( iy >= 121 && iy <=180){


    digitalWrite(Led4, LOW);
```

```
    digitalWrite(Led5, LOW);
    digitalWrite(Led7, HIGH);
}


if ( iz >= 180 ){

  digitalWrite(Led2, LOW);



}

if ( iz <= 90){

  digitalWrite(Led2, HIGH);
}

}
}
```

3.4. Hardware Utilizado

El hardware utilizado consta de tres partes fundamentales que son el Kinect que es el encargado de captar el movimiento de los brazos y manos, un computador con el SDK de Kinect instalado el cual mediante la aplicación realizada en WPF C# reconocerá los datos enviados por el Kinect y los presentará en la pantalla de manera visual, y la tarjeta Arduino UNO que está conectada al computador vía USB la cual permitirá trasmitir los datos adquiridos desde el sensor Kinect hacia la pinza robótica para que esta pueda realizar varios movimientos en manera horizontal, vertical y apretura y cierre de la pinza.

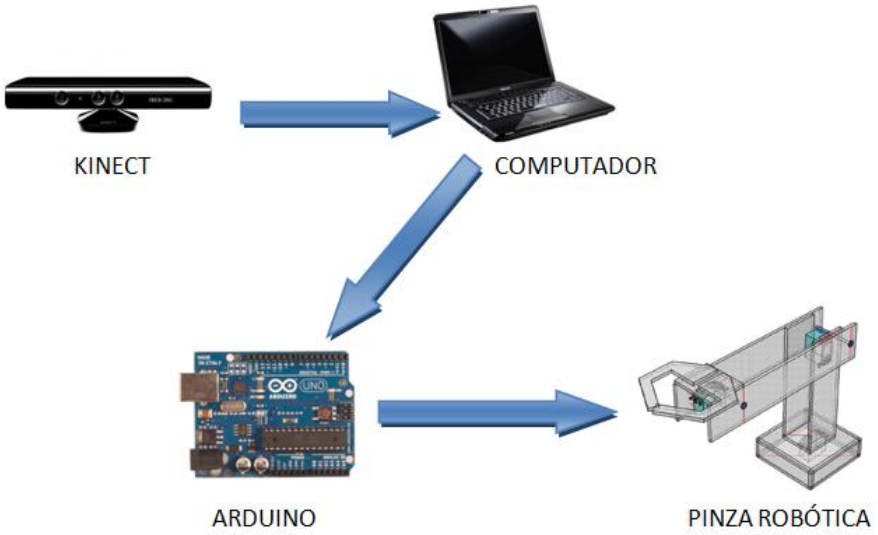


FIGURA 42 Hardware Utilizado[42]

[42] FIGURA 42 Tomada de Autores Tesis

3.4.1. Construcción de la Aplicación Demostrativa

En esta sección se mostrará el proceso desarrollado para la construcción de la aplicación demostrativa "Pinza Robótica", los diferentes diagramas y conexiones que se tienen entre la tarjeta Arduino, los Servomotores y los Diodos Led's.

Todos los diagramas fueron realizados en Fritzing que es un Software Libre especializado para el diseño electrónico.

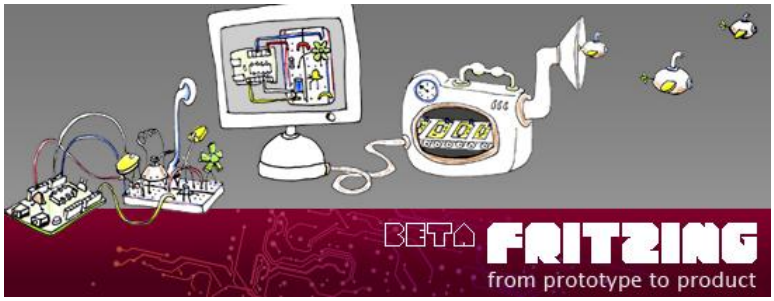


FIGURA 43 Software Fritzing[43]

3.4.2. Diagrama de conexiones

Lo más importante antes de realizar las conexiones en Arduino es tener en cuenta los pines disponibles y cuántos de ellos vamos a utilizar. Para éste caso se necesitan 3 servomotores por lo cual utilizamos las salidas de tipo PWM de Arduino 3, 6 y 9, para el control del encendido y apagado de LED's se utilizan los pines de salida desde el 2 hasta el 13 exceptuando los utilizados en el control de los servomotores.

En el diagrama de conexiones de la figura se muestra cómo se encuentran conectados los servomotores y los LED's hacia la Tarjeta Arduino UNO, tal y como se los debería conectar en un protoboard.

[43] FIGURA 43 Tomada de Pantalla del inicio Programa Fritzing

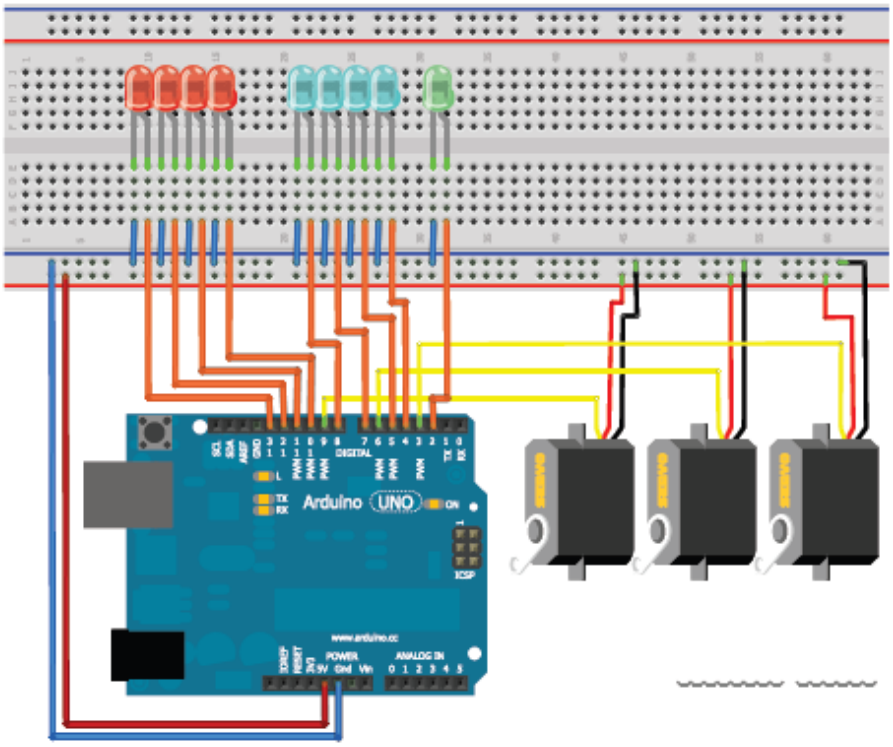


FIGURA 44 Diagrama de conexiones de Arduino UNO con servomotores y LED's indicadores[44]

3.4.3. Diagrama Esquemático

Este diagrama muestra el circuito esquemático de las conexiones de los Led's y los servomotores hacia Arduino.

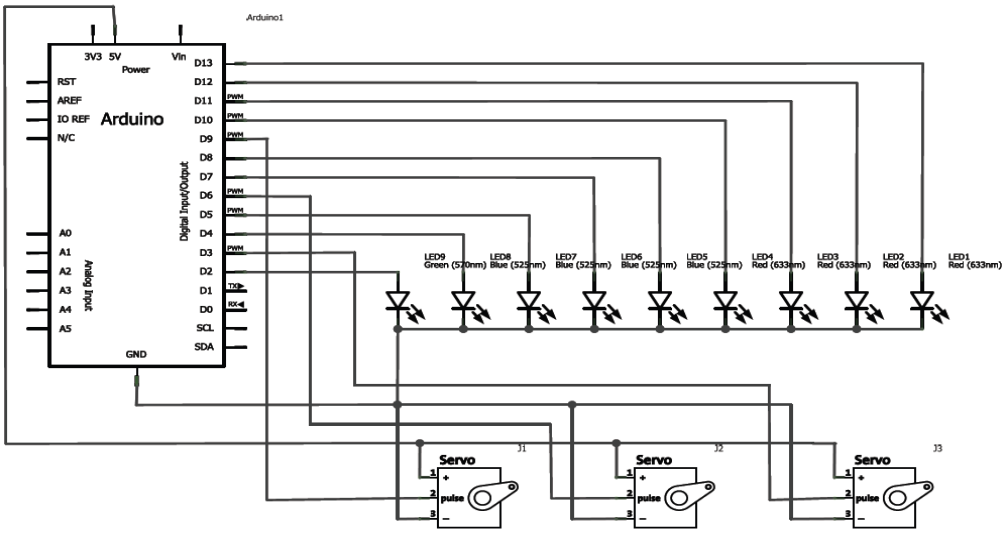


FIGURA 45 Diagrama Esquemático de conexiones [45]

[44] FIGURA 44 Realizada en programa Fritzing
[45] FIGURA 45 Realizada en programa Fritzing

3.4.4. Diseño del Circuito Impreso (PCB)

En la Figura se muestra el circuito impreso diseñado para las conexiones entre los servomotores y los Led's hacia la tarjeta Arduino, se diseñó para que el circuito impreso se conecte directamente hacia la tarjeta Arduino, se utilizaron borneras para la conexión de los Led's y los servomotores.

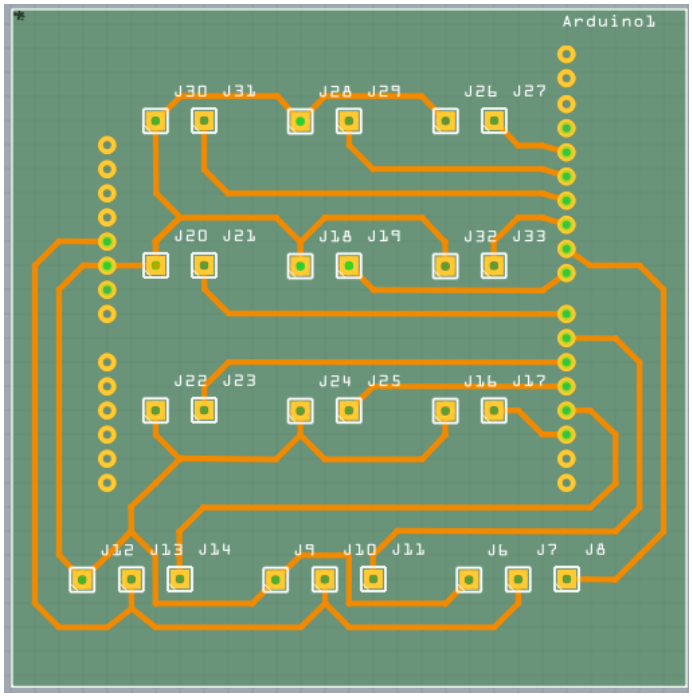


FIGURA 46 Diseño del circuito impreso (PCB)[46]

3.4.5. Diseño de la Pinza Robótica

Como aplicación demostrativa se decidió construir una pinza robótica la cual se moverá de acuerdo al movimiento realizado por el usuario con sus brazos y manos. Tiene 3 grados de libertad, movimiento de izquierda a derecha (180

[46] FIGURA 46 Diagrama esquemático realizado en programa Fritzing

grados en el eje x), movimiento vertical (180 grados en el eje y) y apertura y cierre de la pinza.

La estructura de la pinza robótica está construida en acrílico, los servomotores se encuentran ubicados en la base, en el brazo y en la pinza.

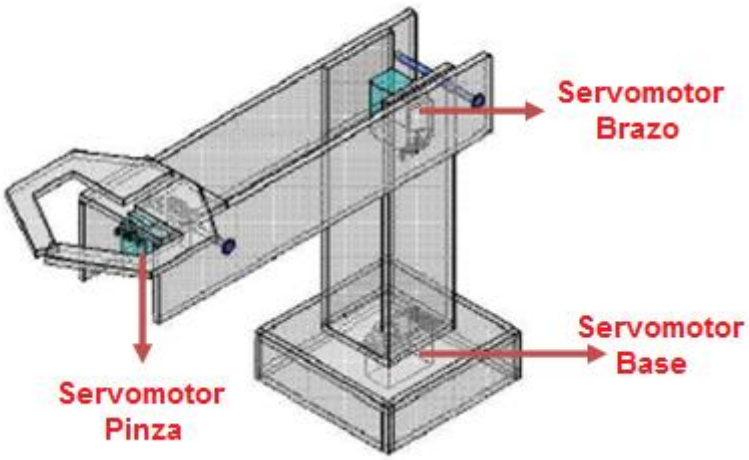


FIGURA 47 Pinza Robótica[47]

En la figura se muestran las diferentes vistas de la pinza robótica las cuales están realizadas en Autocad 2012, también se observa la posición de los servomotores, uno en la base para el movimiento de rotación en X, otro en la parte superior que permite el movimiento hacia arriba y hacia abajo y el último servomotor para el movimiento de la pinza.

[47] FIGURA 47 Realizada en Autocad 2012

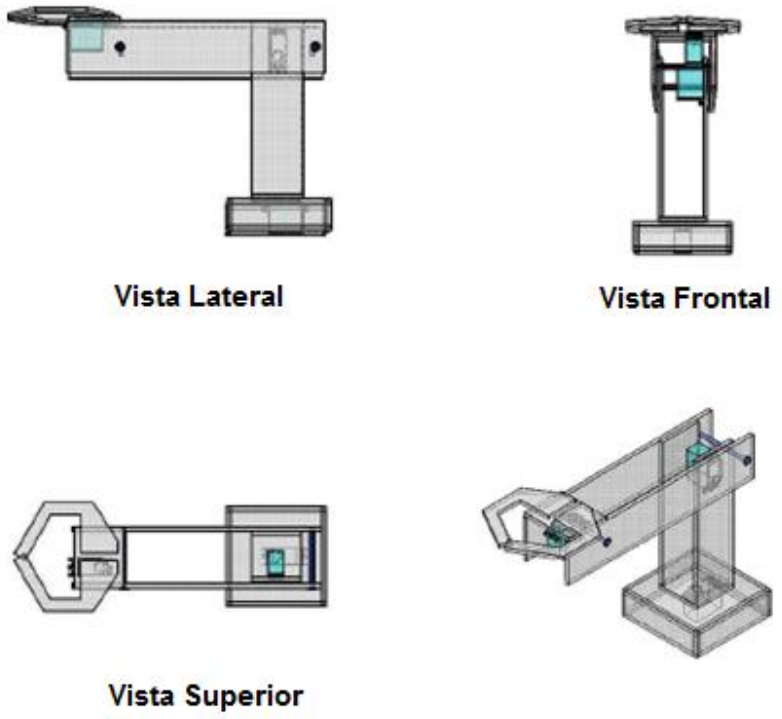


FIGURA 48 Vistas De La Pinza Robótica[48]

[48] FIGURA 48 Realizada en Autocad 2012

CAPITULO 4

4.1. Análisis de Resultados

Para el análisis de resultados se han tomado en cuenta ciertos parámetros que de una manera u otra afectan al correcto funcionamiento de la aplicación demostrativa, se consideró como parámetros más importantes a los siguientes:

- Distancia entre el Usuario y el sensor Kinect.
- Estatura del Usuario
- Número de Usuarios Frente al Sensor Kinect

Con los datos obtenidos se quiere tener un análisis detallado de las condiciones de funcionamiento y determinar las posiciones requeridas del Sensor Kinect y el Usuario para que cualquier persona pueda utilizar la aplicación sin problema alguno.

Todos los datos tomados en las diferentes pruebas fueron realizados teniendo al sensor Kinect ubicado a 0.75m respecto al piso.

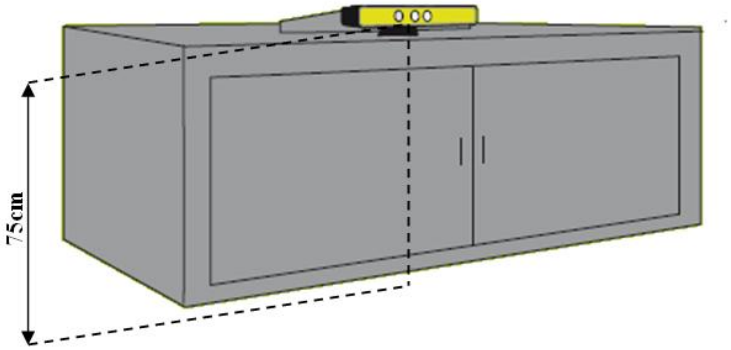


FIGURA 49 Distancia Kinect - Piso[49]

[49] FIGURA 49 REALIZADA POR Autores Tesis

4.1.1. Distancia entre el usuario y el sensor Kinect

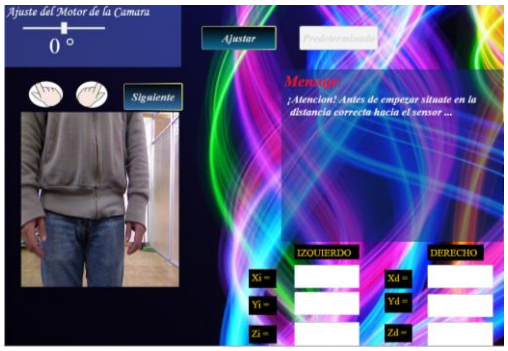
Con los siguientes datos se quiere obtener la distancia correcta entre el usuario y el sensor Kinect, y además determinar qué distancia es considerada muy cercana o muy lejana para el manejo adecuado de la aplicación.

Las pruebas se realizaron en 6 distancias diferentes, comprendidas entre 0.5m a 3m, en la aplicación de ajuste se tomaron los datos de las posiciones de las manos en los diferentes ejes x, y, z detectados por Kinect y en la aplicación final se observo la variación de los grados desplazamiento de los servomotores.

Las capturas de pantalla de cada una de las pruebas son mostradas en las imágenes siguientes:

- **Distancia 1 (0.5m)**

  A esta distancia el sensor Kinect no detecta al usuario que tiene en frente, debido a que se encuentra muy cerca del sensor, además se puede observar que en la Aplicación Final los puntos que corresponden a los Joints del Skeletal de Kinect se agrupan al lado inferior izquierdo de la imagen capturada por la cámara, por lo tanto a esta distancia las aplicaciones realizadas no tienen los datos necesarios para entrar en funcionamiento.



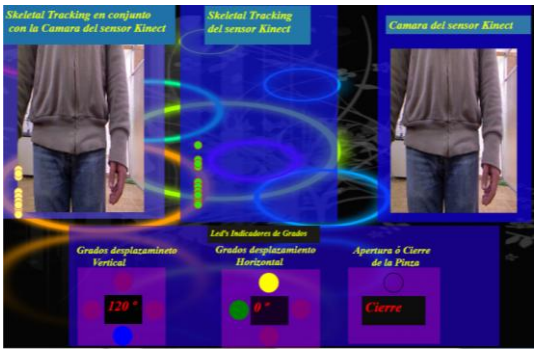
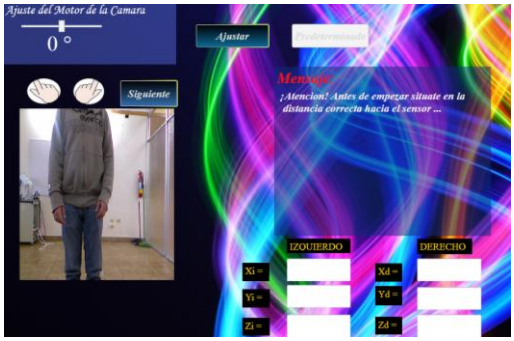


FIGURA 50 Distancia Kinect – Usuario 0.5m (a)[50] FIGURA 51 Distancia Kinect – Usuario 0.5m (b)[51]

- **Distancia 2 (1m)**

  A pesar de que el usuario está un poco más alejado del sensor Kinect aún no se puede observar su rostro ni los hombros, entonces los puntos que son tomados por Kinect para graficar el Skeleton no son los suficientes, y de la misma manera que para la distancia de 0.5m las aplicaciones que fueron realizadas no pueden tomar los datos suficientes para funcionar.



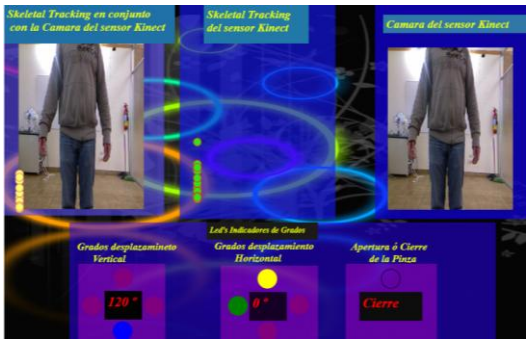


FIGURA 52 Distancia Kinect – Usuario 1m (a) [52] FIGURA 53 Distancia Kinect – Usuario 1m (b)[53]

- **Distancia 3 (1.5m)**

  A esta distancia ya se tienen datos para cada uno de los brazos y aunque todavía el usuario no es visible por completo Kinect ya puede completar los

[50] FIGURA 50 REALIZADA POR Autores Tesis
[51] FIGURA 51 REALIZADA POR Autores Tesis
[52] FIGURA 52 REALIZADA POR Autores Tesis
[53] FIGURA 53 REALIZADA POR Autores Tesis

Joints faltantes y dibujar el Skeletal sobre la imagen del usuario de una manera correcta.

En la aplicación final se pueden observar los grados de desplazamiento que tienen los brazos los cuales sus valores van de 0 (Posición Superior) a 120(Posición Inferior) en el eje vertical y de 0 a 180 para el eje horizontal. Se debe tomar en cuenta que en la aplicación solo es tomado como referencia al brazo derecho para realizar estas medidas.

Para la distancia actual cuando el usuario tiene los brazos arriba la aplicación marca 0 grados y brazos abajo 120 por lo cual tendrá un control total de desplazamiento de manera vertical, cuando tiene los brazos extendidos marca 168 grados de desplazamiento horizontal el cual será el desplazamiento máximo que podrá alcanzar la aplicación en ese eje a esta distancia del sensor.

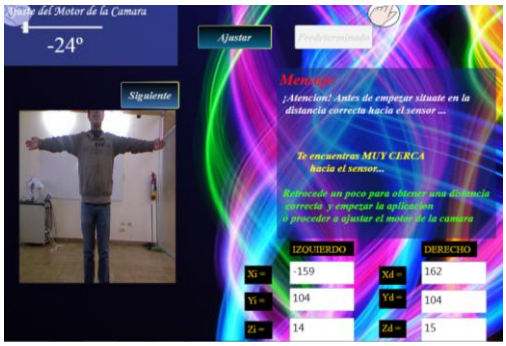


FIGURA 54 Distancia Kinect – Usuario 1.5m (a)[54]

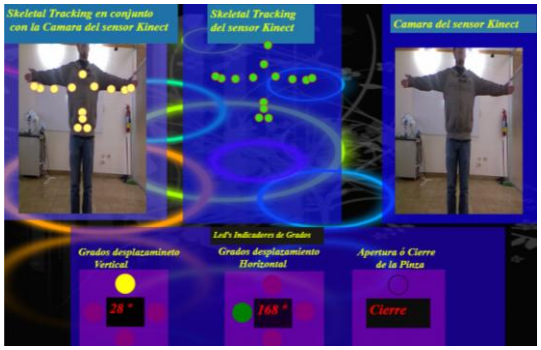


FIGURA 55 Distancia Kinect – Usuario 1.5m (b)[55]

[54] FIGURA 54 REALIZADA POR Autores Tesis
[55] FIGURA 55 REALIZADA POR Autores Tesis

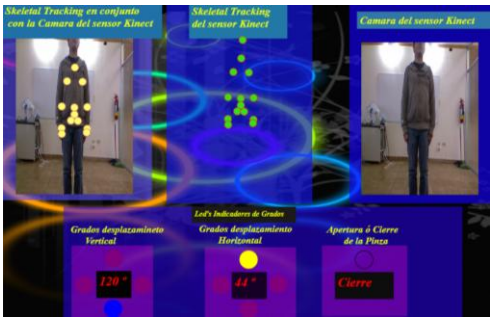


FIGURA 56 Distancia Kinect – Usuario 1.5m (c)[56]

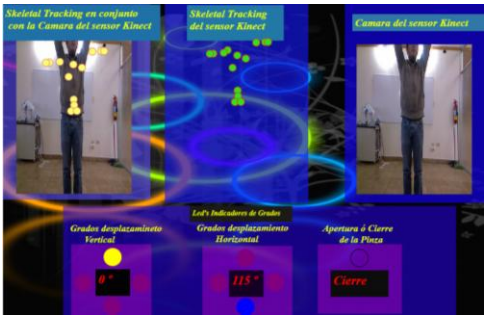


FIGURA 57 Distancia Kinect – Usuario 1.5m (d)[57]

- **Distancia 4 (2m)**

  A esta distancia la imagen del usuario ya se ve completamente en la cámara, Kinect no tendrá ningún problema con tomar los datos del Skeletal, se puede observar que ahora en los datos de ajuste que se tienen en x, y, z para cada brazo han variado de los que se tenía en la anterior distancia, los datos en x y en z van creciendo a medida que el usuario se va alejando del sensor mientras que los del eje y van reduciéndose de a poco.

  Para la distancia actual cuando el usuario tiene los brazos arriba la aplicación marca 0 grados y brazos abajo 120 por lo cual tendrá un control total de desplazamiento de manera vertical, cuando tiene los brazos extendidos marca 151 grados de desplazamiento horizontal el cual será el desplazamiento máximo que podrá alcanzar la aplicación en ese eje a esta distancia del sensor.

[56] FIGURA 56 REALIZADA POR Autores Tesis
[57] FIGURA 57 REALIZADA POR Autores Tesis

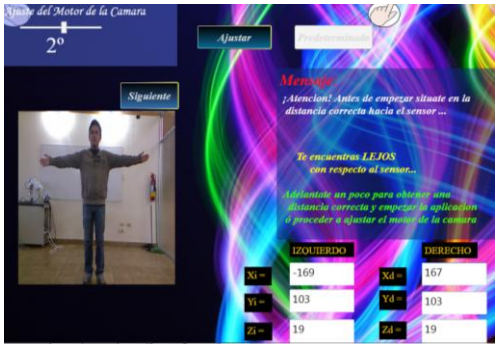


FIGURA 58 Distancia Kinect – Usuario 2m (a)[58]

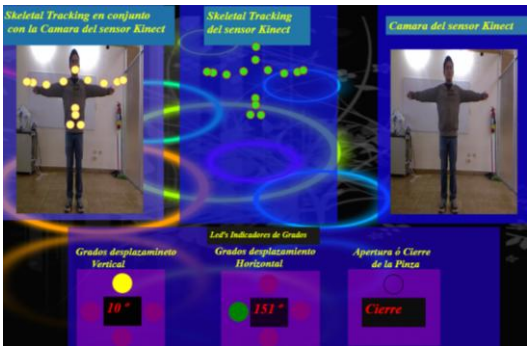


FIGURA 59 Distancia Kinect – Usuario 2m (b)[59]

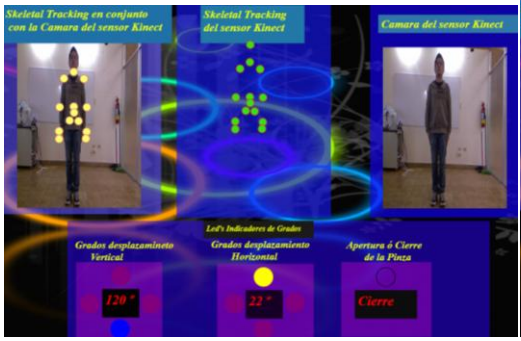


FIGURA 60 Distancia Kinect – Usuario 2m (c)[60]

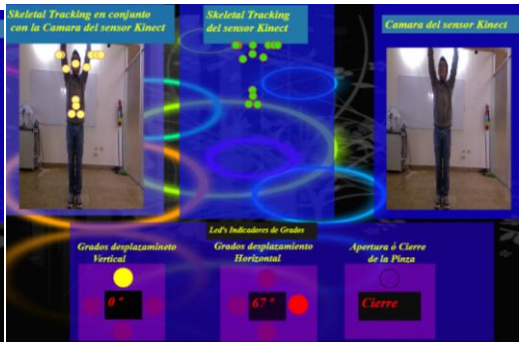


FIGURA 61 Distancia Kinect – Usuario 2m (d)[61]

- **Distancia 5 (2.5m)**

  Ahora mientras el usuario se encuentra más alejado del sensor Kinect lo reconoce pero como se puede observar en las imágenes los puntos Joints del Skeletal no están sobre la imagen del usuario sino que se encuentran sobre él. Los datos que se obtienen de los brazos en x no varían demasiado de la medida anterior, en el eje Y sigue disminuyendo debido que al alejarse del sensor, la imagen del usuario se hace más pequeña y en z sigue aumentando porque éste nos da una medida casi exacta de la distancia a la que se encuentra el usuario, si la medida que tenemos en z se la multiplica por 10

[58] FIGURA 58 REALIZADA POR Autores Tesis
[59] FIGURA 59 REALIZADA POR Autores Tesis
[60] FIGURA 60 REALIZADA POR Autores Tesis
[61] FIGURA 61 REALIZADA POR Autores Tesis

tendremos la distancia en centímetros a la que se encuentra el usuario en ese momento.

Al igual que para las distancias anteriores, para la distancia actual cuando el usuario tiene los brazos arriba la aplicación marca 0 grados y brazos abajo 120 por lo cual tendrá un control total de desplazamiento de manera vertical, cuando tiene los brazos extendidos marca 147 grados de desplazamiento horizontal el cual será el desplazamiento máximo que podrá alcanzar la aplicación en ese eje a esta distancia del sensor.

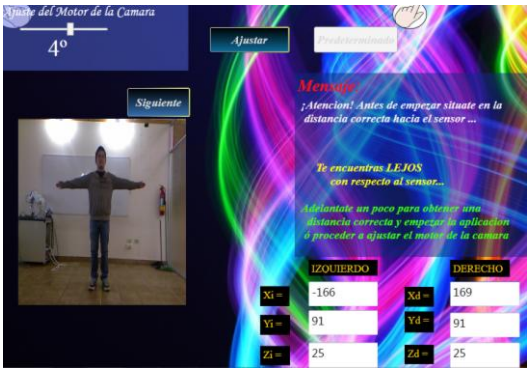


FIGURA 62 Distancia Kinect – Usuario 2.5m (a)[62]

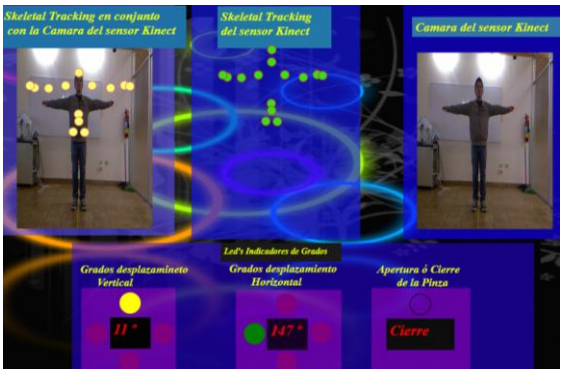


FIGURA 63 Distancia Kinect – Usuario 2.5m (b)[63]

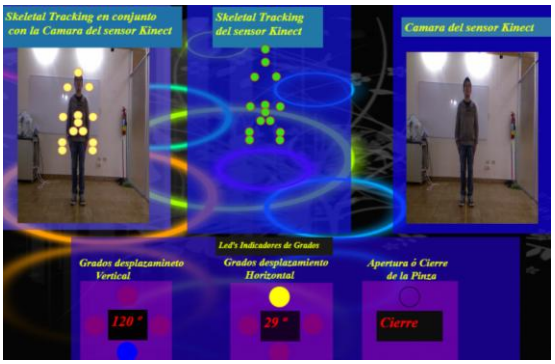


FIGURA 64 Distancia Kinect – Usuario 2.5m (c)[64]

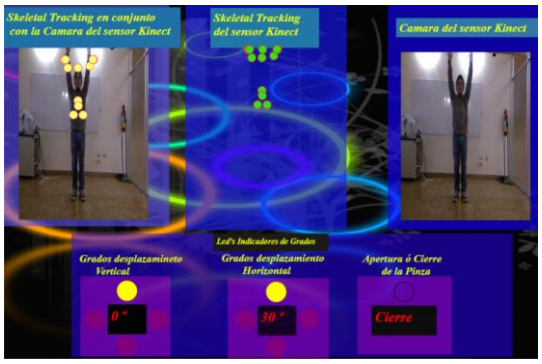


FIGURA 65 Distancia Kinect – Usuario 2.5m (d)[65]

[62] FIGURA 62 REALIZADA POR Autores Tesis
[63] FIGURA 63 REALIZADA POR Autores Tesis
[64] FIGURA 64 REALIZADA POR Autores Tesis
[65] FIGURA 65 REALIZADA POR Autores Tesis

- **Distancia 6 (3m)**

  Cuando el usuario se encuentra a esta distancia el sensor Kinect lo reconoce pero de igual manera que para la distancia de 2.5m los Joints del Skeleton no están sobre la imagen del usuario sino que se encuentran sobre él. Los datos en "y" casi no varían pero aumentan en uno a la medida anterior, puede deberse que el usuario tenía los brazos un poco mas levantados esta vez, los datos en z nos dan 30 por lo que quiere decir que estamos a 3 m del sensor.

  Al igual que para las distancias anteriores para la distancia actual cuando el usuario tiene los brazos arriba la aplicación marca 0 grados y brazos abajo 120 por lo cual tendrá un control total de desplazamiento de manera vertical, cuando tiene los brazos extendidos marca 143 grados de desplazamiento horizontal el cual será el desplazamiento máximo que podrá alcanzar la aplicación en ese eje a esta distancia del sensor, si comparamos esta medida de 143 grados a la que teníamos de 168 grados a una distancia de 1.5m vemos que mientras más se aleja uno del sensor Kinect, los grados de desplazamiento

  en el eje horizontal disminuyen por lo cual no tendríamos un movimiento completo al momento de controlar la pinza robótica desde lejos, ya que los grados de movimiento enviados hacia el servomotor que se encuentra en la base del brazo será menores que los que se tendrían en un posición mas cercana.

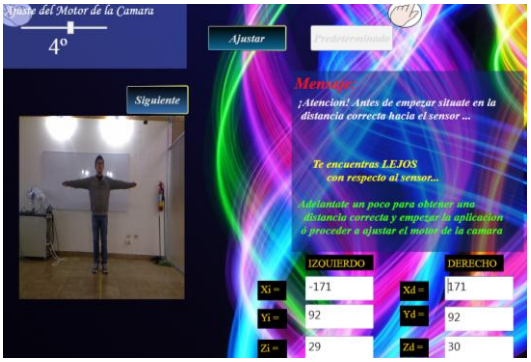


FIGURA 66 Distancia Kinect – Usuario 3m (a)[66]

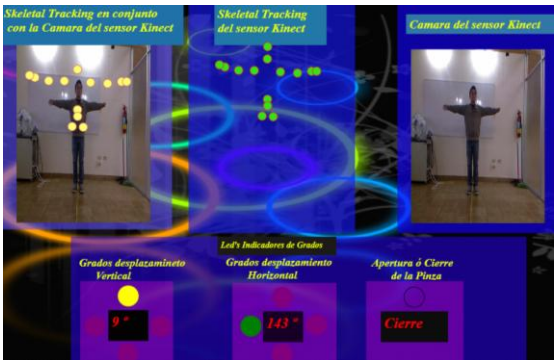


FIGURA 67 Distancia Kinect – Usuario 3m (b)[67]

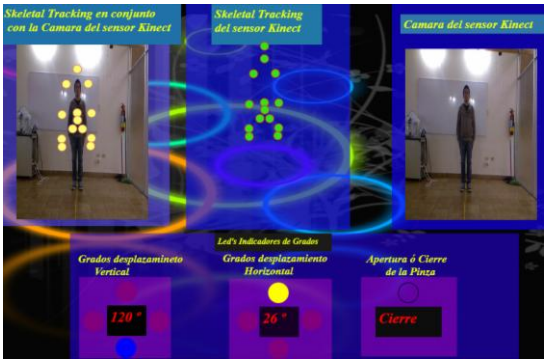


FIGURA 68 Distancia Kinect – Usuario 3m (c) [68]

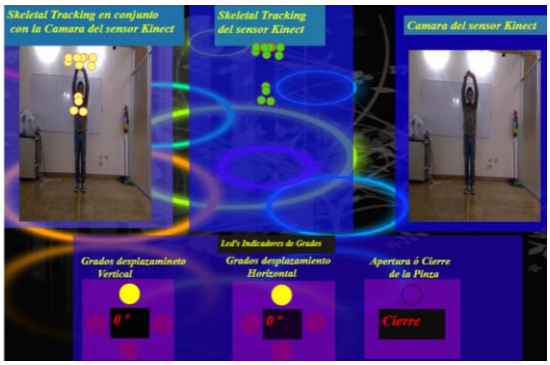


FIGURA 69 Distancia Kinect – Usuario 3m (d)[69]

La siguiente tabla muestra todos los datos obtenidos con estas pruebas:

Distancia entre Usuario y Kinect							
Nº	**Distancia**	**Datos Obtenidos Aplicación Ajuste**					
		X		**y**		**Z**	
		Brazo Izquierdo	**Brazo Derecho**	**Brazo Izquierdo**	**Brazo Derecho**	**Brazo Izquierdo**	**Brazo Derecho**
1.	0.50m	-	-	-	-	-	-
2.	1m	-	-	-	-	-	-
3.	1.5m	-159	162	104	104	14	15
4.	2m	-169	167	103	103	19	19
5.	2.5m	-166	169	91	91	25	25
6.	3m	-171	171	92	92	29	30
		Datos Obtenidos Aplicación Final					
		Grados de Desplazamiento					
		Posición de los brazos		Vertical		Horizontal	
7.	0.50 - 1m	Abiertos		120		0	
		Abajo		120		0	

[66] FIGURA 66 REALIZADA POR Autores Tesis
[67] FIGURA 67 REALIZADA POR Autores Tesis
[68] FIGURA 68 REALIZADA POR Autores Tesis
[69] FIGURA 69 REALIZADA POR Autores Tesis

		Arriba	120	0
8.	1.5m	Abiertos	28	168
		Abajo	120	44
		Arriba	0	115
9.	2m	Abiertos	10	151
		Abajo	120	22
		Arriba	0	67
10.	2.5m	Abiertos	11	147
		Abajo	120	29
		Arriba	0	30
11.	3m	Abiertos	9	143
		Abajo	120	26
		Arriba	0	0

Tabla 5 Distancia Kinect – Usuario

Los datos obtenidos permiten observar la variación que se tiene de los puntos tomados en las manos derecha e izquierda del usuario en cada una de las distancias, se determinó que para distancias menores de 1.5 metros la aplicación no funciona debido a que se está muy cerca del sensor, y que para las distancias mayores de 2 metros los puntos del Skeletal no son bien dibujados sobre la imagen del usuario, estos aparecen más arriba del lugar en que deberían estar.

También se observó que los datos que se obtienen en el “eje y” entre 1.5m y 2m casi no varían en la aplicación de ajuste y en la aplicación final los puntos graficados o Joints del Skeletal se posicionan de manera correcta sobre la imagen capturada por Kinect.

Por lo cual se determinó que la distancia entre el sensor Kinect y el usuario debe estar en el rango entre 1.8 a 2m, para que las aplicaciones funcionen correctamente.

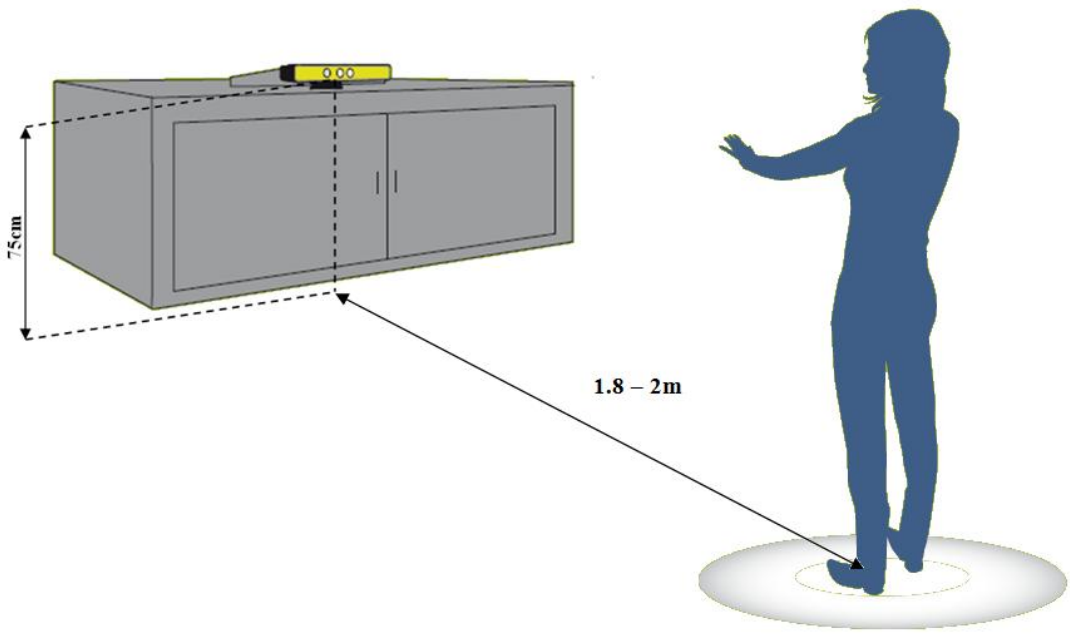


FIGURA 70 Distancia Kinect – Usuario[70]

Se procedió a realizar las mismas pruebas al usuario en este rango de distancia, para comprobar el funcionamiento de la aplicación, las siguientes imágenes muestran los resultados obtenidos a una distancia de 1.8m.

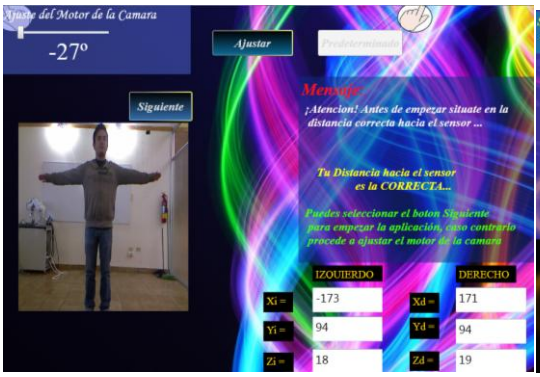


FIGURA 71 Distancia Kinect – Usuario 1.8m (a) [71]

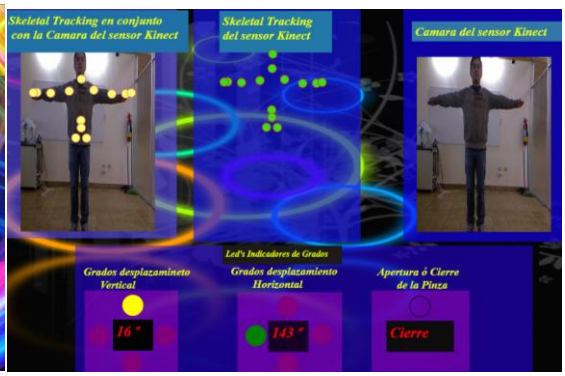


FIGURA 72 Distancia Kinect – Usuario 1.8m (b)[72]

[70] FIGURA 70 REALIZADA POR Autores Tesis
[71] FIGURA 71 REALIZADA POR Autores Tesis
[72] FIGURA 72 REALIZADA POR Autores Tesis

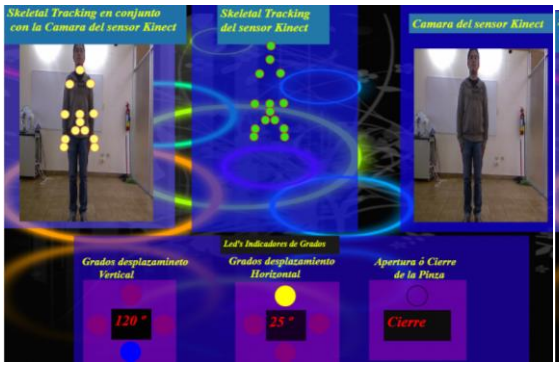


FIGURA 73 Distancia Kinect – Usuario 1.8m (c) [73]

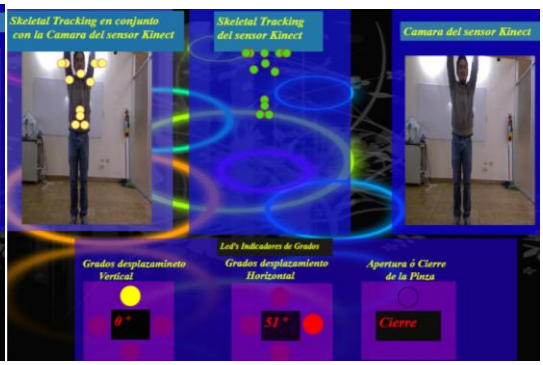


FIGURA 74 Distancia Kinect – Usuario 1.8m (d)[74]

Al estar a esta distancia el sensor Kinect reconoce todo el Skeletal con facilidad y coloca los puntos del Skeletal el lugar correcto sobre la imagen del usuario, teniendo un mejor control sobre la aplicación.

Nota:

A la aplicación de ajuste se le agregó mensajes que le avisan al usuario si se encuentra en la posición correcta o incorrecta, si se encuentra a menos de 1.8m del sensor saldrá un mensaje advirtiéndolo, también si se encuentra a mas de 2m del sensor se le advertirá que está demasiado lejos del sensor, y cuando se encuentre entre 1.8m y 2m aparecerá el siguiente mensaje “tu distancia hacia el sensor es la CORRECTA”.

4.1.2. Estatura del Usuario

Con los siguientes datos se quiere observar las distintas variaciones de comportamiento de la aplicación demostrativa para personas con diferentes alturas, y mediante estos determinar el alcance de movimiento de la aplicación para cada caso.

Las pruebas fueron realizadas a 3 usuarios diferentes, ubicados a una distancia de 1.80m del sensor Kinect, las estaturas de los usuarios no son las mismas, con

[73] FIGURA 73 REALIZADA POR Autores Tesis
[74] FIGURA 74 REALIZADA POR Autores Tesis

lo cual se quiere comprobar si la aplicación realizada responde de igual manera con todos los usuarios y cuales con los cambios más significativos que existen entre cada uno de ellos al momento de manipular la aplicación.

A cada uno de los usuarios se le pidió que movieran los brazos a 4 posiciones diferentes: brazos abiertos, arriba, abajo, y al frente, a continuación se comparará cada uno de los datos obtenidos en las diferentes posiciones entre los usuarios.

- **Usuario 1 (Estatura 1.75m)**

  Para un usuario con esta altura resulta muy fácil controlar la aplicación, si se observa los datos en y cuando tiene las manos abajo y arriba y se los compara se nota que tendrá una mayor movilidad en esta eje ya que se tomaran datos desde -8 hasta 274, y debido que en la aplicación está configurado para que solo se envíen los datos desde 0 a 120 en el "eje y" no existirá perdida de datos al momento de realizar una acción con la pinza robótica cuando la mueva hacia arriba o abajo.

  Los datos que muestran las manos en x solo es importante el de la mano derecha ya que con este es que se controla el movimiento de la pinza en dirección horizontal. Los datos que se envían a los servomotores en este sentido van de 0 hasta 180 grados, en las diferentes pruebas se observa que el usuario tiene una alcanza una medida máxima de 173 cuando tiene los brazos abiertos, la cual será el ángulo de desplazamiento que tendrá en el servomotor, las medidas de la mano derecha en x para este usuario será de 0 a 173 cuando utilice la aplicación ya que aunque en las imágenes no se vea que la mano derecha tome un valor menor a 41 este valor puede seguir bajando mientras se siga moviendo la mano hasta el lado izquierdo.

Los datos en z son importantes para el uso de la pinza robótica ya que esta se abre cuando la mano izquierda llega a 13 y se cierra cuando tiene un valor mayor o igual a18. Para un usuario de esta altura será fácil controlar que se abra o cierre la pinza ya que cuanto extiende sus brazos hacia el frente alcanza sin problemas la distancia 13 necesaria para abrir la pinza y cuando regrese la mano esta se cerrará.

En las imágenes se muestra al usuario colocando las manos en las diferentes posiciones indicadas.

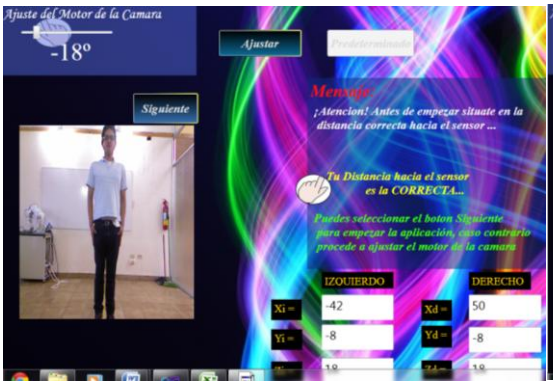


FIGURA 75 Usuario 1 (a)[75]

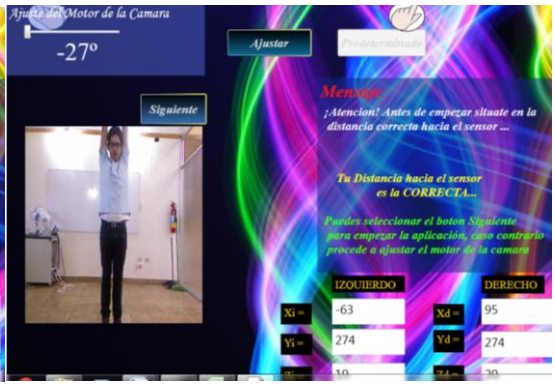


FIGURA 76 Usuario 1 (b) [76]

FIGURA 77 Usuario 1 (c)[77]

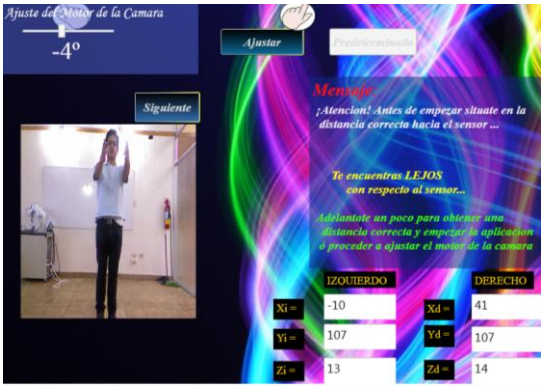


FIGURA 78 Usuario 1 (d) [78]

- **Usuario 2 (Estatura 1.65m)**

[75] FIGURA 75 REALIZADA POR Autores Tesis
[76] FIGURA 76 REALIZADA POR Autores Tesis
[77] FIGURA 77 REALIZADA POR Autores Tesis
[78] FIGURA 78 REALIZADA POR Autores Tesis

Al igual que para el usuario de 1.75m para un usuario con altura de 1.65m resulta muy fácil controlar la aplicación.

Como se mencionó anteriormente solo aremos referencia a los datos tomados de la mano derecha ya que con este es que se controla el movimiento de la pinza en dirección horizontal. Este usuario muestra un rango de movimiento de la mano derecha entre 20 y 147, si el usuario se queda en esa posición, o no se mueve más hacia la derecha con su cuerpo no podrá realizar el movimiento completo de 0 a 180 grados del servomotor, sino que solo logrará mover al servomotor desde 0 hasta 103 grados, pero esto no es un problema ya que como se mencionó si el usuario necesita realizar el movimiento completo de la pinza en el eje horizontal solo es necesario que se desplace hacia la derecha o a la izquierda y Kinect tomará los datos en el lugar donde se encuentre la mano en ese momento.

Para los datos del "eje y" se observa e las imágenes que cuando tiene las manos abajo y arriba se tienen valores de -33 y 210 respectivamente, y debido que en la aplicación está configurada para que solo se envíen los datos desde 0 a 120 en el "eje y" no existirá perdida de datos al momento de realizar una acción con la pinza robótica cuando la mueva hacia arriba o abajo.

Los datos en z son importantes para el uso de la pinza robótica ya que esta se abre cuando la mano izquierda llega a 13 y se cierra cuando tiene un valor mayor o igual a18. Para un usuario de esta altura será fácil controlar que se abra o cierre la pinza ya que cuanto extiende sus brazos hacia el frente alcanza sin problemas la distancia 13 necesaria para abrir la pinza y cuando regrese la mano esta se cerrará.

En las imágenes se muestra al usuario colocando las manos en las diferentes posiciones indicadas.

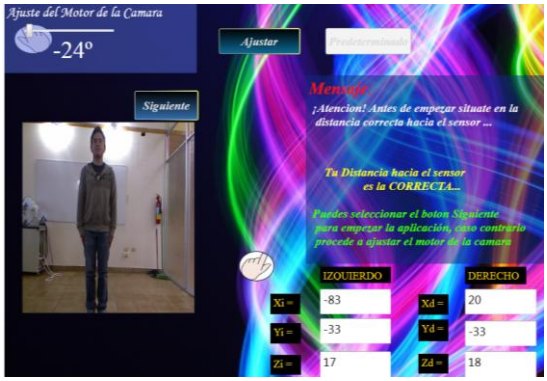


FIGURA 79 Usuario 2 (a) [79]

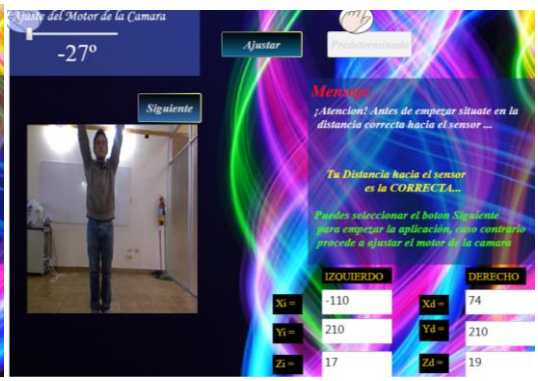


FIGURA 80 Usuario 2 (b) [80]

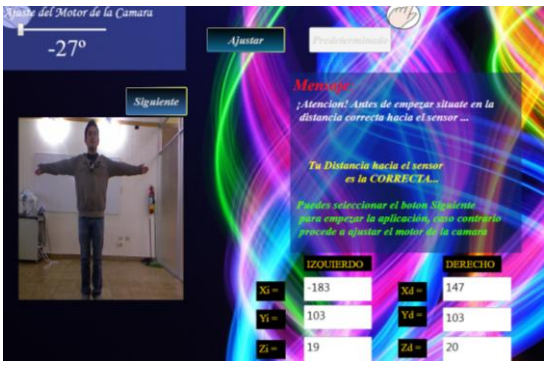


FIGURA 81 Usuario 2 (c) [81]

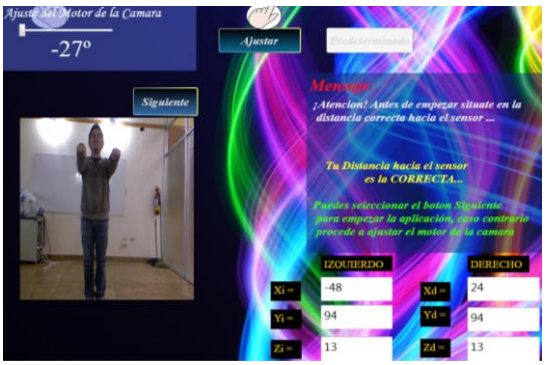


FIGURA 82 Usuario 2 (d) [82]

- **Usuario 3 (Estatura 1.20m)**

Para un usuario con una estatura pequeña como esta fue un poco difícil utilizar la aplicación de ajuste, pero a pesar de demorarse un tiempo más largo para controlar la aplicación, no fue imposible que realizara todos los movimientos necesarios para la aplicación.

Los datos en el eje x muestran que este usuario tiene un rango de movimiento de la mano derecha entre 28 y 113, pero al igual estos datos varían si el usuario se mueve de esa posición, si se mueve más hacia la derecha o a la

[79] FIGURA 79 REALIZADA POR Autores Tesis
[80] FIGURA 80 REALIZADA POR Autores Tesis
[81] FIGURA 81 REALIZADA POR Autores Tesis
[82] FIGURA 82 REALIZADA POR Autores Tesis

izquierda con su cuerpo podrá realizar el movimiento completo de 0 a 180 grados del servomotor.

Para los datos del “eje y” se observa e las imágenes que cuando tiene las manos abajo y arriba se tienen valores de -68 y 117 respectivamente, esto es un problema ya que no alcanza el valor de 180 que es el valor máximo de movimiento de la pinza robótica en el “eje y”, lo que significa que un usuario de este tamaño solo tendrá un ángulo de desplazamiento vertical en la pinza robótica de 0 a 117 grados, lo cual no será completo.

Los datos en z tampoco serán los necesarios para poder abrir la pinza ya que si se queda a 1.80 m de distancia del sensor kinect, el valor en z es 16 el que no es el valor mínimo necesario para poder abrirla. Si el usuario se mueve a una distancia de 1.5m con respecto al sensor kinect ya podrá ser capaz de abrir la pinza, como se muestra en la imagen siguiente.

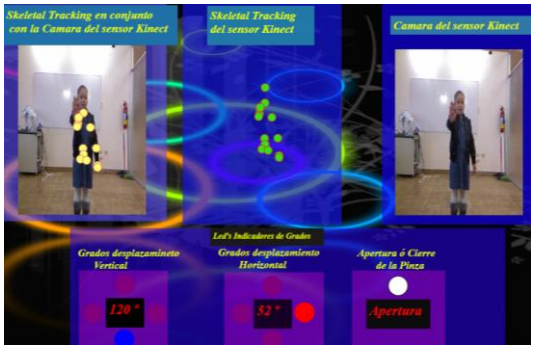


FIGURA 83 Usuario 3 [83]

En las imágenes se muestra al usuario colocando las manos en las diferentes posiciones indicadas.

[83] FIGURA 83 REALIZADA POR Autores Tesis

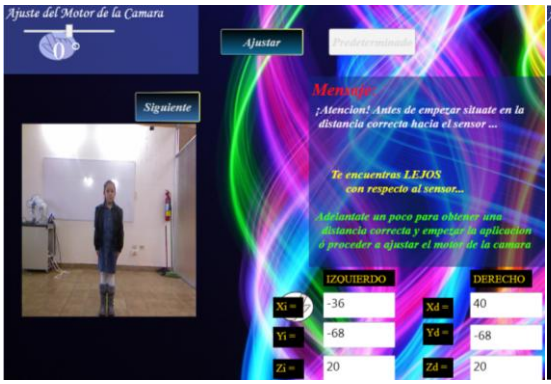


FIGURA 84 Usuario 3 (a) [84]

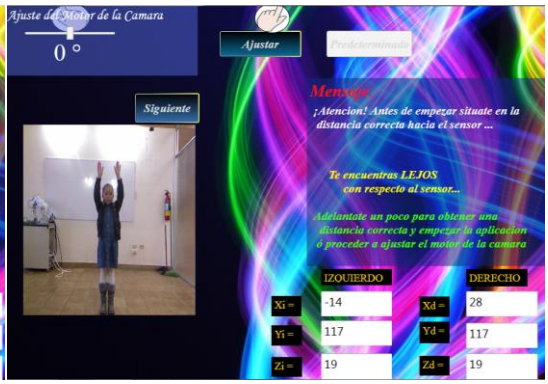


FIGURA 85 Usuario 3 (b)[85]

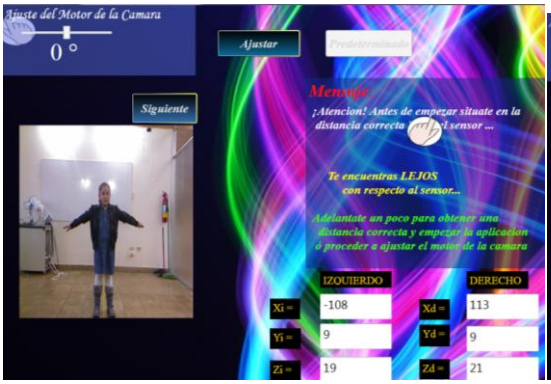


FIGURA 86 Usuario 3 (c) [86]

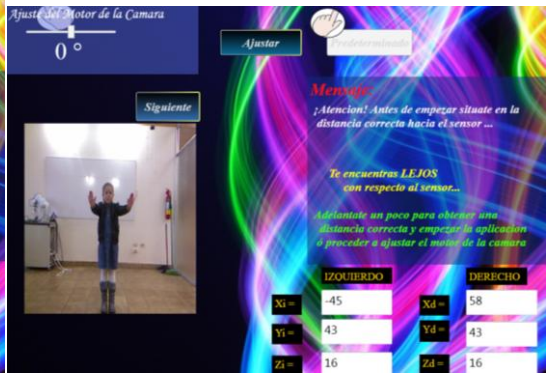


FIGURA 87 Usuario 3 (d)[87]

En la tabla siguiente se muestra los datos obtenidos con las diferentes pruebas que se realizaron a cada uno de los usuarios:

Estatura del Usuario						
	Datos Obtenidos con Brazos Abajo					
Estatura	**X**		**y**		**z**	
	Brazo izquierdo	**Brazo Derecho**	**Brazo izquierdo**	**Brazo Derecho**	**Brazo izquierdo**	**Brazo Derecho**
1.75m	-42	50	-8	-8	18	18
1.65m	-83	20	-33	-33	17	18
1.20m	-36	40	-68	-68	20	20
	Datos Obtenidos con Brazos Arriba					
Estatura	**X**		**y**		**z**	
	Brazo	**Brazo**	**Brazo**	**Brazo**	**Brazo**	**Brazo**

[84] FIGURA 84 REALIZADA POR Autores Tesis
[85] FIGURA 85 REALIZADA POR Autores Tesis
[86] FIGURA 86 REALIZADA POR Autores Tesis
[87] FIGURA 87 REALIZADA POR Autores Tesis

	izquierdo	Derecho	izquierdo	Derecho	izquierdo	Derecho
1.75m	-63	95	274	274	18	20
1.65m	-110	74	210	210	17	19
1.20m	-14	28	117	117	19	19
	Datos Obtenidos con Brazos Abiertos					
Estatura	**X**		**y**		**z**	
	Brazo izquierdo	**Brazo Derecho**	**Brazo izquierdo**	**Brazo Derecho**	**Brazo izquierdo**	**Brazo Derecho**
1.75m	-152	173	93	93	18	19
1.65m	-183	147	103	103	19	20
1.20m	-108	113	9	9	19	21
	Datos Obtenidos con Brazos Al frente					
Estatura	**X**		**y**		**z**	
	Brazo izquierdo	**Brazo Derecho**	**Brazo izquierdo**	**Brazo Derecho**	**Brazo izquierdo**	**Brazo Derecho**
1.75m	-10	41	107	107	13	14
1.65m	-48	24	94	94	13	13
1.20m	-45	58	43	43	16	16

Tabla 6 Estatura Usuario

Nota: En el desarrollo de la aplicación demostrativa se pudo verificar que los distintos usuarios tenían problemas con el ángulo de posición de la cámara.

Solución:

Se procedió a colocar una aplicación que permita ajustar el ángulo del motor del sensor Kinect y situarlo a la altura deseada por el usuario, mediante el movimiento de la mano izquierda y se verificó que Kinect permite mover el ángulo de elevación del motor de la cámara ± 30 grados, por motivos de seguridad la aplicación fue diseñada para que el usuario pueda mover en un rango de ± 27 grados.

4.1.3. Números de usuarios frente al sensor Kinect

Ya que la aplicación fue diseñada para un solo usuario estando frente al sensor Kinect, se pretende analizar el funcionamiento de la aplicación cuando existen dos usuarios frente al sensor.

Cuando existen dos usuarios frente a Kinect la aplicación reconoce al que entró primero en el rango de visión de la cámara como muestra la figura.

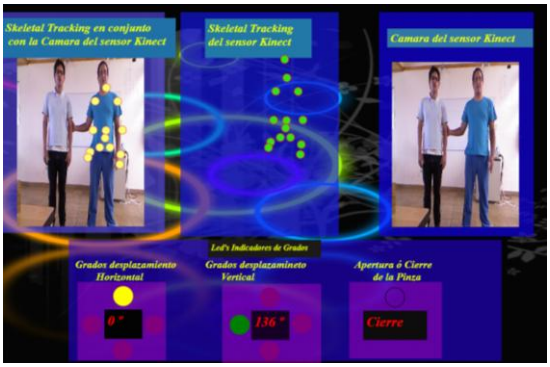


FIGURA 88 2 Usuarios (a)[88]

Ahora si el segundo usuario que entro empieza a moverse y pasa por atrás del primer usuario hasta acercarse a la cámara, de igual manera Kinect sigue reconociendo solo al primer usuario, y obtiene los puntos del Skeleton sobre la imagen del primer usuario como se muestra en la imagen.

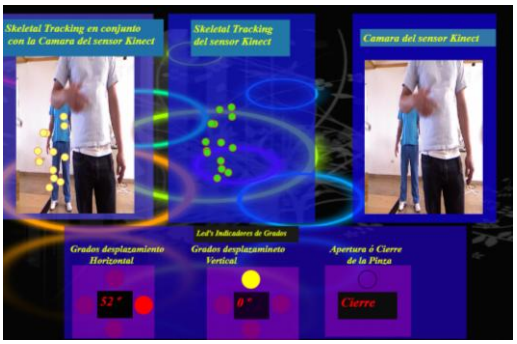


FIGURA 89 2 Usuarios (b)[89]

[88] FIGURA 88 REALIZADA POR Autores Tesis
[89] FIGURA 89 REALIZADA POR Autores Tesis

Pero si el segundo usuario pasa frente al primero y empieza a realizar cualquier movimiento Kinect deja de reconocer al primer usuario y ahora ubica los puntos del Skeleton sobre el segundo usuario, pasando a ser este el nuevo encargado en controlar la aplicación.

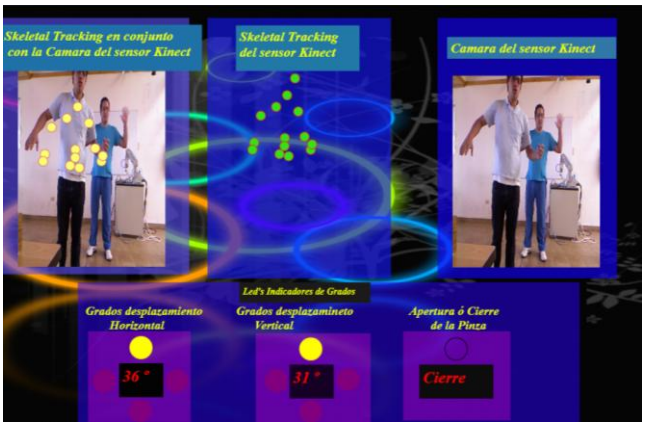


FIGURA 90 2 Usuarios (c)[90]

Nota: Debido a que Kinect reconoce a varios usuarios se recomienda que para la manipulación de este proyecto solo una persona esté frente al sensor Kinect cuando esté utilizando la aplicación demostrativa.

[90] FIGURA 90 REALIZADA POR Autores Tesis

CAPITULO 5

5.1. Conclusiones

- Mediante el análisis de los códigos fuentes SDK de Kinect Beta 2, fue posible implementar una aplicación demostrativa, que capta los movimientos de los brazos y manos del cuerpo humano y los reproduce mediante el movimiento de la pinza robótica, de esta manera se logró la intercomunicación entre el lenguaje de programación C# con las distintas funcionalidades que presenta el sensor Kinect.

- Se desarrolló el análisis de los códigos fuentes SDK de Kinect, Skeletal Traking, Depth Data y Camera Fundamentals, en donde se especifican en forma detallada las funciones características de cada uno de ellos, esta información servirá como referencia técnica para los estudiantes de la Carrera de Ingeniería Electrónica de la Universidad Politécnica Salesiana que quieran innovar en nuevos procesos de automatización. El estudio de los códigos están realizados por segmentos o partes, en donde cada uno de ellos se encarga de la adquisición y el procesamiento de la información enviada por el sensor. El sensor de la cámara RGB da colores a las imágenes capturadas, el sensor 3D permite determinar a qué distancia se encuentra el usuario hacia el mismo y el Skeletal Tracking permite enlazar las diferentes articulaciones del cuerpo humano en puntos o "Joints" y de esta manera se puede graficar el esqueleto humano.

- Se obtuvieron las coordenadas de los puntos o “Joints” en los ejes X, Y, Z de las articulaciones del cuerpo humano cuando se realiza el barrido horizontal, apertura y cierre de brazos y manos, para esto se recurrió a la función Skeletal Tracking que permitió almacenar los mismos y mediante el lenguaje de programación C# se logró comunicar los datos adquiridos en la aplicación realizada, para que de esta manera cuando una persona realice un movimiento frente a Kinect sea mostrado.

- Se realizó una interfaz la cual permite al usuario ajustar la mejor posición del ángulo del motor de la cámara comprendido en un rango de -27 a 27 grados en forma vertical y por medio del sensor de profundidad se implementó un cuadro de mensajes que indica la distancia óptima de posicionamiento entre el usuario y Kinect, además se utilizó la cámara RGB del sensor para detectar los diferentes puntos o Joints y así poder desplegar la imagen de las acciones que realiza el usuario.

- por el puerto serial en el lenguaje de programación C# se logró enviar los datos adquiridos en la aplicación demostrativa de las coordenadas de los ejes en X, Y, Z a través de un vector tipo Byte de tres posiciones # y la lectura de los mismos se realizo a través de la Tarjeta Arduino UNO la misma que permite controlar del movimiento de los servo motores que conforman la pinza robótica , permitiendo así un movimiento de 0 a 120 grados en el eje Y, de 0 a 180 grados en el eje X y de 90 a 147 grados en el eje Z, para facilidad del usuario.

- Se logró la comunicación mediante la implementación de una Pinza Robótica vía USB con el uso de la Tarjeta Arduino UNO, que cuenta con pines de salidas digitales, salidas PWM, se programó el microcontrolador Atmega328P de la tarjeta para recibir y enviar los datos por comunicación serial, con esto se controló el encendido y apagado de Led's y el movimiento de los servo motores, de acuerdo a la acción que realice el usuario con sus manos y brazos.

5.2. Recomendaciones

- Se recomienda tener una distancia entre el usuario y el sensor de 1.80m a 2m, para que el sensor capte correctamente los datos del usuario al momento de realizar los distintos movimientos con sus extremidades superiores (Brazos y Manos).

- La interfaz gráfica fue desarrollada bajo la resolución 940 x 640 pixeles, es recomendable que al momento de utilizar la aplicación la resolución de pantalla de la computadora en que se ejecute el programa sea de 1280 x 720 pixeles o superior.

- La aplicación demostrativa fue desarrollada en el Lenguaje de Programación C#, en el cual se agregaron las referencias de las librerías Microsoft.Research.Kinect.Nui Versión 1.0.0.45 y Coding4Fun.Kinect.Wpf versión 1.1.0.0, ya que si se utilizan librerías con versiones que no sean las antes mencionadas, la aplicación demostrativa (Pinza Robótica) no tendrá un correcto funcionamiento.

- La aplicación demostrativa fue desarrollada con la versión de Microsoft Kinect SDK Beta2, no se asegura que la interfaz de la aplicación demostrativa se ejecute de manera adecuada en diferentes versiones que la indicada.

- Debido a que Kinect detecta varias personas y objetos, la aplicación demostrativa fue desarrollada para que solo un usuario esté al frente del sensor y este sea captado para poder controlar la pinza robótica, si existiera la presencia de mas usuarios el sensor Kinect no podrá reconocer al usuario que lo maneja, tornándose impreciso en la captación de los datos del movimiento del cuerpo humano.

- Es recomendable que entre el Kinect y el usuario no existan elementos u objetos que obstaculicen el campo de visión del sensor Kinect hacia el usuario.

- No se recomienda realizar movimientos bruscos con los brazos al momento de manipular la pinza robótica, ya que los elementos que están implementados en el desarrollo de la misma pueden desconectarse o sufrir algún daño.

BIBLIOGRAFÍA

- **Arduino:**

http://www.arduino.cc/en/Main/arduinoBoardUno

http://rua.ua.es/dspace/bitstream/10045/11833/1/arduino.pdf

http://docs-asia.electrocomponents.com/webdocs/0e8b/0900766b80e8ba21.pdf

http://es.scribd.com/archive/plans?doc=62962340

http://www.atmel.com/Images/doc8161.pdf

http://arduino.cc/es/Reference/Board

http://arduino.cc/en/Main/ArduinoBoardUno

- **Kinect:**

http://es.wikipedia.org/wiki/Kinect

http://msdn.microsoft.com/en-us/library/hh855347.aspx

http://blogs.msdn.com/b/esmsdn/archive/2011/07/07/sdk-de-kinect-desarrolla-con-kinect.aspx

http://blogs.msdn.com/b/esmsdn/archive/2011/07/20/reto-kinect-usar-las-c-225-maras-del-sensor.aspx

http://blogs.msdn.com/b/esmsdn/archive/2011/08/08/reto-sdk-de-kinect-detectar-poses-con-skeletal-tracking.aspx

http://blogs.msdn.com/b/esmsdn/archive/2011/08/22/reto-sdk-kinect-reconocer-gestos-con-skeletal-tracking.aspx

ANEXOS

- **Anexo 1**

Depth Data

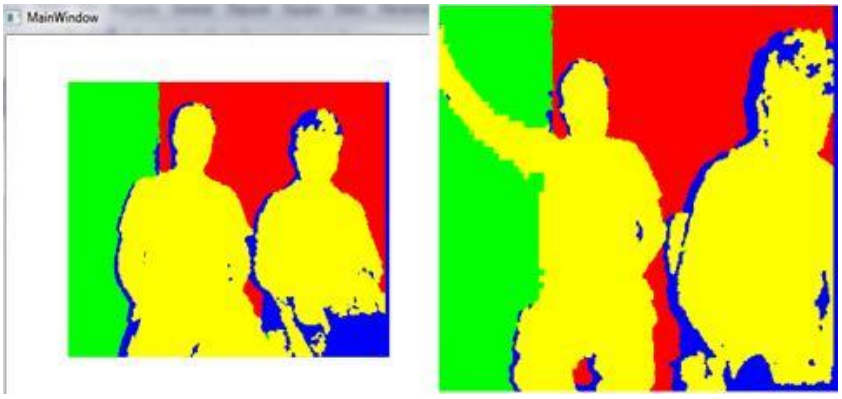


FIGURA 91 Ejecución del programa Depth Data para C# perteneciente a los códigos fuentes de coding4fun de la versión Beta 2 del SDK de Kinect[91]

- **Anexo 2**

Cámara Fundamental

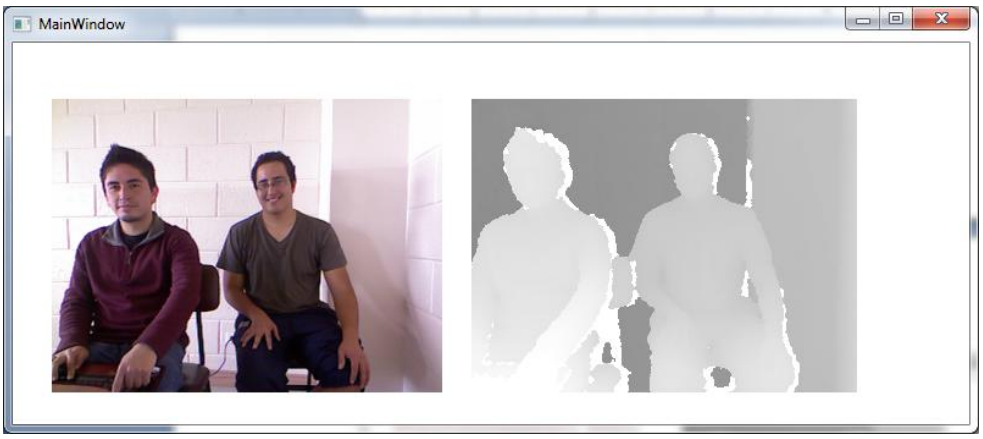


FIGURA 92 Ejecución del programa Cámara Fundamental para C# perteneciente a los códigos fuentes de coding4fun de la versión Beta 2 del SDK de Kinect[92]

[91] FIGURA 91 REALIZADA POR Autores Tesis
[92] FIGURA 92 REALIZADA POR Autores Tesis

- **Anexo 3**

Skeletal Tracking

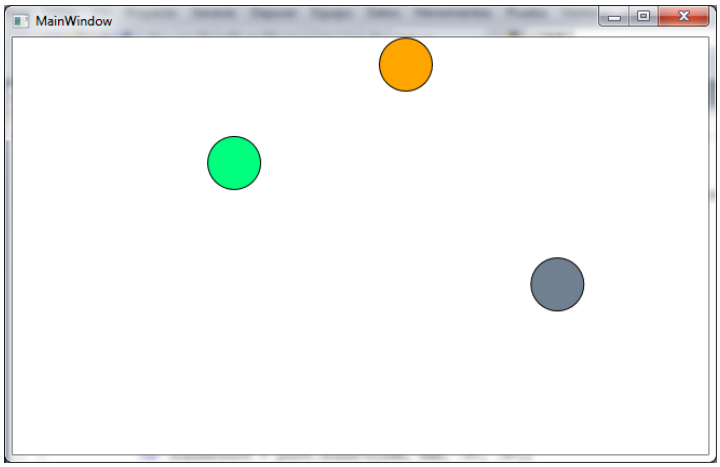
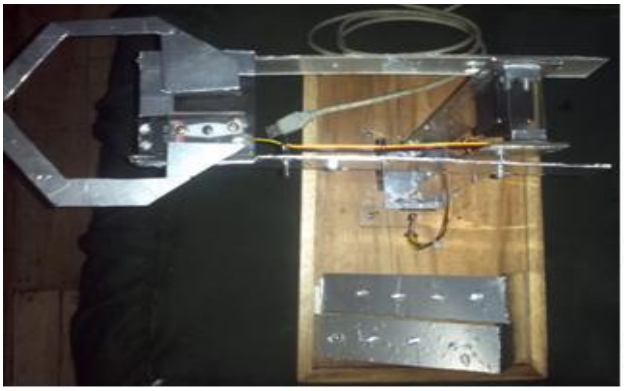


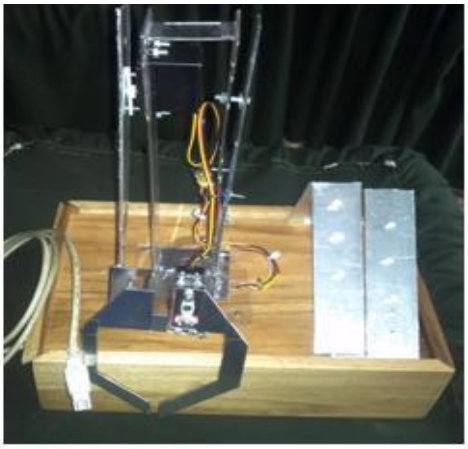
FIGURA 93 Ejecución del programa Skeletal Tracking para C# perteneciente a los códigos fuentes de coding4fun de la versión Beta 2 del SDK de Kinect[93]

- **Anexo 4**

Vista Superior y Frontal de la Pinza Robótica



Vista Superior



Vista Frontal

FIGURA 94 Vista Superior y Frontal de la Pinza Robótica [94]

[93] FIGURA 93 REALIZADA POR Autores Tesis
[94] FIGURA 94 REALIZADA POR Autores Tesis

➢ **Anexo 5**

Diagrama Esquemático Arduino UNO

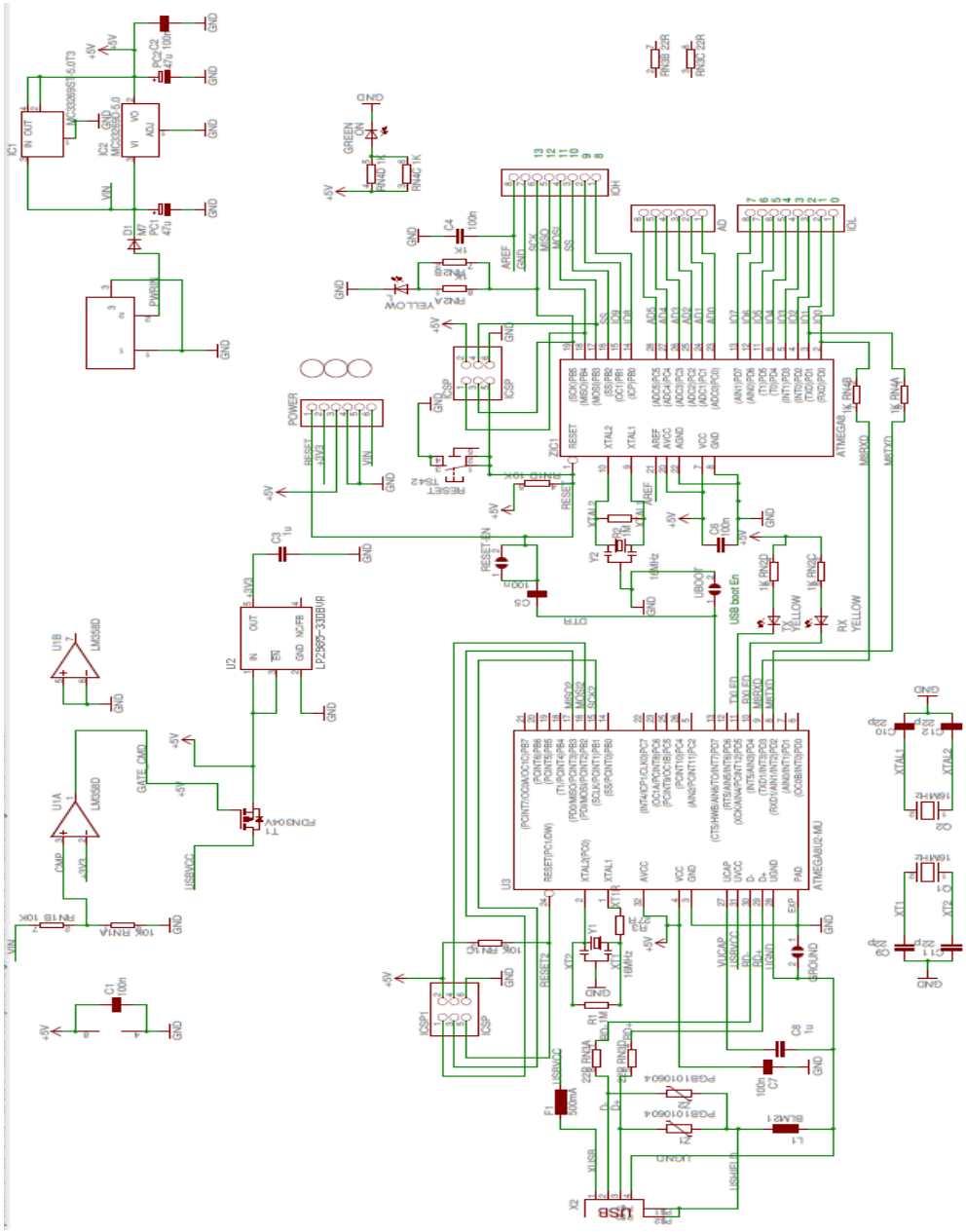


FIGURA 95 Diagrama esquemático de la tarjeta Arduino UNO [95]

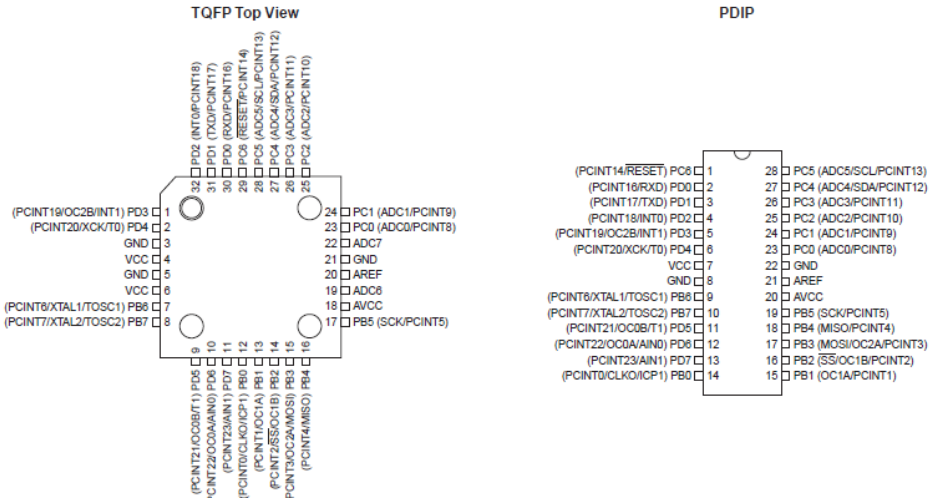
[95] FIGURA 95 REALIZADA POR Autores Tesis

➢ **Anexo 6**

Datasheet Atmega328P

1. Pin Configurations

Figure 1-1. Pinout ATmega48PA/88PA/168PA/328P



28.2.4 ATmega328P DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.[2]	Max.	Units
I_{CC}	Power Supply Current[1]	Active 1 MHz, V_{CC} = 2V		0.3	0.5	mA
		Active 4 MHz, V_{CC} = 3V		1.7	2.5	mA
		Active 8 MHz, V_{CC} = 5V		5.2	9	mA
		Idle 1 MHz, V_{CC} = 2V		0.04	0.15	mA
		Idle 4 MHz, V_{CC} = 3V		0.3	0.7	mA
		Idle 8 MHz, V_{CC} = 5V		1.2	2.7	mA
	Power-save mode[3][4]	32 kHz TOSC enabled, V_{CC} = 1.8V		0.8	1.6	µA
		32 kHz TOSC enabled, V_{CC} = 3V		0.9	2.6	µA
	Power-down mode[3]	WDT enabled, V_{CC} = 3V		4.2	8	µA
		WDT disabled, V_{CC} = 3V		0.1	2	µA

Notes:
1. Values with "Minimizing Power Consumption" enabled (0xFF).
2. Typical values at 25°C. Maximum values are test limits in production.
3. The current consumption values include input leakage current.
4. Maximum values are characterized values and not test limits in production.

28.5 System and Reset Characteristics

Table 28-3. Reset, Brown-out and Internal Voltage Characteristics[1]

Symbol	Parameter		Min	Typ	Max	Units
V_{POT}	Power-on Reset Threshold Voltage (rising)		1.1	1.4	1.6	V
	Power-on Reset Threshold Voltage (falling)[2]		0.6	1.3	1.6	V
SR_{ON}	Power-on Slope Rate		0.01		10	V/ms
V_{RST}	$\overline{RESET}$ Pin Threshold Voltage		0.2 V_{CC}		0.9 V_{CC}	V
t_{RST}	Minimum pulse width on $\overline{RESET}$ Pin				2.5	µs
V_{HYST}	Brown-out Detector Hysteresis			50		mV
t_{BOD}	Min Pulse Width on Brown-out Reset			2		µs
V_{BG}	Bandgap reference voltage	V_{CC}=2.7 T_A=25°C	1.0	1.1	1.2	V
t_{BG}	Bandgap reference start-up time	V_{CC}=2.7 T_A=25°C		40	70	µs
I_{BG}	Bandgap reference current consumption	V_{CC}=2.7 T_A=25°C		10		µA

Notes: 1. Values are guidelines only.
2. The Power-on Reset will not work unless the supply voltage has been below V_{POT} (falling)

28.8 ADC Characteristics

Table 28-7. ADC Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
	Resolution			10		Bits
	Absolute accuracy (Including INL, DNL, quantization error, gain and offset error)	V_{REF} = 4V, V_{CC} = 4V, ADC clock = 200 kHz		2		LSB
		V_{REF} = 4V, V_{CC} = 4V, ADC clock = 1 MHz		4.5		LSB
		V_{REF} = 4V, V_{CC} = 4V, ADC clock = 200 kHz Noise Reduction Mode		2		LSB
		V_{REF} = 4V, V_{CC} = 4V, ADC clock = 1 MHz Noise Reduction Mode		4.5		LSB
	Integral Non-Linearity (INL)	V_{REF} = 4V, V_{CC} = 4V, ADC clock = 200 kHz		0.5		LSB
	Differential Non-Linearity (DNL)	V_{REF} = 4V, V_{CC} = 4V, ADC clock = 200 kHz		0.25		LSB
	Gain Error	V_{REF} = 4V, V_{CC} = 4V, ADC clock = 200 kHz		2		LSB
	Offset Error	V_{REF} = 4V, V_{CC} = 4V, ADC clock = 200 kHz		2		LSB
	Conversion Time	Free Running Conversion	13		260	µs
	Clock Frequency		50		1000	kHz
AV_{CC}[1]	Analog Supply Voltage		V_{CC} - 0.3		V_{CC} + 0.3	V
V_{REF}	Reference Voltage		1.0		AV_{CC}	V
V_{IN}	Input Voltage		GND		V_{REF}	V
	Input Bandwidth			38.5		kHz
V_{INT}	Internal Voltage Reference		1.0	1.1	1.2	V
R_{REF}	Reference Input Resistance			32		kΩ
R_{AIN}	Analog Input Resistance			100		MΩ

Note: 1. AV_{CC} absolute min/max: 1.8V/5.5V

FIGURA 96 Diagrama esquemático de la tarjeta Arduino UNO [96]

[96] FIGURA 96 Tomada de http://www.dz863.com/datasheet-8322888163-ATMEGA328_8-bit-Avr-Microcontroller-With-4-8-16-32k-Bytes-In-system-Programmable-Flash/

- **Anexo 7**

Especificaciones técnicas Servomotor Hitec HS-311

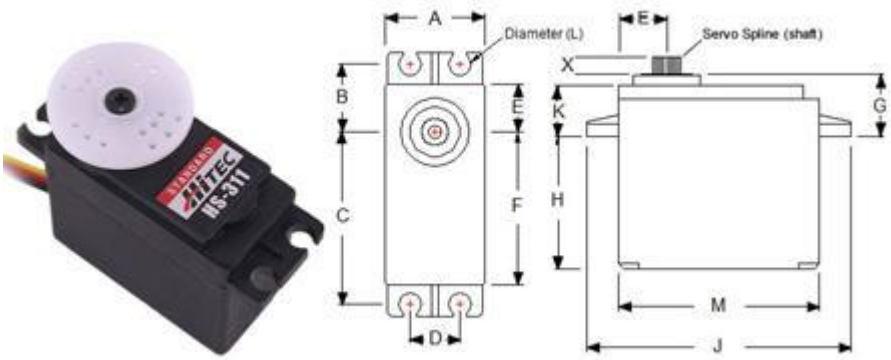


FIGURA 97 Servomotor HS-311 Standard[97]

" Control System: **+Pulse Width Control 1500usec Neutral**
Required Pulse: **3-5 Volt Peak to Peak Square Wave**
Operating Voltage: **4.8-6.0 Volts**
Operating Temperature Range: **-20 to +60 Degree C**
Operating Speed (4.8V): 0.19sec/60° at no load
Operating Speed (6.0V): 0.15sec/60° at no load
Stall Torque (4.8V): 42 oz/in (3.0 kg/cm)
Stall Torque (6.0V): 51 oz/in (3.7 kg/cm)
Current Drain (4.8V): **7.4mA/idle, 160mA no load operating**
Current Drain (6.0V): **7.7mA/idle, 180mA no load operating**
Dead Band Width: **5usec**
Operating Angle: **45° one side pulse traveling 450usec**
Direction: **Multi-directional**
Motor Type: **Cored Metal Brush**
Potentiometer Drive: **4 Slider/Direct Drive**
Bearing Type: **Top Resin Bushing**
Gear Type: **Nylon**
360 Modifiable: **Yes**
Connector Wire Length: **11.81" (300mm)**
Weight: **1.52oz (43g)" [2]**

[97] FIGURA 97 Tomado de http://www.servocity.com/html/hs-311_standard.html
[2] TOMADO DE http://www.servocity.com/html/hs-311_standard.html

- **Anexo 8**

Instalación de los SDK de KINECT

El software que se empleo para el desarrollo del proyecto de titulación fue la versión Kinect for Windows SDK BETA 2 para Windows 7 x86. Cabe recalcar que en el transcurso del desarrollo de esta tesis Windows ha lanzado una nueva versión Kinect for Windows SDK v1.5 y posiblemente seguirá lanzando nuevas versiones.

Los pasos descritos son para la Versión de Kinect BETA 2:

- ✓ Poseer un Kinect Xbox-360
- ✓ No conectarlo a la fuente de corriente ni al ordenador.
- ✓ Tener instalados en el ordenador Windows 7(x86 o x64) y Visual Studio 2010(Cualquier Versión)
- ✓ Proceder a descargar el paquete Kinect for Windows SDK BETA2 correspondiente al sistema operativo que se emplee (32bits o 64 bits) de la pagina web www.kinectforwindows.org
- ✓ Seguir los pasos del instalador

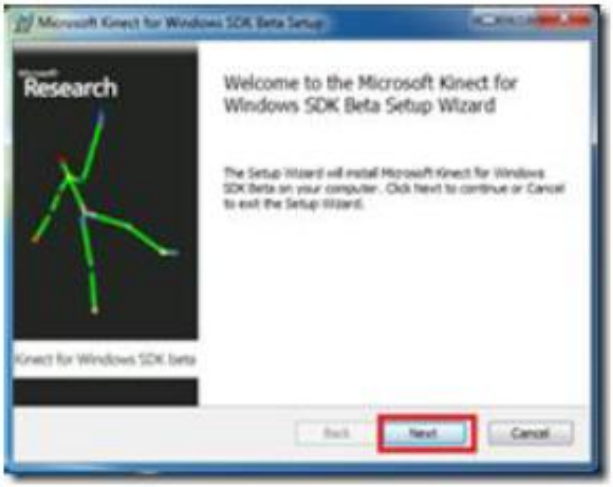


FIGURA 98 Instalador Kinect[98]

[98] FIGURA 98 Tomada de Autores Tesis

Una vez Instalado el SDK para Windows se procederá a conectar el Kinect a la fuente de corriente y a conectar su dispositivo USB al ordenador, el sistema operativo reconocerá el nuevo Hardware y procederá a instalarlo automáticamente.

Si se desea se puede dar Click en mas información para ver el estado de la instalación.

FIGURA 35 Kinect Listo [99]

Realizar la respectiva comprobación de la correcta instalación de Kinect dirigiéndose a "Panel de Control/Sonido y Hardware /Administrador de Dispositivos" y ahí aparecerá el sensor Instalado.

FIGURA 100 Dispositivo Kinect Conectado[100]

Cabe recalcar que para proceder a la Instalación se debe de desinstalar versiones anteriores a SDK de Kinect y tener Visual Studio 2010 cerrado cualquier aplicación.

[99] FIGURA 99 Tomada de Autores Tesis
[100] FIGURA 100 Tomada de Autores Tesis

- **Anexo 9**

Instalación de ARDUINO UNO en el Sistema Operativo Windows

Lo primero que se necesita es comprar una tarjeta Arduino y un cable USb standard ("A-to-B" plug si se usa una placa Arduino Uno).

Arduino funciona en Windows, Mac OS X, y en Linux, por lo que hay una versión para cada tipo de sistema operativo, para obtenerlo se necesita ir a la página oficial de Arduino http://arduino.cc/en/Main/Software y descargar la versión del software compatible con el sistema operativo.

- *Pasos para la Instalación*

Primero se debe descargar el archivo Windows.zip y descomprimirlo. Se creará una carpeta llamada Arduino-1.0.1 Esta es la carpeta necesaria para poder acceder al programa y puede ser instalada en cualquier carpeta del computador, no necesariamente en archivos de programa.

Antes de empezar a trabajar con la placa es necesario instalar los drivers de Arduino. Para la placa Arduino Uno se tiene que seguir estos pasos:

Conectar la tarjeta a la computadora, y esperar que Windows inicie el proceso de instalación del driver. Después de un tiempo saldrá un error en la instalación, debido a que es necesario instalar manualmente los drivers.

Dar click en inicio y abrir el panel de control.

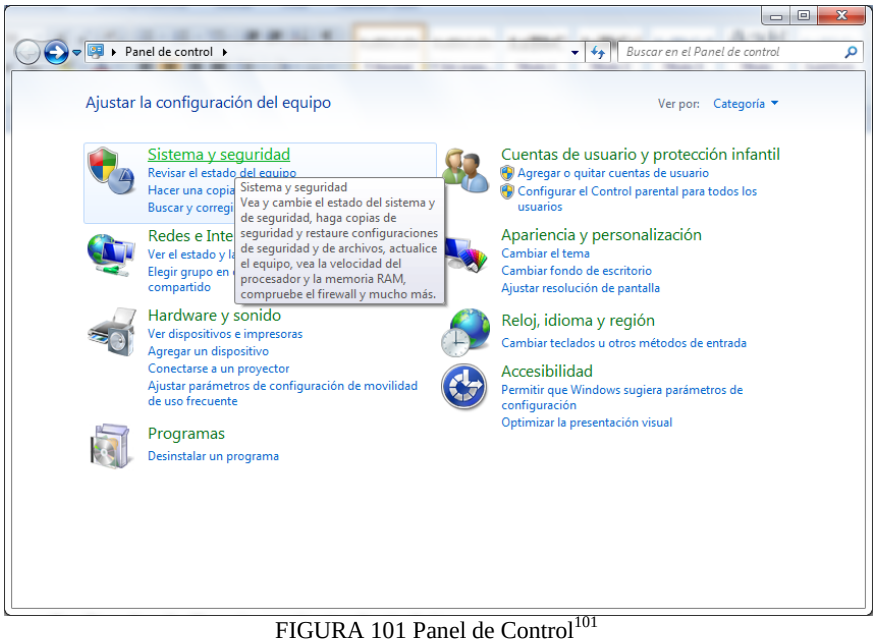


FIGURA 101 Panel de Control[101]

Ir a Sistema y Seguridad, luego a Sistema, y de ahí a Administrador de Dispositivos.

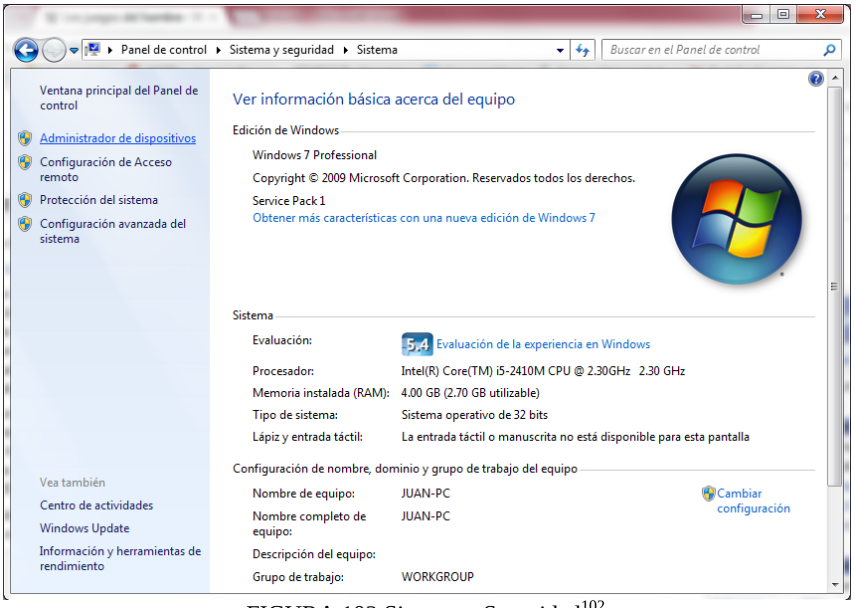


FIGURA 102 Sistema y Seguridad[102]

En el administrador de dispositivos buscamos los Puertos (COM y LPT) se ve que no existe uno para Arduino, abrimos otros dispositivos y aparece uno como dispositivo desconocido o como Arduino.

[101] FIGURA 101 Tomada de Autores Tesis
[102] FIGURA 102 Tomada de Autores Tesis

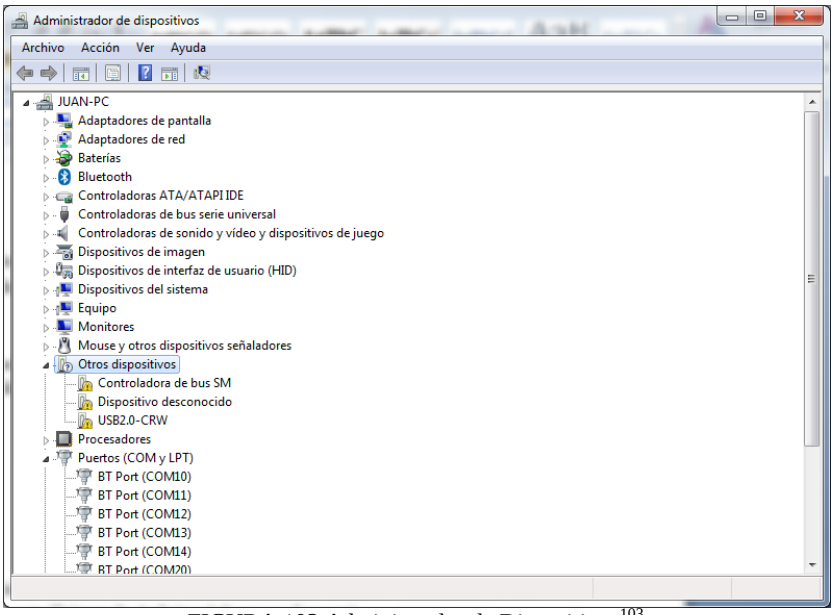


FIGURA 103 Administrador de Dispositivos[103]

Dar click derecho sobre el dispositivo desconocido y seleccionar "Actualizar Software de Dispositivo" elegir "buscar software de controlador en el equipo".

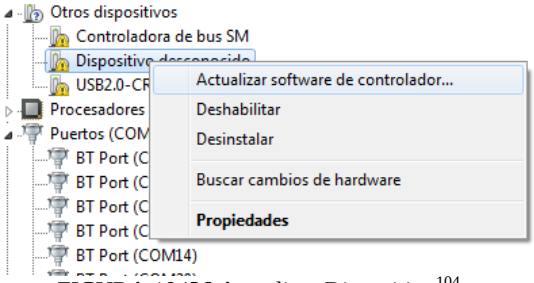


FIGURA 10436 Actualizar Dispositivo[104]

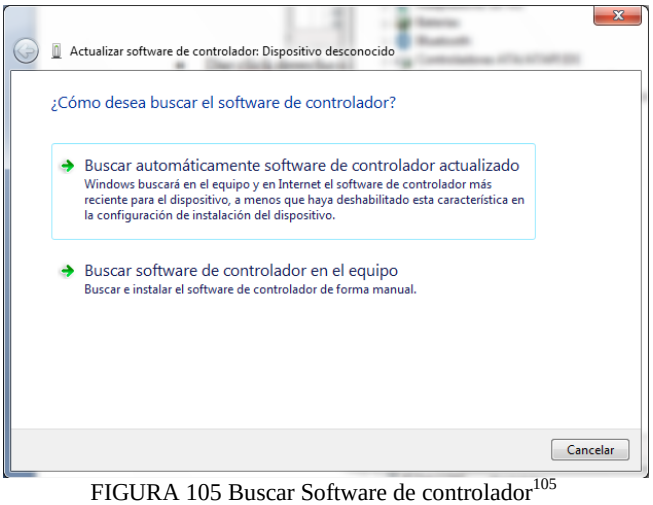


FIGURA 105 Buscar Software de controlador[105]

[103] FIGURA 103 Tomada de Autores Tesis
[104] FIGURA 104 Tomada de Autores Tesis
[105] FIGURA 105Tomada de Autores Tesis

Finalmente, seleccionar el archivo del Driver de Arduino Uno llamado "ArduinoUNO.inf" localizado en la carpeta Drivers de Arduino-1.0.1 que se descargó.

Ahora Windows lo reconocerá.

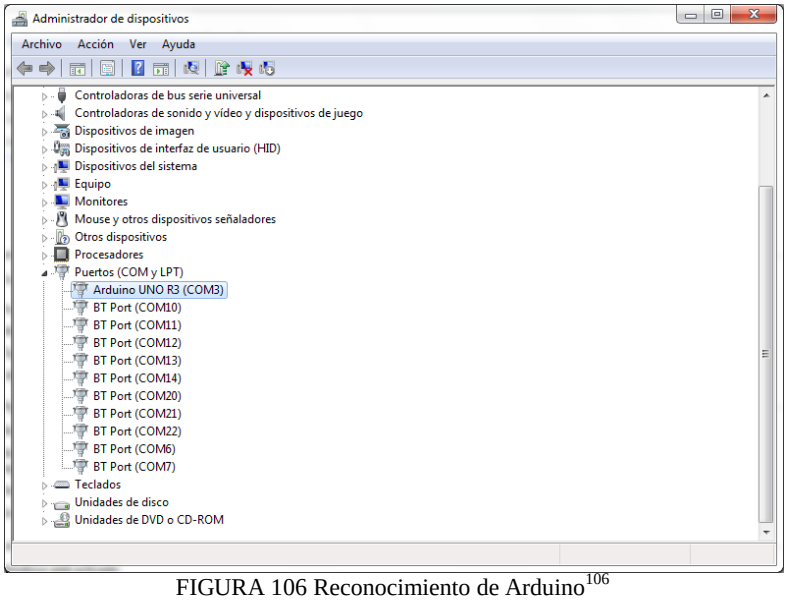


FIGURA 106 Reconocimiento de Arduino[106]

[106] FIGURA 106 Tomada de Autores Tesis