Tecnológico Nacional de México

Instituto Tecnológico de Tuxtla Gutiérrez

Departamento de Ingeniería Eléctrica y Electrónica

Ciclo Escolar Enero - junio 2018

López Liévano José Alejandro

Carrera:
Ingeniería Electrónica

Asesor interno:
Alvaro Hernandez Sol

Asesor externo:
Gerardo Luis Velazquez Garcia

Reporte final de Residencia Profesional

Nombre del Proyecto:

“Diseño Y Programacion De Interfaz Para La Manipulacion De Un Robot Humanoide A Traves De Kinect”

ÍNDICE

Objetivos

Diseñar la interfaz para controlar un robot humanoide para controlar robot integrado por 4 grados de libertad y una pinza de sujeción, programado para moverse de acuerdo a los movimientos hechos por el brazo del usuario de forma

simultánea, dichos movimientos del usuario serán adquiridos a través del sensor Microsoft Kinect .

Específicos

El diseño, de la interfaz y programación del robot humanoide en conjunto, es considerado un sistema en tiempo real, por lo que el movimiento del brazo debe realizarse en un tiempo que sea visualmente paralelo al tiempo en el que se mueve el brazo perteneciente al usuario. Los objetivos específicos establecidos para realizar el trabajo se definen a continuación:

- Adquisición de datos de profundidad y de usuario por medio del sensor Kinect, utilizados para procesar y capturar cada movimiento que realiza el usuario con el brazo derecho.

- Desarrollar una interfaz gráfica de usuario que permita visualizar la ejecución del brazo del usuario, desplegando los grados desplazados por cada articulación.

- Programación del microcontrolador ATmega328 dedicado a recibir los datos de la computadora, datos que contienen los grados a desplazarse de cada motor y poder así generar el movimiento de cada uno de los motores, de forma simultánea, bajo el concepto de sistema multitarea.

JUSTIFICACIÓN

La presente propone la programación de una interfaz para el control de un robot que sea controlado mediante el movimiento del brazo del usuario a través de una interfaz de usuario basada en el dispositivo Microsoft Kinect, el robot humanoide debe imitar en tiempo real el movimiento del brazo del usuario, mejorando el tiempo de respuesta, precisión y facilidad para controlar los brazos manipuladores. El brazo robótico es construido para ayudar en tareas que sustituyan el trabajo que comúnmente es realizado por el contacto directo de la mano del usuario y el objeto, en una determinada labor, evitando así accidentes que son causados por dicha actividad. El brazo robótico también ayudará a realizar trabajos que tal vez no sean peligrosos pero dependen de una herramienta que realice movimientos similares a los ejecutados por el brazo del cuerpo humano, siendo el brazo del usuario el control para colocar el robót humanoide al punto deseado, además se pretende innovar el control de brazos manipuladores utilizando un dispositivo de última tecnología como Kinect, remplazando dispositivos que requieren más recursos o son difíciles de trabajar en forma conjunta.

CAPITULO 1. GENERALIDADES

1.0.1 Información de la institución donde se desarrolló el proyecto

INSTITUTO DE CAPACITACIÓN EN MANUFACTURA Y AUTOMATIZACIÓN

En ICMA se ofrecen cursos de capacitación, asesoría en control y automatización de procesos, especialmente relacionados con la industria eléctrica, mecánica y automotriz.

1.0.2 Misión

Capacitar en automatización y control de procesos de estudiantes y profesionales en las áreas industriales de electricidad, electrónica y electromecánica, atravesó de un aprendizaje personalizado y practico, logrando su inserción exitosa en la industria.

1.0.3 Visión

Consolidarnos a nivel regional como la empresa número uno en capacitación evaluación y certificación en automatización control diseño y manufactura por computadora, respaldado en la calidad certificado de nuestro egresados y su desempeño en el sector productivo del estado de puebla .

1.0.4 Valores

Los valores que nos caracterizan son superación continua , espíritu de equipo respeto mutuo vocación de servicio integridad y responsabilidad .

1.0.5Ubicación

CAPÍTULO 2. FUNDAMENTOS TEÓRICOS

2.01 Control de un Robot ABB con Kinect

Estudiantes de Ingeniería Robótica que en el verano de 2011, en Suecia, usaron el software de seguimiento de esqueleto del Kinect, para controlar un brazo robótico ABB [1]. En la figura 2.1 se puede ver como manejan el robot con el Kinect, en

donde apilan una serie de bloques de madera para después escribir utilizando el robot.

Figura 2.1 Robot ABB moviendo piezas de madera, controlado por Kinect [1].

2.02 Brazo robótico controlado mediante gestos a través de Kinect

En la Universidad Tsukuba (Ibaraki, Kanto, Japón), fue presentado el “Sistema de Brazo Robot Controlado por Gestos”, que tiene la capacidad de comprender los movimientos tanto del brazo en general, como de la mano y sus dedos. El sistema necesita de dos cámaras, encargadas de transmitir en tiempo real información sobre el movimiento y la forma de la mano, para después ser duplicados en la réplica robótica. Un detalle el cual ya parece haber sido resuelto por sus desarrolladores es que no todas las manos humanas son iguales, por esta razón cargaron al sistema con una gran base de datos repleta de manos, de forma que sólo es necesario encontrar una mano similar en forma y tamaño para que el brazo robot pueda operar sin inconvenientes [1].

En este caso las dos estados del arte fueron hechas en el mismo documento consultado, eso no es valido, tienes que referenciar al documento individual de cada uno de ellos.

2.03 Uso médico del Kinect

Estudiantes de la Universidad de Washinton (EE.UU.) han querido ir más allá. Han adaptado la tecnología para realizar cirugías robóticas asistidas en la vida real. Este método implica el uso de Kinect para ayudar a los cirujanos a utilizar las herramientas cuando realicen una cirugía. En la actualidad, los cirujanos suelen

utilizar la robótica para intervenciones quirúrgicas mínimamente invasivas. El principal problema, con los métodos actuales, es que los cirujanos no tienen ninguna forma de palpar lo que están haciendo. Si se mueve un instrumento quirúrgico en algo sólido, el instrumento se detendrá pero el mando de control sigue en movimiento. El equipo de ingeniería eléctrica de la universidad ha resuelto este problema gracias al código abierto de Kinect para asignar y reaccionar a los entornos en tres dimensiones y enviar la información sobre el entorno al usuario. Al usar Kinect se establecen "espacios electrónicos" que restringen el movimiento de la herramienta quirúrgica, es decir, si la herramienta toca un hueso se deja de mover. En cambio, si el instrumento se mueve a lo largo de un hueso, el joystick sigue el mismo camino. Incluso es posible definir las zonas de acceso restringidas para proteger los órganos vitales. El equipo espera que esta cirugía robótica sea fiable y práctica a largo plazo, permitiendo así a los médicos llevar a cabo fácilmente en las principales ciudades cirugías en pacientes en pueblos pequeños y aislados. Chizeck asegura que esta idea se puede extrapolar en catástrofes o guerras. En un hospital de Canadá han mostrado un nuevo ejemplo del uso de Kinect para mantener la sala de operaciones como un entorno esterilizado. La idea es que se realicen determinadas tareas sin salir del quirófano, como puede ser la búsqueda de información sobre el estado de un paciente en un ordenador. La realización de un escáner por rayos X, ya que podría manejar de forma remota el ordenador que controla el aparato. Además de eso, se evitaría tener que lavarse y desinfectarse, como sucede ahora cada vez que utiliza esa máquina [1].

Capítulo 3 Marco teórico

3.01 Sistemas en tiempo real

Es un sistema electrónico de procesamiento que interactúa con el medio, controla procesos, y entrega resultados dentro de tiempos establecidos de manera confiable y segura.

El principal objetivo de un sistema en tiempo real es controlar o realizar varios procesos de forma simultánea, lo que se conoce como multitarea. Estos procesos o tareas duran un tiempo determinado, dependiendo la aplicación o la determinación del usuario, y son atendidos por un mismo procesador o CPU (unidad central de procesamiento). Al tiempo que dedica el procesador a cada tarea se le conoce como prioridad, esta se define dependiendo la aplicación o el uso que tenga [2].

La figura 3.1 muestra la representación de forma general de un sistema de procesamiento digital, en el cual intervienen mínimo dos variables y se obtiene una respuesta en un tiempo acorde a la necesidad del usuario.

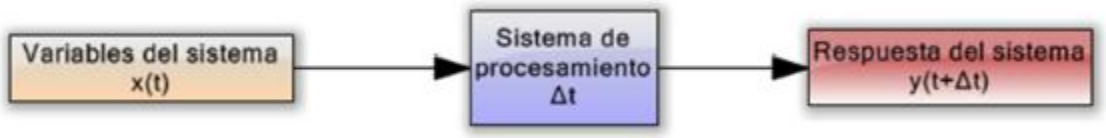

Figura 3.1 Diagrama a bloque de un sistema de procesamiento digital.

En el primer bloque se representa todas las variables que interactúan con el sistema, en el caso de un sistema electrónico, estas variables pueden ser sensores debido a que estos interactúan con la naturaleza o el medio, y además dependen del tiempo. El segundo bloque representa el proceso que se encarga de controlar todas las variables del sistema y realizar determinados procesos. El último bloque representa la respuesta del sistema, la cual debe ser entregada por el sistema en un tiempo que no afecte algún proceso o sea catastrófico para el sistema.

En un sistema en tiempo real intervienen los siguientes factores para el tiempo de respuesta:

- Físicos

-Capacitancias, inductancias, resistencias de los circuitos, etc
-Velocidad de propagación de los medios de comunicación digital
-Velocidad de procesamiento
-Etapas del sistema

- Software

-Algoritmo
-Nivel de lenguaje(c++,asm,etc.)

-Compilador

Para los tipos de comunicación digital en un sistema en tiempo real, los más comunes son los siguientes:

- Serie

-USB
-RS232
-RS485
-Ethernet
• Paralelo
-ICE
-PCI
-DB25

- Inalámbricos

-Bluetooth
-Wifi

Para entregar resultados al usuario, los sistemas en tiempo real realizan diferentes procesos que determinan el tiempo de respuesta. En la figura 2.2 se muestra el diagrama a bloques de los procesos que realiza un sistema en tiempo real para entregar una respuesta o resultado, cada bloque representa un proceso que tiene determinada duración en el sistema. El tiempo total de respuesta del sistema está dado por la suma de los tiempos correspondientes a cada etapa, como se determina en la ecuación 3.2.

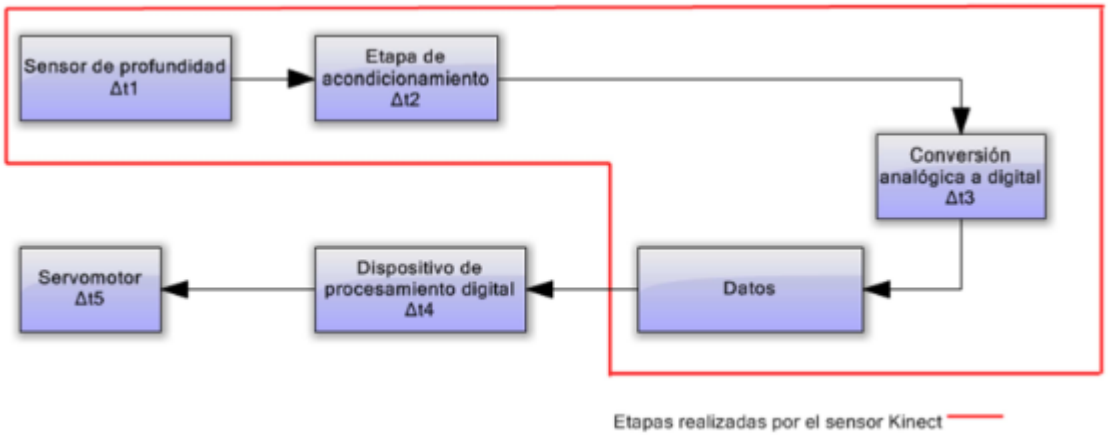

Figura 3.2 Diagrama a bloques de un sistema en tiempo real.

$$\Delta tT = \sum_{i=1}^{5} \Delta ti \quad \text{(Ecuación 2.1)}$$

3.02 Sensores

Una parte fundamental en todo sistema de adquisición de datos es el elemento encargado de percibir la magnitud a medir. Los sensores son dispositivos capaces de convertir una magnitud física como puede ser temperatura, presión, etc., en una diferencia de potencial o una variación de intensidad de corriente, es decir, realizan una conversión de energías y suministran información sobre el estado y tamaño de la magnitud. Los sensores informan de su entorno y además esa información es cuantificable, es decir, medible por algún instrumento.

La posibilidad de que un sensor perciba una determinada magnitud depende de:

- **Que exista una propiedad en algún material que cambie en función desea magnitud**

  Preferiblemente esa función debe ser lineal para el rango en el que estemos interesados, por ejemplo la relación que se da en los conductores entre su resistencia al paso de la corriente eléctrica y la temperatura, es decir a mayor resistencia, mayor temperatura.

- **En otras ocasiones existe una relación entre una magnitud y un fenómeno físico**

  Si la relación es predecible y estable, el fenómeno se puede usar como base para la determinación de la magnitud. En muchas ocasiones la dificultad está en conseguir que la propiedad o fenómeno esté en función únicamente de la magnitud que queremos evaluar. Suele ocurrir que la medida es función de varios factores, de tal manera que cualquier variación en alguno de ellos altera el resultado final.

En la actualidad, la mayor parte de los sensores generan una salida en voltaje o corriente, o bien modifican una propiedad que puede ser evaluada de forma eléctrica. De esta manera, y con el debido acondicionamiento, la señal de salida puede ser tratada por un equipo digital de adquisición de datos. Las señales del mundo real son, en general, analógicas y varían de manera continua en el tiempo, para que una computadora sea capaz de procesarla se debe convertir a datos digitales. Cada uno de estos sensores tiene unas características propias y genera una tensión o intensidad determinada, por lo que estas señales tienen que ser adaptadas para ser tratadas en una tarjeta de adquisición de datos.

En el tratamiento de imagen y sonido, los sensores más utilizados son:

- **Micrófono:** Capta la información sonora que se propaga por el aire y la convierte a una señal eléctrica.
- **Cámara:** Capta la información visual y convierte la información en datos digitales.

3.03 Acondicionamiento de la señal

El objetivo del acondicionador de señal es generar a partir de lo obtenido por los sensores, una señal que sea aceptable por las tarjetas de adquisición de datos. Las funciones principales que va a tener que realizar el acondicionador de señal son las siguientes:

- **Amplificación:** La señal proporcionada por los sensores suele ser de un valor muy pequeño, por lo que debe ser amplificada con el fin de que pueda ser detectada correctamente por la tarjeta de adquisición de datos. La amplificación debe ser tal que las variaciones de la señal recorran todo el margen de la tarjeta de adquisición de datos. La amplificación de las señales, en su origen, reduce el ruido que les puede afectar en su transmisión hasta la computadora.
- **Filtrado**: Con el filtrado se pretende eliminar ruidos de frecuencia que pueden hacer perder exactitud al sistema de adquisición de datos. Lo ideal es transportar la señal del sensor lo más limpia posible a la tarjeta de adquisición.
- **Excitación**: Hay algunos sensores que necesitan de una excitación en corriente o voltaje, para producir la variación proporcional a la magnitud a medir.

Linealización: No todos los sensores tienen una variación lineal con respecto a las variaciones de la magnitud que se miden; a veces es necesario convertir la respuesta del sensor en lineal.

3.04 Clasificación de los sistemas en tiempo real

Los sistemas en tiempo real se pueden dividir según los errores que se puedan presentar:

- **Críticos:** si no se cumplen con los tiempos establecidos se destruye el sistema y/o causa una catástrofe al medio o al usuario.
- **No críticos**: es aquel en el que se puede tolerar que en algunas ocasiones los resultados no se obtengan en los tiempos establecidos, ya que no se degrada el servicio o el proceso que está controlando el sistema, en este tipo se especifica una tasa de probabilidad que permite tolerar las fallas en los tiempos de respuesta.
- **Inflexibles**: no se cumplen con tiempos de respuesta, no ocurre daño al sistema, pero la aplicación queda inservible.

El sistema desarrollado en la presente tesis es clasificado dentro de la categoría de sistemas en tiempo real no críticos, debido a que el sistema puede tolerar que los resultados no se obtengan en los tiempos establecidos, en este caso las imágenes que entrega el sensor son enviadas a la computadora 30 veces en un segundo, por lo cual si alguna imagen no es procesada por la computadora, no afecta sobre el sistema debido a que el sensor genera las imágenes con un tiempo superior a los movimientos del usuario, los cuales son capturados a través del Kinect.

3.05 Descripción del Hardware

El proyecto tiene como base al sensor Microsof Kinect, el cual envía datos que se actualizan constantemente, los cuales determinan los movimientos del brazo robótico cuya estructura cuenta con tres grados de libertad. Cada grado está limitado por el movimiento máximo de un servomotor (0-180 grados) y al ángulo de los movimientos del brazo del usuario, mismos que se encuentran normalmente dentro del mismo rango.

El diagrama de casos de uso que se muestra en la figura 3.3 representa como debe el usuario interactuar con el sistema.

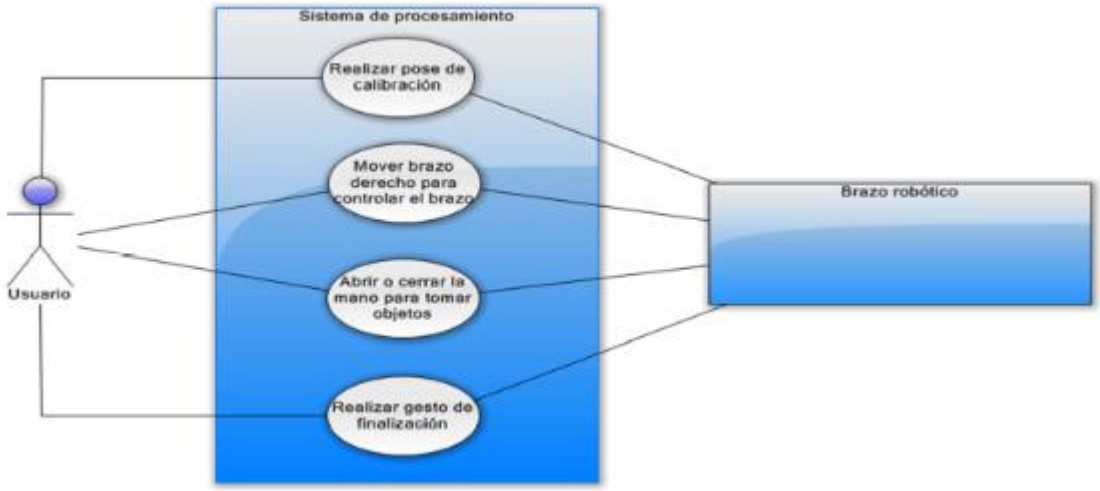

Figura 3. 3 Diagrama de casos de uso para el sistema.

La figura 3.3 muestra el diagrama de casos de uso para el sistema del brazo robótico, el usuario debe interactuar con el sistema en todo momento, realizando alguna de las siguientes tareas:

- Pose de calibración: es necesaria para poder iniciar el movimiento del brazo y poder controlarlo.
- Movimiento del brazo derecho: es el control del brazo robótico, el cual realiza los movimientos en tiempo real que sean realizados por el usuario.
- Apertura o cierre de la mano izquierda: movimiento establecido para tomar (cerrar la mano), o dejar objetos (abrir la mano) utilizando la pinza del brazo robótico.
- Gesto de finalización: gesto predeterminado para finalizar el movimiento del brazo.

3.06 Sensor de profundidad

Basa su funcionamiento en una técnica llamada “luz estructurada”, la cual consiste en la proyección de un patrón de puntos infrarrojos sobre una superficie, al hacer corresponder la imagen captada con el patrón original, es posible triangular la posición de cada píxel y determinar su profundidad con respecto al plano perpendicular a la cámara [7] .

3.07 Detección y seguimiento de usuario

Cuando se adquieren datos a través de la cámara de profundidad, imagen con la información de la distancia de los objetos o usuario, esta imagen es analizada para extraer información que esté relacionada con el usuario, su posición y la pose realizada. El sensor contiene dentro de su software un algoritmo de detección que consiste en dividir el cuerpo del usuario en 31 partes que son reconocidas en un plano tridimensional. Una representación del proceso de detección de usuario se muestra en la figura 3.4.

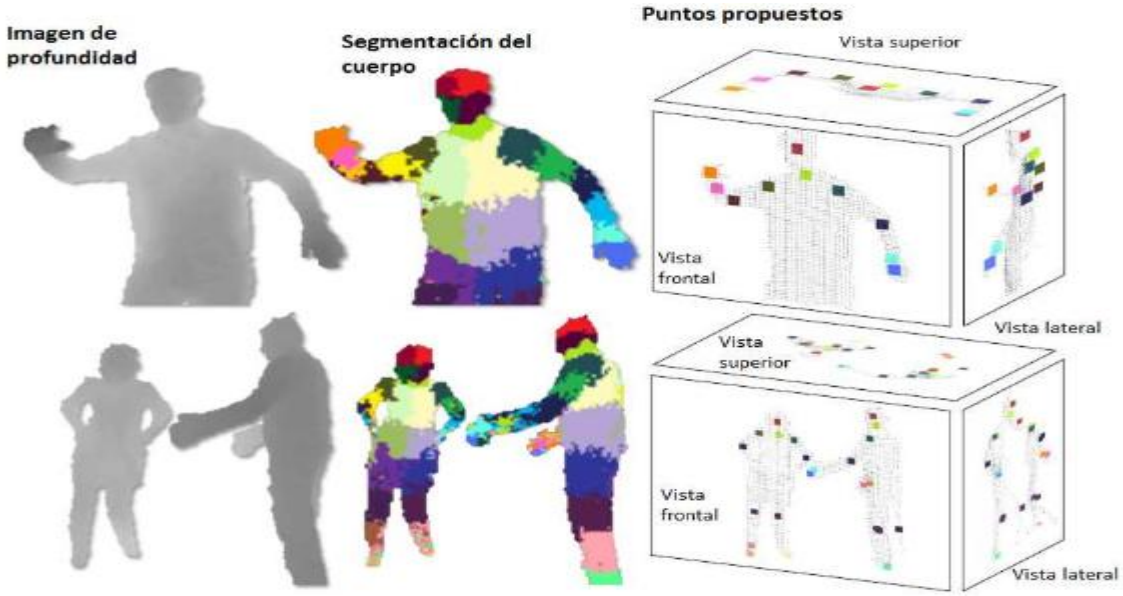

Figura 3. 4 Proceso de segmentación y obtención de articulaciones a partir de imágenes de profundidad [11].

La detección de articulaciones es demasiado precisa, con un 91.4 % de probabilidad de ser localizados correctamente dentro del plano tridimensional.

Las articulaciones o puntos del cuerpo, de los cuales se puede realizar un seguimiento dentro del plano que abarca el sensor, son los siguientes:

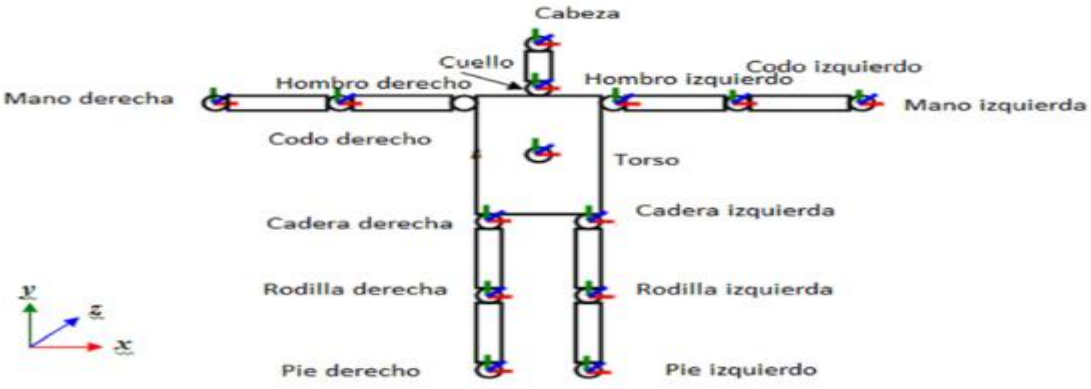

Figura 3. 5 Articulaciones del cuerpo del usuario, de las cuales se puede realizar un seguimiento (x,y,z).

3.08 Limitaciones de la detección de usuario

La detección y seguimiento de usuario que proporciona Kinect es precisa y de cierta forma confiable, aunque también sufre algunas limitaciones. Estas limitaciones son insignificantes cuando la aplicación del sensor es exclusivamente en software, pero cuando se aplica a campos como el de la robótica, estas limitaciones pueden ser vitales para la aplicación. Las principales limitaciones en cuanto a la detección y seguimiento de usuario son las siguientes:

- **No trabaja bajo la luz solar (hardware)**

El sensor Kinect proyecta una luz infrarroja a través de su emisor de 60mW (mili Watts) con una longitud de onda de 830nm (nano metros). La luz solar tiene un ancho espectro de luz infrarroja que afecta a la luz proyectada por el sensor, la luz proyectada es blindada por el brillo de la luz solar, entonces la cámara que detecta la proyección de puntos infrarrojos es deshabilitada para poder detectar los patrones emitidos por el emisor. Esto propicia que exista una muy pobre detección de profundidad en áreas que están expuestas a la luz solar.

- **No detecta usuarios dentro de superficies transparentes y reflejantes (hardware)**

La detección en superficies reflejantes o transparentes es también difícil cuando se utilizan sensores ópticos, como es el caso del sensor Kinect. Este factor se debe a que la mayoría de los sensores ópticos reflejan la luz sobre la superficie, esta reflexión puede ser alta o baja y puede afectar en las lecturas del sensor.

- **La resolución de la cámara limita el reconocimiento de gestos demasiados precisos, como pueden ser gestos realizados con los dedos a determinada distancia (hardware)**

La resolución de la cámara de profundidad se limita a 640*480 pixeles con pocos centímetros de resolución, esta limitante determina que algunos gestos no puedan ser reconocidos por el sensor. Dentro de los gestos que no son complicados y pueden ser detectados por el sensor Kinect se encuentran: realizar un movimiento de onda con la mano, levantar y agitar la mano o realizar un círculo con la misma. Sin embargo si una pequeña área del cuerpo es observada, por ejemplo la mano del usuario, gestos tan específicos como los realizados por esta, no pueden ser reconocidos debido a que la región de interés de la imagen que debe ser analizada, es pequeña en comparación con la imagen del cuerpo.

- **El Kinect debe de mantenerse fijo sobre una base o superficie (hardware)**

Para simplificar la detección y seguimiento de usuario, el sensor debe estar totalmente fijo, esto debido a que cuando el sensor se encuentra sobre una superficie móvil, puede detectar determinados objetos como si estos estuvieran en movimiento y/o confundirlos con siluetas humanas para después entregar resultados incorrectos. La figura 3. muestra los datos erróneos que entregaría el sensor Kinect cuando se encuentre montado sobre una superficie móvil, algunos objetos son detectados como usuarios debido a que cuando el sensor se mueve, la imagen obtenida aparenta el movimiento del objeto.
La solución a este problema es la calibración que lleva a cabo el usuario para confirmar que se trata de una silueta humana.

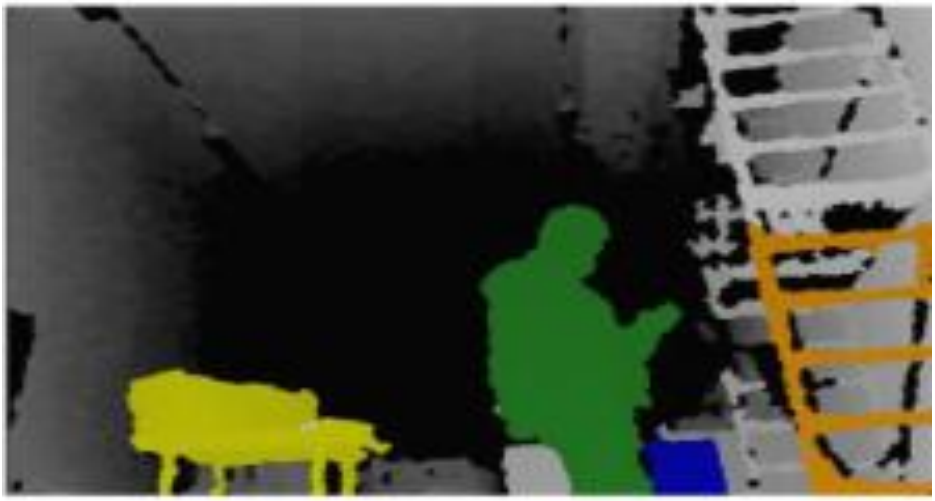

Figura 3.6 Imagen que muestra cuando el sensor se encuentra sobre una superficie móvil, algunos objetos son detectados como usuarios siendo marcados con determinadocolor [12].

3.0.9 Librerías de libre acceso para la programación de Kinect

Cuando el sensor Kinect salió a la venta, no existían controladores ni librerías para habilitar el uso del sensor en una computadora personal, el uso de este sensor se aplicaba exclusivamente a la consola Xbox360. Fue hasta el 2010 que Microsoft liberó el SDK (Kit de desarrollo de software) para Kinect, el cual permite acceder a todas las funciones del sensor: sensor de profundidad, seguimiento de usuario, cámara RGB y acceso al arreglo de micrófonos. Debido a que Microsoft no fue el creador del controlador principal del sensor Kinect, se han desarrollado diferentes librerías de programación de libre acceso compatibles con el sensor Kinect.

En la tabla 3.7 se muestran las diferentes librerías de libre acceso que se han desarrollado para el sensor Microsoft Kinect, algunas son compatibles con distintos lenguajes de programación y plataformas de desarrollo

Nombre	Lenguajes	Plataformas	Características
Open Kinect/ Libfreenect	C,Python, ActionScript, C#,C++, Java JNI and Java JNA, JavaScript, Com-monLisp	Linux, Windows, Mac OS X	Imágenes de color y profundidad, control de la base motorizada y del led.
CL SDK NUI and driver	C, C#	Windows	Imágenes de color y profundidad, control del led y de la base motorizada.
Robot Operating System	Pyhon, C++	UNIX	Imágenes de color y profundidad. Acceso a base motorizada.
OpenNI/NITE Middle-ware	C,C++,C#, Java	Windows, Linux, Ubuntu, Mac OS X	Identificación de usuario, detección de gestos, seguimiento y orientación de articulaciones de usuario, imágenes de color y profundidad.
SDK Microsoft Kinect	C++,C#, Visual		Identificación de usuario, detección de gestos, seguimiento de articulaciones de usuario, imágenes de profundidad y de color.

Tabla 3. 7 Comparación de librerías de libre acceso.

3.1.0Descripción de las librerías de uso libre para programar el sensor

OpenNI (Interfaz natural de uso libre) es un conjunto de librerías multiplataforma y multilenguaje para desarrollar aplicaciones utilizando interfaces naturales como esensor Kinect. Las librerías fueron desarrolladas por una organización llamada “OpenNI” que fue fundada por la empresa Israelí “PrimeSense”, la compañía que otorgó la licencia a Microsoft para construir el sensor Kinect [13]. El principal propósito de OpenNI es establecer un desarrollo de software estándar que habilite la comunicación con:

- Sensores de audio y visión: en esta tesis de utiliza el sensor Microsoft Kinect, pero OpenNI puede utilizarse con el sensor desarrollado por “PrimeSense” llamado “XAzus”.
- Percepción de audio y video(los componentes del software analizan los datos de audio y video que son capturados a través de las imágenes), por ejemplo software que recibe datos visuales como imágenes, regresando la localización de la palma de la mano.

El estándar de OpenNI es aplicado al desarrollo de aplicaciones de interacción natural para realizar un seguimiento dentro de una escena tridimensional, estos datos son capturados por medio de las imágenes de profundidad obtenidas por el sensor. Las librerías de programación de OpenNI se puede describir en tres capas (véase figura 3.8), en donde cada una representa un elemento integro. La capa inferior contiene el hardware o dispositivos que adquieren los datos del mundo real. La segunda capa contiene los componentes de las librerías que interpretan y analizan los datos del sensor. Finalmente la capa superior contiene el software que implementa las aplicaciones de interacción natural.

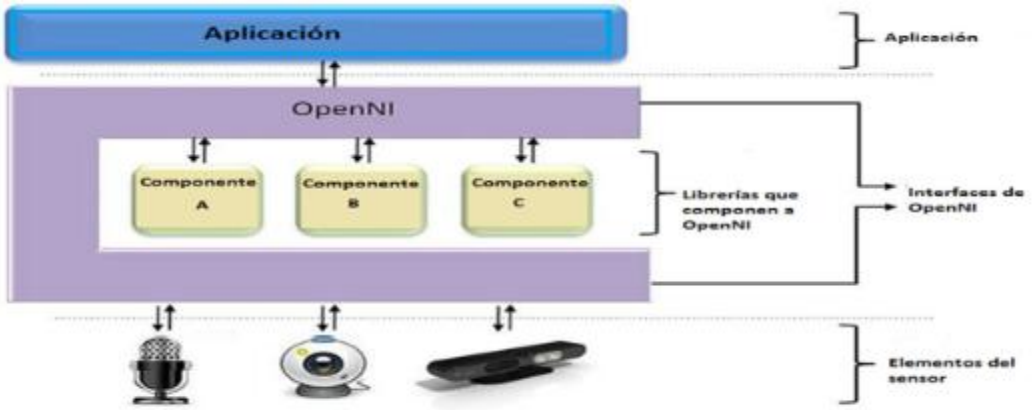

Figura 3.8 Representación abstracta del concepto de OpenNI dividido en tres capas [14].

La forma en la que trabaja OpenNI es a través de la producción de nodos. La producción de nodos es el establecimiento de componentes que forman los datos del sensor que serán utilizados en las aplicaciones.

OpenNI clasifica la producción de nodos en dos categorías:

- Producción de nodos relacionada con el sensor
- Producción de nodos relacionada con las librerías

Los nodos relacionados con el sensor son los siguientes:

- Dispositivo: Un nodo que representa un dispositivo físico (por ejemplo un sensor de profundidad o una cámara RGB), el principal objetivo de este nodo es habilitar el dispositivo que se utilizará.
- Generador de profundidad: Genera mapas de profundidad, este nodo debe ser implementado en algún sensor que capture imágenes en tercera dimensión y esté certificado por la compañía OpenNI.
- Generador de imágenes: Nodo que genera imágenes a color, debe serimplementado en algún sensor que obtenga imágenes a color y estécertificado por la compañía OpenNI.
- Generador infrarrojo: Nodo que genera imágenes infrarrojas.
- Generador de audio: Genera cadenas de audio.

Los nodos relacionados con las librerías de OpenNI son los siguientes:

- Generador de gestos: genera eventos en la aplicación cuando determinados gestos son detectados.
- Analizador de escena: separa el fondo de una escena de un objeto o imagen de usuario, permitiendo trabajar los pixeles de fondo de forma independiente.
- Generador de puntos en la mano: permite reconocer la palma de la mano de un usuario y realizar un seguimiento cuando la mano cambia de posición.
- Generador de usuario: representación del cuerpo(o parte del cuerpo) en una escena tridimensional.

3.1.1 Calibración

Para la producción del nodo de generación de usuario es necesaria una pose por parte del usuario, debido a que antes de realizar este proceso, los datos del usuario que proporciona la cámara de profundidad no están alineados con respecto a los datos de la cámara de color. La calibración se utiliza para poder utilizar los datos de profundidad, junto con los de la imagen de color. También optimiza el seguimiento de usuario, debido a que ofrece una mayor precisión a diferencia de no utilizar la mencionada calibración.

3.1.2 Dispositivos de procesamiento digital

Para poder programar el sensor Kinect se requiere el uso de una computadora personal, debido a que la mayor parte de los datos, se adquieren, procesan y muestran en forma de video (datos digitales), lo cual demanda velocidades de procesamiento altas como las que se llevan a cabo en computadoras personales de reciente tecnología, además de que la adquisición de datos que se obtiene del sensor es a través del puerto USB(bus serial universal) y todas las librerías de desarrollo son únicamente compatibles con sistemas operativos que se ejecutan en computadoras personales y entornos de desarrollo para las mismas (Visual Studio, NetBeans, Eclipse, etc.).

A diferencia del procesamiento para el sensor Kinect, el brazo robótico requiere de un dispositivo ajeno a la computadora personal para generar el movimiento de los servomotores (modulación por ancho de pulso) que conforman la estructura del robot, este dispositivo debe contar con algún protocolo de comunicación que facilite la recepción de datos desde la computadora y los procese para poder generar el movimiento de los servomotores bajo el concepto de sistema multitarea, es decir, de forma simultánea.

3.1.3 Servomotores

El motor es capaz de ubicarse en cualquier posición dentro de un rango de operación (generalmente de 180º) y mantenerse estable en dicha posición. Los servomotores se suelen utilizar en robótica, automatización, debido a su gran precisión en el posicionamiento.

En general, los servomotores suelen estar compuestos por 4 elementos fundamentales:

- **Motor de corriente continua (DC)**: Es el elemento que le brinda movilidad al servomotor. Cuando se aplica un potencial a sus dos terminales, este motor gira en un sentido a su velocidad máxima. Si el voltaje aplicado sus dos terminales es inverso, el sentido de giro también se invierte.
- **Engranajes reductores**: Tren de engranajes que se encarga de reducir la alta velocidad de giro del motor para incrementar su capacidad de torque.
- **Sensor de desplazamiento:** Suele ser un potenciómetro colocado en el eje de salida del servomotor que se utiliza para conocer la posición angular del motor.
- **Circuito de control:** Es una placa electrónica que implementa una estrategia de control de la posición por realimentación. Para ello, este circuito compara la señal de entrada de referencia (posición deseada) con la posición actual medida por el potenciómetro. La diferencia entre la posición actual y la deseada es amplificada y utilizada para mover el motor en la dirección necesaria para reducir el error.

En la figura 3.9 se muestran los 4 elementos mencionados anteriormente, además se muestra la carcasa en donde vienen incluidos estos elementos y en conjunto forman al servomotor.

Figura 3.9 Elementos que conforman un servomotor [16].

3.1.4 Principio de funcionamiento

Los servos disponen de tres cables (Figura 3.10): dos cables de alimentación (positivo y negativo) que suministran un voltaje 4.8-6V y un cable de control que indica la posición deseada al circuito de control mediante señales PWM ("Modulación por ancho de pulsos").

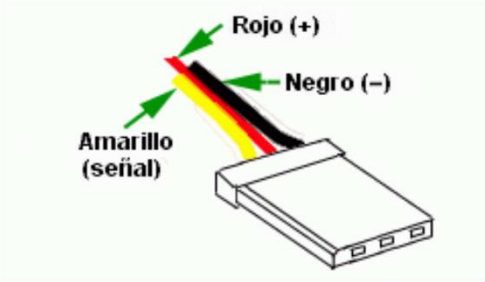

Figura 3.10 Cables de un servomotor [16].

Las señales PWM (modulación por acho de pulso) utilizadas para controlar los servos están formadas por pulsos positivos cuya duración es proporcional a la posición deseada del servo y que se repiten cada 20ms (50Hz). Todos los servos pueden funcionar correctamente en un rango de movimiento de 90º, que se corresponde con pulsos PWM comprendidos entre 0.9 y 2.1ms. Sin embargo, también existen servos que se pueden mover en un rango extendido de 180º y sus pulsos de control varían entre 0.5 y 2.5ms. Antes de utilizar un servo se debe comprobar experimentalmente su rango de movimiento para no dañarlo. Para mantener fijo un servo en una posición habrá que enviar periódicamente el pulso correspondiente; ya que si no recibe señales, el eje del servo quedará libre y se podrá mover ejerciendo una leve presión.

3.1.5 Control de un servomotor

Para controlar un servo, se le ordena un cierto ángulo, medido desde 0 grados, enviándole una serie de pulsos. En un tiempo alto de pulso se le indica el ángulo al que debe posicionarse. Generalmente se considera que en 1.5ms está en el centro (90 grados) .Entre límites de 1 ~ 2ms son las recomendaciones de los fabricantes, normalmente se puede usar un rango mayor de 1.5ms para obtener un ángulo mayor e incluso de 2ms para un ángulo de rendimiento de 180 grados o más. El factor limitante es el tope del potenciómetro y los límites mecánicos construidos en el servo. Un sonido de zumbido normalmente indica que usted está forzando por encima al servo, entonces debe disminuir un poco. En la figura 3.11 se muestra el ciclo de trabajo o también nombrado pulso en alto para que un servomotor se posicione en 90 grados (posición neutra), 0 grados y 180 grados, el ciclo de trabajo para las posiciones de 0 y 180 grados depende del servomotor y del fabricante.

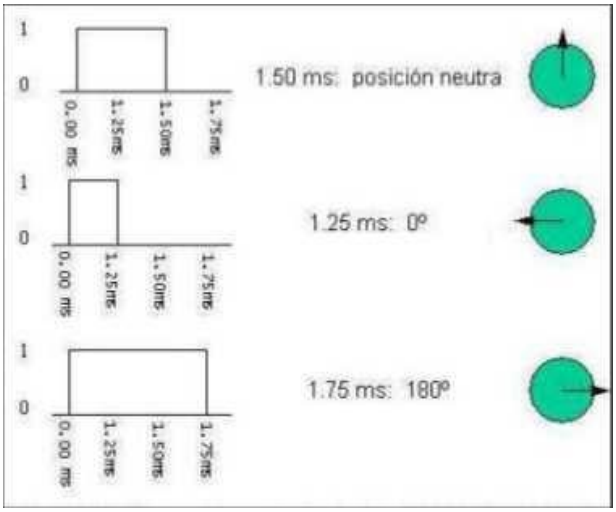

Figura 3.11 Ciclos de trabajo para posicionar un servomotor estándar [16].

3.1.6 Descripción del software

En el diagrama mostrado en la figura 3.12 se marcan de color azul los procesos que serán ejecutados por la computadora, mientras que de color verde los procesos que ejecutará el microcontrolador . A pesar de que algunos procesos son ejecutados por el mismo dispositivo (computadora o sensor Kinect), cada proceso depende de otro y tiene una prioridad asignada para su ejecución.

En nuestro primer proceso o inicio del algoritmo del sistema, se inicializa el sensor, en el segundo proceso se adquieren los datos de profundidad, después se entra en el proceso calibración de usuario, hasta que sea completado de forma correcta se puede trabajar con los datos de usuario. El siguiente paso es el cálculo del movimiento del brazo del usuario, es aquí donde se aplica el algoritmo de seguimiento del brazo que controla el robot. La segunda condición que existe dentro del sistema es la detección del gesto de finalización, dato que ordena al microcontrolador que realice la detención del brazo robótico, esta condición es aplicada dentro de cada ciclo del diagrama de flujo, y procesada antes de enviar el paquete de datos hacia el microcontrolador. El siguiente proceso en el diagrama de flujo es nuevamente la condición de finalización, la diferencia entre la condición marcada en verde y la marcada en azul, radica en que el proceso marcado de verde es aplicado por el microcontrolador, si ésta condición se cumple, el microcontrolador detiene el brazo robótico dando fin a la aplicación, en caso contrario se genera el movimiento del robot, y después regresa nuevamente al proceso de adquisición de imágenes, que propicia la repetición del ciclo del algoritmo, el cual llega a su fin cuando el gesto de finalización es detectado.

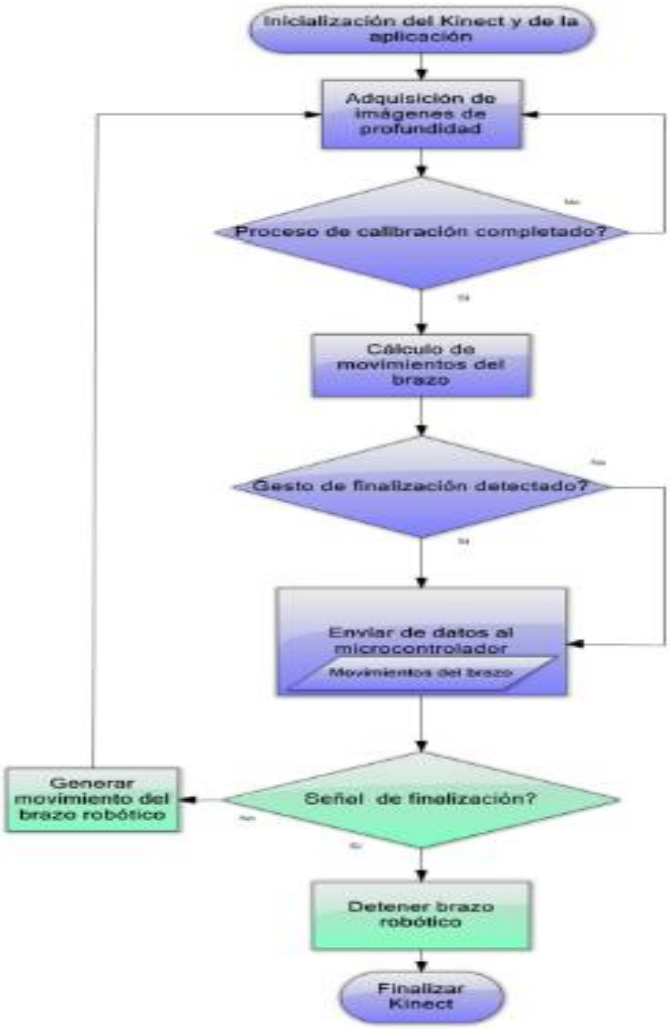

Figura 3.12 Diagrama de flujo del software del sistema.

El sistema en tiempo real implementado en este trabajo, omite algunas etapas de proceso dentro de un sistema en tiempo real, debido a que parte de los datos que son adquiridos del ambiente a través del sensor Kinect, son procesados por el sensor y enviados a la computadora como datos digitales. El diagrama de la figura 3.13 muestra las etapas que componen al sistema en tiempo real para el movimiento del brazo robótico y los procesos que son ejecutados dentro de cada etapa

Figura 3.13 Diagrama a bloques de las etapas del sistema.

Inicialización del Kinect: el sensor es la variable de entrada que interactúa con el ambiente y entrega datos del mismo (en nuestro caso usuario), además de que estos resultados dependen del tiempo, es por eso que se determinada como la primer etapa del sistema. A pesar de que solo se utiliza un sensor, se puede considerar la adquisición de múltiples variables, debido a los diferentes datos que entrega el Kinect.

- **Adquisición de imágenes y datos de profundidad:** se capturan todos los datos que envía el sensor hacia la computadora, permitiendo utilizarlos para la implementación de nuestro algoritmo de movimiento.
- **Proceso de calibración:** no todos los usuarios presentan el mismo tamaño de extremidades del cuerpo, motivo por el cual se realiza un proceso de calibración que permite la correcta lectura de los datos del usuario.
- **Selección de articulaciones a utilizar**: Microsoft Kinect entrega datos referentes a las articulaciones del cuerpo del usuario, sin embargo no todosestos datos se utilizan en el proyecto, dentro de esta etapa se adquieren únicamente las articulaciones que servirán para realizar el cálculo del movimiento del brazo derecho y mano izquierda del usuario, las demás son descartadas.

3.1.7 Inicialización del Kinect

Realizando investigación sobre las librerías mas utilizados para el desarrollo de aplicaciones con Kinect: el software de desarrollo de Microsoft para Kinect y OpenNI, se optó por elegir al segundo. A diferencia del software de Microsoft para Kinect, OpenNI fue desarrollado por la empresa que otorgó la patente a Microsoft para la construcción del dispositivo, por lo que estas librerías contienen herramientas que para este trabajo, representan una gran ventaja al lado del SDK de Microsoft. La licencia otorgada por Microsoft es exclusivamente para su uso no comercial, a diferencia del software utilizado en este proyecto, que sí permite el uso comercial de sus librerías, factor que puede contribuir en una futura implementación comercial.

En OpenNI los recursos que se utilizan del sensor para el desarrollo de aplicaciones se dividen por nodos, estos dependen de que recursos se vallan a utilizar del sensor, por ejemplo: si se desea utilizar únicamente la cámara de color RGB, se debe utilizar el nodo denominado “Generador de Imagen” que permite generar las imágenes a color por parte de la cámara del sensor y adquirir las imágenes cuando el sensor realice una captura y esta puedan ser enviada hacia la computadora. Los nodos se pueden utilizar de forma individual o de forma conjunta. Para poder hacer uso de los nodos en OpenNI, se debe hacer uso de un archivo de configuración con extensión .xml, este archivo contiene los nodos que serán utilizados en la aplicación.

3.1. 8 Software para adquisición de datos

Para la programación del software se implementó un modelo basado en el diagrama de clases mostrado en la figura 4.3, ocupando un objeto por cada clase, con el objetivo de organizar los algoritmos implementados en cada clase y poder tener un control y acceso para las características que ofrece el sensor.

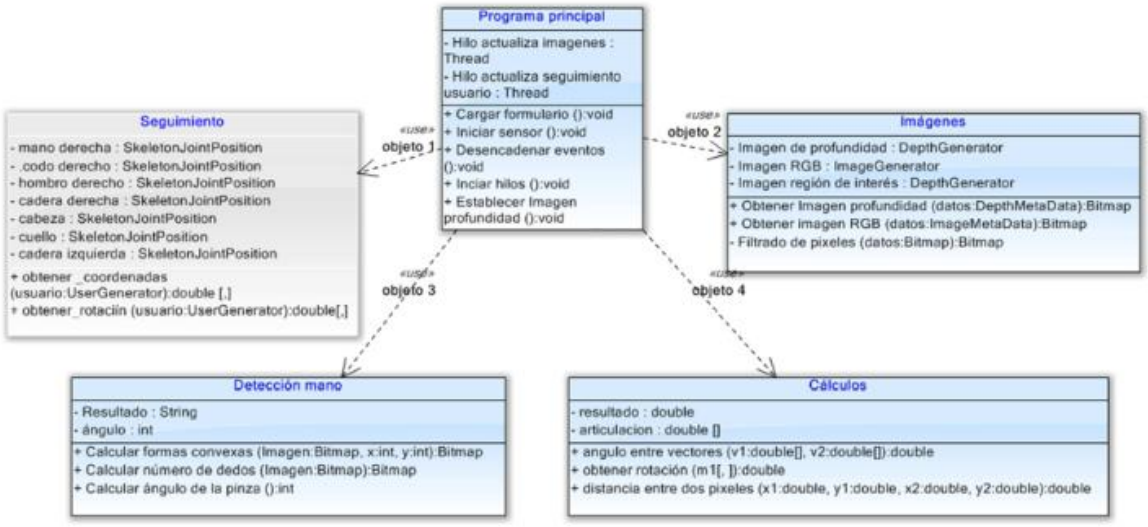

Figura 3.14 Diagrama de clases del software diseñado para la computadora.

El contenido de las clases se resume a continuación:

- Seguimiento: contiene los métodos para obtener todos los datos referentes al seguimiento del usuario como las coordenadas (x,y,z) de las articulaciones y su rotación.
- Imágenes: realiza la adquisición y procesamiento de todas las imágenes con la que se trabajan como imagen de profundidad e imagen RGB.
- Cálculos: implementa todos los algoritmos matemáticos utilizados en el software.
- Detección de mano: procesa dentro de sus métodos la imagen que obtiene la forma de la mano del usuario y con ello determina su estado (abierto o cerrado).
- Programa principal: inicia todos los procesos a través de dos subprocesos que se ejecutan de forma paralela para el funcionamiento del programa.

3.1.9 Imagen de profundidad

Como se mencionó anteriormente, los datos y seguimiento de usuario, dependen de las imágenes de profundidad, estas son generadas a través del nodo “Generador de profundidad” y entregadas a una velocidad de 30 tramas por segundo. Una desventaja de OpenNI es que no genera ningún tipo de evento cuando una imagen ha sido producida y enviada por el sensor hacia la computadora, simplemente los datos son entregados a través del método “Obtener Metadatos”, el cual entrega los datos de profundidad, debido a esto, la velocidad de adquisición de la trama o imagen, dependerá del algoritmo implementado en la aplicación y el tiempo que se asigne para procesar cada imagen.

La adquisición y procesamiento de datos del sensor Kinect son desarrollados utilizando tecnología .Net, y lenguaje de programación C#, trabajando las imágenes como mapas de bits. Estos mapas permiten crear una imagen del tamaño de la imagen entregada por el sensor, bloquear la imagen durante el proceso de copiado de datos y una vez que se termina este proceso, mostrar la imagen. En la figura 3.15 se muestra una representación del mapa de bits que permite almacenar la cantidad exacta de datos entregados por el nodo de profundidad.

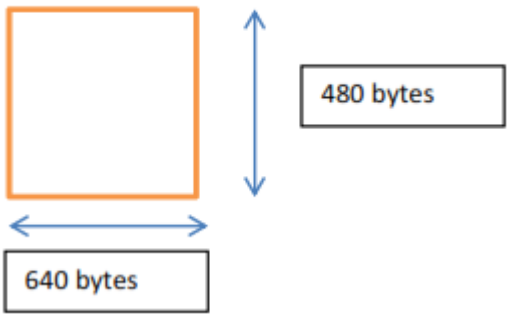

Figura 3.15 Mapa de bits a utilizar para copiar imágenes entregadas por el sensor

Cada byte dentro del mapa contiene 8 bits, esto indica que solo se pueden almacenar datos con valor máximo de 255. Realizando el producto entre las filas y columnas del mapa de bits, obtenemos un total de 307200 bytes para cada canalutilizado en las imágenes. La representación de los datos en una imagen de profundidad en OpenNI es establecida en escala de grises. Normalmente el valor de un pixel en escala de grises se representa por un canal, y cuyo valor del pixel se encuentra en el rango de 0-255, el nodo de profundidad entrega normalmente valores de 0 a 10000 para cada pixel, este valor representa la distancia en cm que existe entre el sensor y las coordenadas del espacio3D que corresponden a ese pixel, pero para poder representarlo en escala de grises se normaliza al rango de 0-255. Perceptivamente dentro de la imagen de profundidad, este valor representa la intensidad de un pixel, graficándose en pantalla como un determinado color entre el blanco y el gris, por ejemplo: si el valor del pixel es 0, el color del pixel es negro, y

de forma contraria, si el valor es 255 ,el color es blanco, si el valor se encuentra dentro de ese rango, obtendrá el color al cual su valor este mas cercano, ejemplo: 150 (obtenemos un color parecido al blanco), 30(obtenemos un color parecido al negro), el umbral establecido para definir la frontera de los valores de cada pixel es 128. Para poder obtener una imagen de profundidad y visualizarla dentro de una interfaz gráfica de usuario, se debe implementar un algoritmo que capture los datos de profundidad que entrega el sensor cada 30 milisegundos, y realice el proceso de copiado hacia el mapa de bits, para después mostrarlos de forma gráfica.

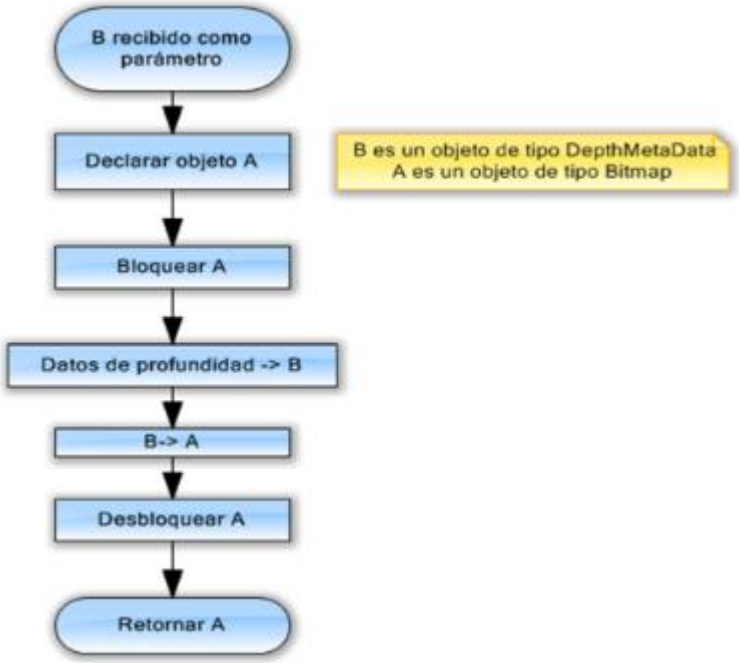

Figura 3.16 Algoritmo para obtener una imagen de profundidad.

El objeto de tipo “Bitmap” sirve para poder almacenar los datos de profundidad obtenidos de la función “GetMetaData”, el cual debe ser bloqueado durante el proceso de copiado, para evitar que algunos datos sean sobre escritos. El objeto de tipo “DepthMetaData” es utilizado para almacenar todos los datos de profundidad en el caso de que se quiera trabajar con ellos. La figura 3.17 despliega la imagen de profundidad que se obtiene cada 30 milisegundos a una distancia de 2 metros. Curiosamente se puede observar en la imagen de profundidad que los contornos del usuario o de los objetos se visualizan de color negro, simulando una sombra, esto se debe a que los puntos infrarrojos son desviados por los contornos de un objeto o persona, por lo que el valor de esas regiones se establece en 0. Los valores de estos pixeles que se encuentran en la frontera entre el usuario y la imagen, no afectan de forma directa dentro de la detección de usuario, debido a que el contorno puede ser obtenido utilizando otros algoritmos de visión por computadora

Figura 3.17 Imagen de profundidad y color del usuario a una distancia de 2metros utilizando el nodo "Generador de profundidad".

3.2.0 Imagen en el espacio RGB

La imagen RGB sirve de apoyo en este proyecto para que el usuario pueda referenciar y comparar el movimiento realizado por su brazo y el movimiento que ejecuta el brazo robótico, pero en el movimiento del robot, no afecta de ninguna manera. El nodo encargado de generar datos para adquisición de imágenes a color es "Generador de Imagen ". La composición de una imagen a color es diferente a una imagen de profundidad, un pixel de una imagen a color esta conformado por tres canales: R(rojo) ,G(verde)B(azul), los cuales contienen valores de 0 a 255. OpenNI permite alinear los datos de profundidad con los datos de la cámara RGB, permitiendo mostrar únicamente pixeles donde se muestre el usuario deseado.

El algoritmo para obtener imágenes a color es similar al de las imágenes de profundidad, la diferencia radica en los tipos que se manejan para obtener los datos del contexto y en la forma de copiarlos en una imagen, tomando en cuenta que en una imagen a color se obtienen tres canales y los valores de estos tres forman un color por cada pixel, el algoritmo para imágenes a color se muestra en la figura 3.18

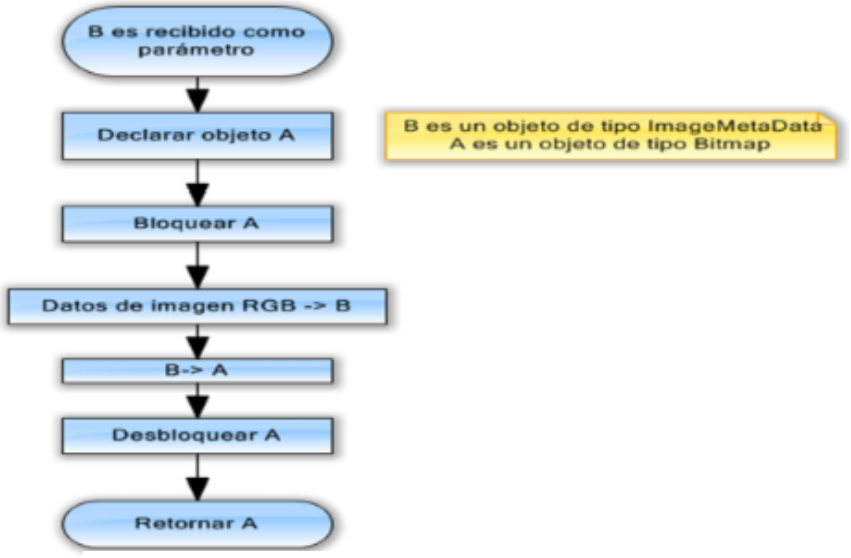

Figura 3.18 Algoritmo para obtener imagen a color.

En la figura 3.19 se muestra la imagen a color que se obtiene a través del nodo Generador de Imagen. La resolución de la imagen es normalmente de 640*480 pixeles, pero el sensor Kinect puede generar imágenes a color con una resolución de 1280*1024 reduciendo la velocidad de envío a 15 tramas por segundo.

Figura 3.19 Imagen del patrón infrarrojo emitido por el sensor de profundidad, obtenida a través del nodo "Generador Infrarrojo".

3.2.1 Proceso de calibración

El sensor de profundidad detecta máximo a seis usuarios dentro del área que cubre, sin embargo, para realizar un mejor seguimiento de usuario, es recomendable trabajar con un máximo de dos usuarios, ya que el aumento de seguimiento de estos, provoca retardos en el procesamiento de datos de usuario por parte de la computadora. En el presente trabajo se propone a un solo usuario como operador del brazo robótico, pero también se consideran factores como la intromisión accidental o provocada de otra persona dentro del área del sensor.

OpenNI tiene la ventaja de generar eventos cuando se detecta o pierde un usuario, cuando se detecta un nuevo usuario asigna un número de identificación (ID) al usuario detectado, por ejemplo: si la aplicación es iniciada y entra un primer usuario al área que cubre el sensor, se le es asignado el número 1 como ID, si posteriormente ingresa otra persona se le asigna el ID número 2, continuando de esta manera sucesiva hasta llegar al máximo de usuarios que pueden ser detectados, si llegaran a ingresar más usuarios dentro del área del sensor, estos no serian detectados o simplemente obtendríamos un error en tiempo de ejecución. Una pregunta común que puede llegar a surgir dentro de la detección de usuario es: ¿Qué pasa cuando un usuario es identificado, pero después sale del área del sensor y regresa nuevamente? , el sensor Microsof Kinect guarda los datos de un usuario cuando es perdido, es decir cuando abandona el rango que cubre el sensor, si un nuevo usuario es detectado posteriormente, el sensor compara los datos del usuario detectado, con los datos del usuario perdido, buscando semejanzas en cuando a la forma del cuerpo y tamaño, por lo tanto es posible que al usuario detectado se le asigne el ID del usuario perdido, si es que se tratara de la misma persona, en caso contrario se le asigna un nuevo ID , esto propicia algunos errores que pueden llegar

a suceder durante la identificación, por ejemplo: que un usuario salga del área del sensor y otro entre, tomando el ID del primero debido a una posible similitud en las extremidades y forma del cuerpo entre ambos , lo cual es difícil que pueda suceder.

Otra ventaja de poder identificar mediante ID a los diferentes usuarios dentro del área del Kinect es que podemos utilizar los pixeles de profundidad que correspondan únicamente a ese determinado usuario, permitiendo realizar interfaces gráficas de usuario en donde solo se despliegue la imagen que corresponde al cuerpo del usuario, o en su defecto alinear los pixeles de usuario con los de la cámara RGB para obtener imágenes a color del cuerpo del usuario.

La diferencia para adquirir datos de usuario entre el SDK de Microsoft y las librerías de OpenNI para el seguimiento de usuario, radica en que OpenNI necesita una pose de calibración. Las librerías de Microsoft no requieren de una pose de calibración para acceder a las articulaciones del usuario, esto se debe a que la calibración que realiza el software de Microsoft se hace mediante un algoritmo que localiza el centro de masa del usuario detectado, y a partir de ahí, se localizan los puntos a los cuales se tiene acceso. OpenNI incluye una pose de calibración, la cual no impide que se puedan acceder a las articulaciones del usuario, pero se obtiene una mejor precisión del seguimiento realizando esta pose de calibración. El proceso de calibración también ayuda en el seguimiento de usuarios que estén dentro del área del sensor y tengan parecidas pero no idénticas formas en los sus cuerpos. OpenNI contiene eventos que se generan cuando ocurren procesos relacionados con la detección y calibración de usuario. En la figura 3.20 se muestra el diagrama de flujo para poder realizar la correcta calibración del usuario, estos procesos son establecidos por OpenNI de forma general, sin poder ser modificados, implementando la calibración

Figura 3.20 Algoritmo para calibración de usuario

Los procesos marcados de color azul dentro del algoritmo de calibración mostrado en la figura 4.11 representan los eventos que se generan cada que ocurre dicho proceso, los procesos de color verde representan métodos que deben ser llamados para realizar la tarea específica. El proceso de calibración debe ser realizado para cada usuario detectado, si es el caso que se requiera trabajar con los datos de un nuevo usuario. El proceso de calibración de un usuario se puede describir de la siguiente manera: un nuevo o primer usuario es detectado por el sensor generando el evento "Nuevo usuario" , a partir de este momento se realiza la petición para detectar una pose de calibración, es decir, se asume que el usuario detectado realizará la pose de calibración, el método que realiza esta petición es "Iniciar pose de detección", este método recibe dos parámetros: el ID(número de usuario) del que se desea calibrar, y la pose de calibración a realizar, la cual esta definida de forma predeterminada en OpenNI y es la que se muestra en la figura 3.21 .

Figura 3.21 Pose "Psi" para llevar a cabo el proceso de calibración [14]

La razón de la forma de la pose para calibrar al usuario se debe que a través de la pose se puede tomar una imagen donde se visualicen casi la mayoría de las articulaciones del usuario, o por lo menos la parte superior del cuerpo del usuario. Una vez que se realizó la petición de detección de pose de calibración y el usuario realizó la pose, se genera el evento "Pose detectada", el cual indica que la pose ha sido detectada, dentro de este evento se debe llamar al siguiente método que indica el diagrama de flujo , "Petición de calibración", el cual se encargará de realizar la petición para la calibración del usuario, este método recibe como argumento el ID del usuario que se requiera calibrar, antes de llamar a este método de petición de calibración se debe de llamar al método "Detener detección de pose", ya que si no se hace, el proceso entra en un ciclo infinito porque elevento "Pose detectada" se genera constantemente. El siguiente evento que se genera es "Calibración completa", evento que se genera cuando se completa la calibración, en este proceso

se deben tomar en cuenta algunas consideraciones, el termino de la calibración no significa la correcta calibración de usuario, por ese motivo OpenNI cuenta con una bandera dentro de este evento que indica si la calibración se completo correctamente, a partir del valor de esa bandera se determina el comienzo del seguimiento de usuario y acceso a las coordenadas 3D del mismo, en el caso de que el valor de la bandera sea verdadero se asume la correcta calibración del usuario. El método encargado de comenzar el seguimiento de usuario es “Iniciar seguimiento”, método que de igual forma recibe el numero ID de usuario a seguir, si el valor de la bandera que indica la correcta calibración de usuario es establecido es erróneo, se debe repetir el proceso de calibración a partir de la petición de la detección de pose de calibración, llevándose a cabo este proceso de forma iterativa hasta obtener una correcta calibración del usuario. Los factores que influyen para que la calibración de usuario no se lleve a cabo de forma correcta se deben a una mala imitación de la pose de calibración o a que el usuario deje de realizar la pose de calibración antes de completarse el proceso, normalmente la calibración de usuario es completada de forma correcta en menos de dos segundos. La imagen 4.13 muestra la localización de cada uno de las articulaciones después del proceso de calibración, se puede observar que las coordenadas de localización de los articulaciones se encuentran alineadas con los pixeles de la imagen RGB, entonces surge una nueva pregunta: ¿como alinear las coordenadas de un pixel de la imagen RGB con las coordenadas de profundidad?, el sensor de profundidad entrega coordenadas dentro del plano tridimensional del mundo real(es decir del plano 3D en las superficie que se proyectan los puntos infrarrojo), estas coordenadas se alinean mediante el proceso de calibración.

3.2.2 Requisitos previos para la instalación de la Kinect de móldelo 1414

Kinect para Windows requiere lo siguiente:

- Uno de los siguientes sistemas operativos:

- Windows 7
- Windows Embedded Standard 7

- Requisitos de hardware

- Procesador de 32 bits (x86) o 64 bits (x64)
- Procesador de doble núcleo, de 2.66-GHz o más rápido
- Bus USB 2.0 dedicado
- 2 GB de RAM
- Sensor Kinect para Windows

- Requisitos de software
- Microsoft Visual Studio 2010

- NET Framework 4.0
- Microsoft Speech Platform SDK v11
- Microsoft DirectX Software Development Kit
- Microsoft Speech Platform Runtime, versión 10.2 (x86)
- Microsoft Speech Platform - Server Runtime Languages
- Microsoft Speech Platform - Software Development Kit

  - conocimientos en software

    - Arduino
    - Visual studio

  - Conocimientos en progamacion.
    - lenguaje C++
    - visual studio C#
    - c Sharp

3.2.3 EL SENSOR KINECT Y SU COMPOSICIÓN

El sensor de Kinect es un equipo alargado conectado a un pivote, diseñado para estar en una posición horizontal. El dispositivo tiene una cámara RGB, un sensor de profundidad y un micrófono multi-array bidireccional que, en conjunto, capturan imágenes y movimientos de los cuerpos en 3D, además de ofrecer reconocimiento facial y aceptar comandos de voz Kinect figura 3.22.

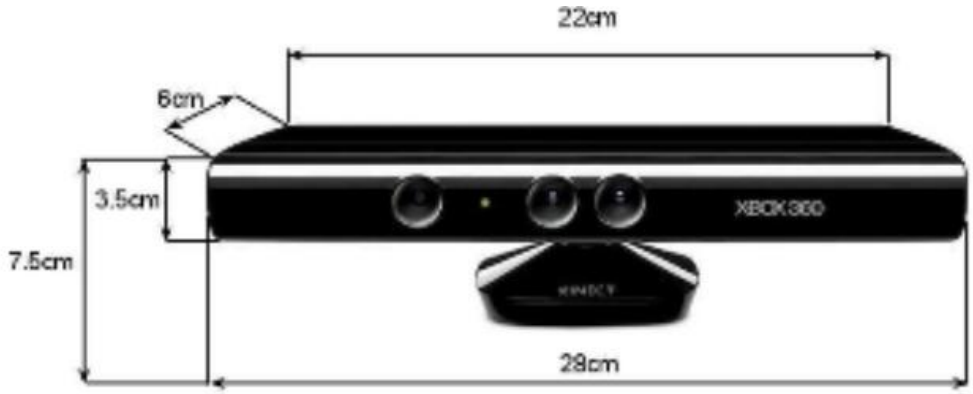

Figura 3.22 dimisiones del Kinect

El sensor de Kinect adquiere imágenes de video con un sensor CMOS de colores a una frecuencia de 30 Hz, en colores RGB de 32 bits y resolución VGA de 640×480 pixeles. El canal de video monocromo CMOS es de 16 bits, resolución QVGA de 320×240 pixeles con hasta 65,536 niveles de sensibilidad.

Para calcular distancias entre un cuerpo y el sensor, el sensor emite un haz láser infrarrojo que proyecta un patrón de puntos sobre los cuerpos cuya distancia se determina. Una cámara infrarroja capta este patrón y por hardware calcula la profundidad de cada punto. El rango de profundidad del sensor de Kinect está entre 0.4 y 4 mts. Existen 2 modos (Default y Near) para determinar distancias. Se ha elegido el modo “Default” ya que permite medir hasta 4 metros de distancia con respecto al sensor.

El ángulo de vista (FOV) es de 58° horizontales y 45° verticales. Por otro lado el pivote permite orientar en elevación, hacia arriba o hacia abajo incrementando el FOV hasta en 27°. El array del micrófono tiene cuatro cápsulas, y opera con cada canal procesando en 16 bits de audio con un ratio de frecuencia de 16 kHz.

3.2.4 DETECCIÓN DE ARTICULACIONES MEDIANTE EL SENSOR KINECT

Normalmente, el flujo de datos “crudos” que suministra el sensor Kinect proporciona la información de 20 puntos de una persona cuando está dentro del rango visual admisible. De esta manera, es posible obtener la información posicional en tiempo real y en los 3 ejes cartesianos de estos 20 puntos por persona. Cabe destacar que la información recolectada está en un formato imagen de 640x480 pixeles tomada a una velocidad de 30 FPS (Frames per Second o Cuadros por segundo).

Con el fin de obtener todas las trayectorias que realiza una persona, es necesario registrar y archivar los 20 puntos que entrega el dispositivo. Cabe destacar que esta es la funcionalidad más importante que tiene el sensor para la aplicación buscada, dado que otorga la posibilidad de tener la posición tridimensional en cada instante de tiempo. Si se toma una sucesión dinámica de estas entregas, se podrá obtener todas las trayectorias de todos las partes del cuerpo de una persona.

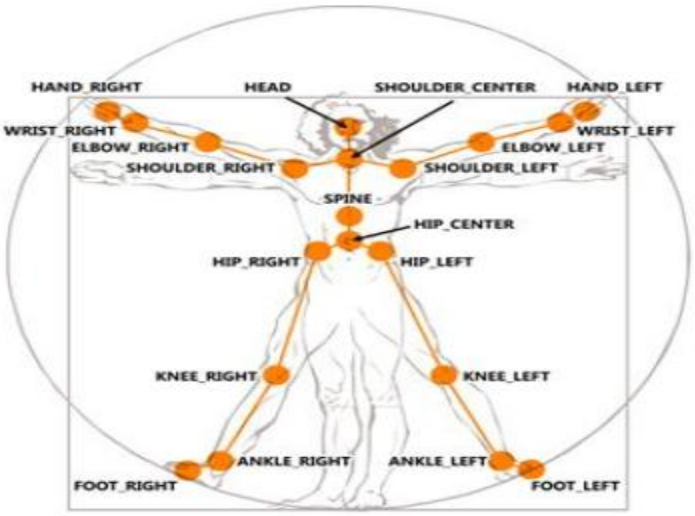

Figura 3.23 articulaciones del cuerpo humano.

3.2.5RECONOCIMIENTO DE GESTOS

Cuando nos referimos a reconocimiento de gestos, en este caso, se trata de asignar a ciertos movimientos consecutivos de partes del cuerpo una determinada acción (saltar,saludar, girar, etc.).

Al igual que nos pasa con la detección de posturas el reconocimiento de gestos también tiene muchas técnicas diferentes que se pueden aplicar para lograr una mejor identificación o una implementación más sencilla. Una técnica muy utilizada para estos casos en utilizar redes neuronales las cuales se pueden entrenar para ir alcanzando cada vez más precisión y calidad de detección.

También podemos usar técnicas como definir algorítmicamente el gesto, al igual que hicimos con la postura, o comparar con una serie de plantillas ya definidas. La técnica de las plantillas es muy eficiente para gestos que siempre se realizan de la misma forma y que con cualquier otra técnica sería muy difícil de detectar.

Para ilustrar figura 3.24 la técnica de comparación de plantillas podemos poner el golpeo de una bola jugando al tenis. Este movimiento es bastante complejo como para definirlo algorítmicamente y es más fácil si tenemos una serie de plantillas (gestos) e ir comparando cada captura con ellas. El movimiento capturado no será igual a la plantilla que tengamos pero se puede permitir un margen de error y si el resto del movimiento se sigue pareciendo al resto de plantillas podemos darlo por válido.

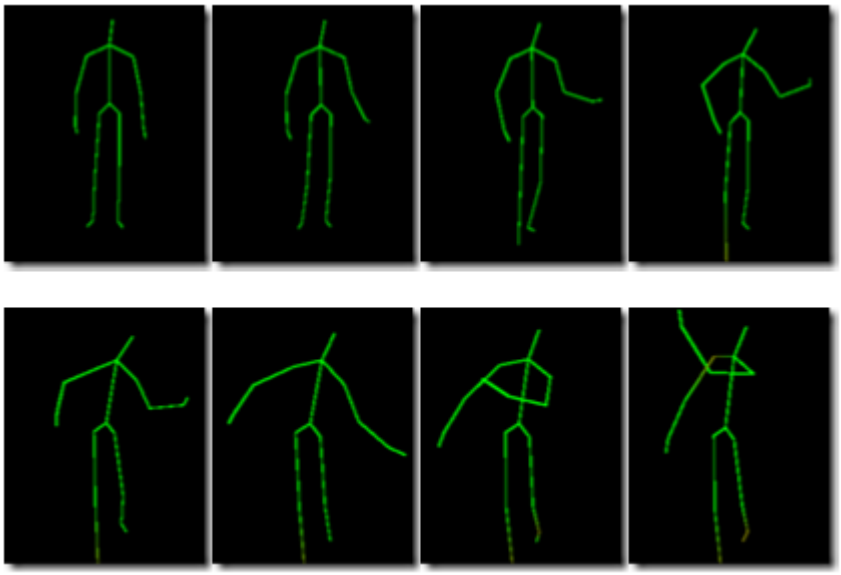

Figura 3.24 la técnica de comparación de plantillas

3.2.6 PLATAFORMAS DE DESARROLLO

En este proyecto se trabaja con Visual Studio 2017, ayudado por las librerías de SDK Kinect y, posteriormente, utilizar la plataforma ARDUINO para enviar los comandos al movimiento y que este realicé la acción debida.

3.2.7 MICROSOFT VISUAL STUDIO

Visual Studio es una plataforma de desarrollo para sistemas operativos Windows. Soporta varios lenguajes de programación tales como C++, C#, J# y Visual Basic .NET, al igual que entornos de desarrollo web como ASP.NET. Aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

El entorno de desarrollo integrado (IDE) de figura 3.25 Visual Studio presenta un conjunto de herramientas destinadas a ayudarle a escribir y modificar el código para los programas, así como a detectar y corregir errores en los programas.

Elegimos esta plataforma por su fácil manejo y, sobre todo, debido a que el sensor Kinect cuenta con una plataforma propia que puede ser fácilmente unida y utilizada con Visual Studio.

Para este proyecto se utilizó la versión 2017, puesto que es la más compatible y estable en el momento. Es por eso que los códigos del programa se realizaron en esta versión.

Figura 3.25 plataforma de desarrollo para sistemas operativos Windows.

3.2.8 LENGUAJE DE PROGRAMACIÓN C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre C Sharp fue inspirado por la notación musical, donde '#' (sostenido, en inglés sharp) indica que la nota (C es la nota do en inglés) es un semitono más alta, sugiriendo que C# es superior a C/C++. Además, el signo '#' se compone de cuatro signos '+' pegados.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

3.2.9 CARACTERÍSTICAS DEL LENGUAJE C#

El estándar ECMA-334 lista las siguientes características en el diseño para C#:

- Lenguaje de programación orientado a objetos simple, moderno y de propósito general.
- Inclusión de principios de ingeniería de software tales como revisión estricta de los tipos de datos, revisión de límites de vectores, detección de intentos de usar variables no inicializadas, y recolección de basura automática.
- Capacidad para desarrollar componentes de software que se puedan usar en ambientes distribuidos.
- Portabilidad del código fuente.
- Fácil migración del programador al nuevo lenguaje, especialmente para programadores familiarizados con C, C++ y Java.
- Soporte para internacionalización.
- Adecuación para escribir aplicaciones de cualquier tamaño: desde las más grandes y sofisticadas como sistemas operativos hasta las más pequeñas funciones.
- Aplicaciones económicas en cuanto a memoria y procesado.

3.3.0 MICROSOFT SDK KINECT

Un kit de desarrollo de software o SDK (Software Development Kit) es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etc.

Es algo tan sencillo como una interfaz de programación de aplicaciones o API (Application Programing Interface) creada para permitir el uso de cierto lenguaje de programación, o puede, también, incluir hardware sofisticado para comunicarse con un determinado sistema embebido. Las herramientas más comunes incluyen soporte para la detección de errores de programación como un entorno de desarrollo integrado o IDE (Integrated Development Environment) y otras utilidades. Los SDK frecuentemente incluyen, también, códigos de ejemplo y notas técnicas de soporte u otra documentación de soporte para ayudar a clarificar ciertos puntos del material de referencia primario.

El SDK no sólo incluye controladores sino también interfaces de programación de aplicaciones (API's), interfaces de dispositivo, documentos de instalación y materiales con recursos. Esto representa otro emocionante hito para la tecnología que ha capturado la imaginación de millones y se ha convertido en el dispositivo electrónico de ventas más rápidas de todos los tiempos.

Para este proyecto utilizamos la versión más actualizada del SDK de Kinect, la versión

3.3.1 ¿POR QUÉ KINECT?

La cámara del sensor Kinect cuenta ya con la electrónica necesaria para que el sensor funcione a nuestras necesidades; Solo necesitamos programar, en Visual Studio, el código necesario para que detecte ciertos puntos del esqueleto humano y que los comandos sean enviados al móvil.

SDK de Kinect, otorga la facilidad de contar con las librerías que incluyen dichas instrucciones, como es la de detectar puntos en el cuerpo y reconocer gestos realizados. Esto hace que la programación sea menos complicada y más eficaz.

3.3.2 ARDUINO

Arduino es una figura 3.26 plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

El hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 por su sencillez y bajo costo que permiten el desarrollo de múltiples diseños. Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque (boot loader) que corre en la placa.

Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.

figura 3.26 plataforma de hardware libre

3.3.3 BIBLIOTECAS EN ARDUINO

Para hacer uso de una biblioteca en Sketch (el IDE de Arduino), basta con hacer clic sobre “Import Library” en el menú, escoger una biblioteca y se añadirá el #include correspondiente. Las bibliotecas estándar que ofrece Arduino son las siguientes:

Serial: Lectura y escritura por el puerto serie.

*EEPROM: Lectura y escritura en el almacenamiento permanente.

- read(), write()

*Ethernet: Conexión a Internet mediante “Arduino Ethernet Shield“. Puede funcionar como servidor que acepta peticiones remotas o como cliente. Se permiten hasta cuatro conexiones simultáneas.

- Servidor: Server(), begin(), available(), write(), print(), println()
- Cliente: Client(), connected(), connect(), write(), print(), println(), available(), read(), flush(), stop()

Firmata: Comunicación con aplicaciones de ordenador utilizando el protocolo estándar del puerto serie.

*LiquidCrystal: Control de LCDs con chipset Hitachi HD44780 o compatibles. La biblioteca soporta los modos de 4 y 8 bits.

*Servo: Control de servo motores. A partir de la versión 0017 de Arduino la biblioteca soporta hasta 12 motores en la mayoría de placas Arduino y 48 en la Arduino Mega.

- attach(), write(), writeMicroseconds(), read(), attached(), detach()

El manejo de la biblioteca es bastante sencillo. Mediante attach(número de pin) añadimos un servo y mediante write podemos indicar los grados que queremos que tenga el motor (habitualmente de 0 a 180).

*SoftwareSerial: Comunicación serie en pines digitales. Por defecto Arduino incluye comunicación sólo en los pines 0 y 1 pero gracias a esta biblioteca podemos realizar esta comunicación con el resto de pines.

*Stepper: Control de motores paso a paso unipolares o bipolares.

- Stepper(steps, pin1, pin2), Stepper(steps, pin1, pin2, pin3, pin4), setSpeed(rpm), step(steps)

El manejo es sencillo. Basta con iniciar el motor mediante Stepper indicando los pasos que tiene y los pines a los que está asociado. Se indica la velocidad a la que queramos que gire en revoluciones por minuto con setSpeed(rpm) y se indican los pasos que queremos que avance con step(pasos).

*Wire: Envío y recepción de datos sobre una red de dispositivos o sensores mediante

Two Wire Interface (TWI/I2C).

Además las bibliotecas Matrix y Sprite de Wiring son totalmente compatibles con Arduino y sirven para manejo de matrices de leds.

También se ofrece información sobre diversas bibliotecas desarrolladas por contribuidores diversos que permiten realizar muchas tareas.

3.3.4 MICROCONTROLADORES PARA ARDUINO

Los microcontroladores Arduino Diecimila, Arduino Duemilanove y una figura 3.27 Arduino Mega están basados en Atmega168, Atmega 328 y Atmega128

Microcontrolador	ATmega328
Voltaje de funcionamiento	5V
Alimentación	7-12V
Voltaje máximo de entrada	20V
Pines digitales I/O	14 (6 con salida PWM)
Pines de entrada analógica	6
Corriente DC por I/O Pin	40 mA
Corriente DC para el pin 3.3V	50 mA
Memoria Flash	32 KB
SRAM	2 KB
EEPROM	1 KB
Velocidad de reloj	16 MHz

figura 3.27 Arduino tabla de datos

CAPÍTULO 4. DESARROLLO DEL PROYECTO

Para asegurarse que funciona, dirijase a “C:\Program Files\Microsoft Research KinectSDK” o la ruta que se haya elegido para la instalación del SDK y ejecute el programa Skeletal Viewer o Shape Game que son 2 programas de ejemplo que se instalan con el SDK.

Ahora se procede a crear un proyecto para iniciar la programación con el SDK de Kinect para Windows y que servirá como proyecto base para las aplicaciones.

Una vez abierto Visual Studio 2017 se crea un proyecto WPF (Windows Presentation Foundation). Se le da un nombre y aceptar.

4.0.1 Creación del código de la cámara de la Kinect

Abrimos visual studio damos nuevo proyecto y seleccionamos aplicación WPF

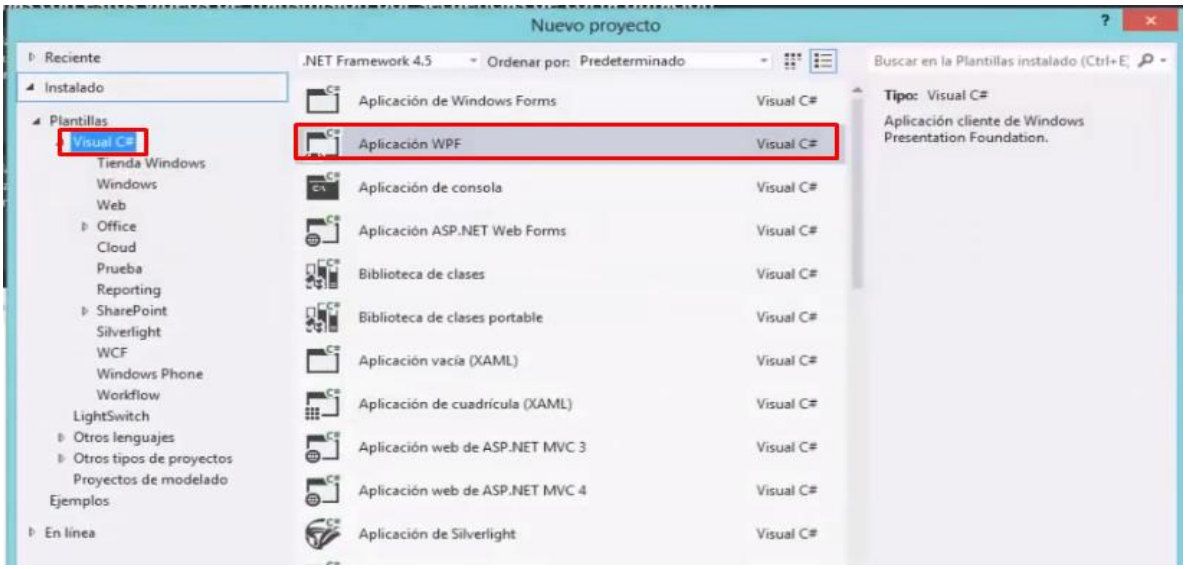

Figura 4.0 selección de lenguaje de programación

Le pondremos de nombre CamaraWeb damos aceptar por que la Kinect tomara los datos y nosotros las miraremos en la aplicación del programa

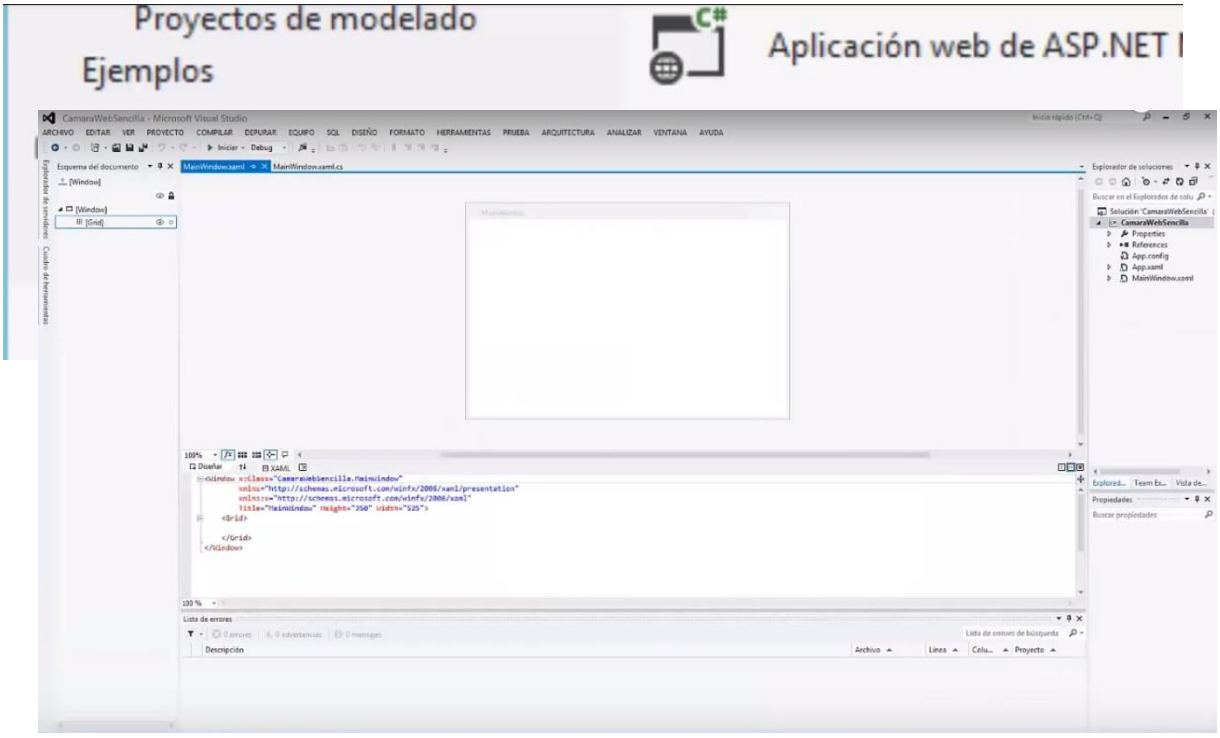

Figura 4.2 ventana completa de visual studios y sus herramientas

Luego podemos visualizar la ventana completa de visual studios y sus herramientas.

Como podemos ver en la ventana de visual hacemos referencia a esta parte que es la más importante del formato WPF en done colocaremos nuestra seudocódigos Y diseños de nuestra aplicación a través de código XML

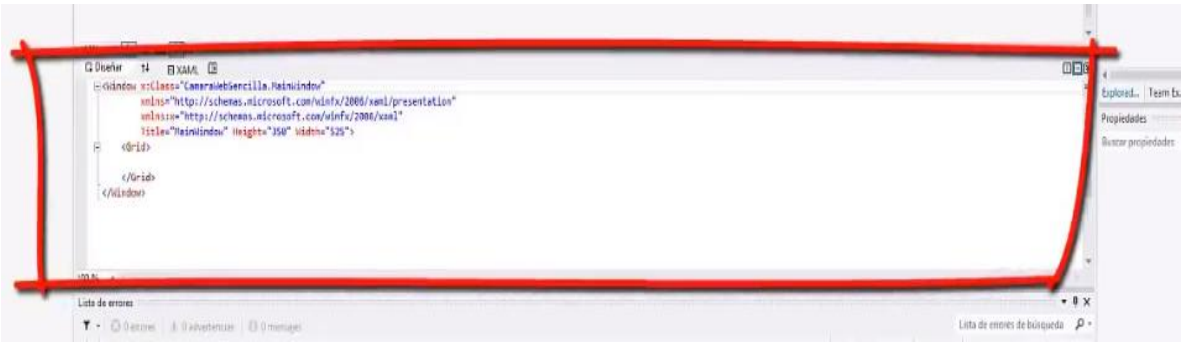

Figura 4.3 ventana de código XML

Ya que tenemos creado nuestro proyecto lo que aremos es conectar la Kinect a la pc y para poder dar de alta con visual agregaremos las librerías de SDK de Kinect para agregar estas librerías primero debemos crear una referencia de ellos vamos a exploradores de soluciones

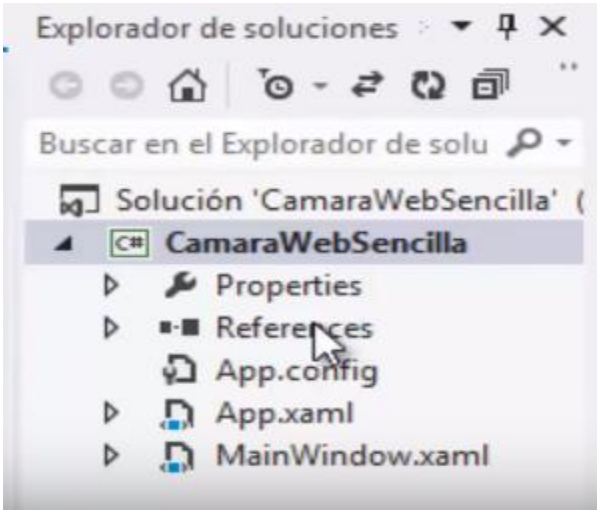

Figura 4.4 soluciones de los archivos de la programación

Luego damos clic con el botón derecho sobre references y damos agregar referencia…

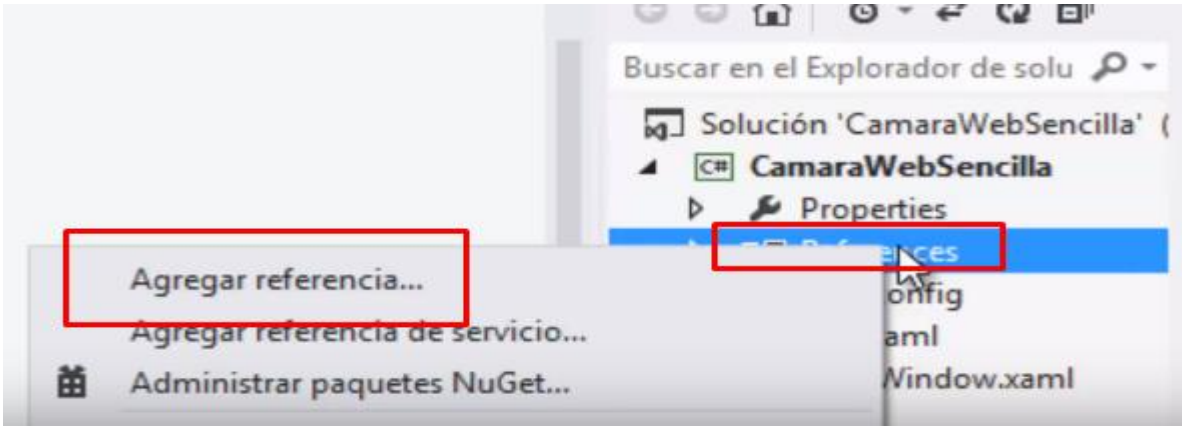

Figura 4.5 agregar las referencias o librería

Senos abrir una ventana que dirá administrador de referencias respeto el nombre del proyecto en esa ventana nosotros daremos de alta la librería que ocuparemos

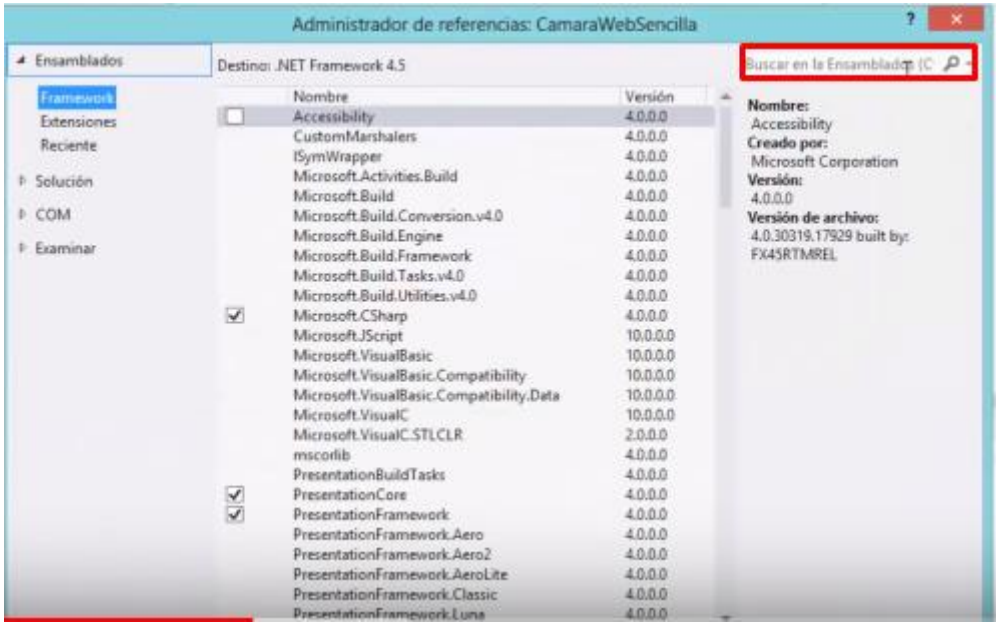

Figura 4.6 buscador de librerías

Nos iremos a la parte del buscador para escribiéremos el nombre de la librería que ocuparemos por ejemplo será la de Kinect luego lo marcaremos dándole clic en el cuadro blanco hasta que se sombre con una palomita y le damos aceptar

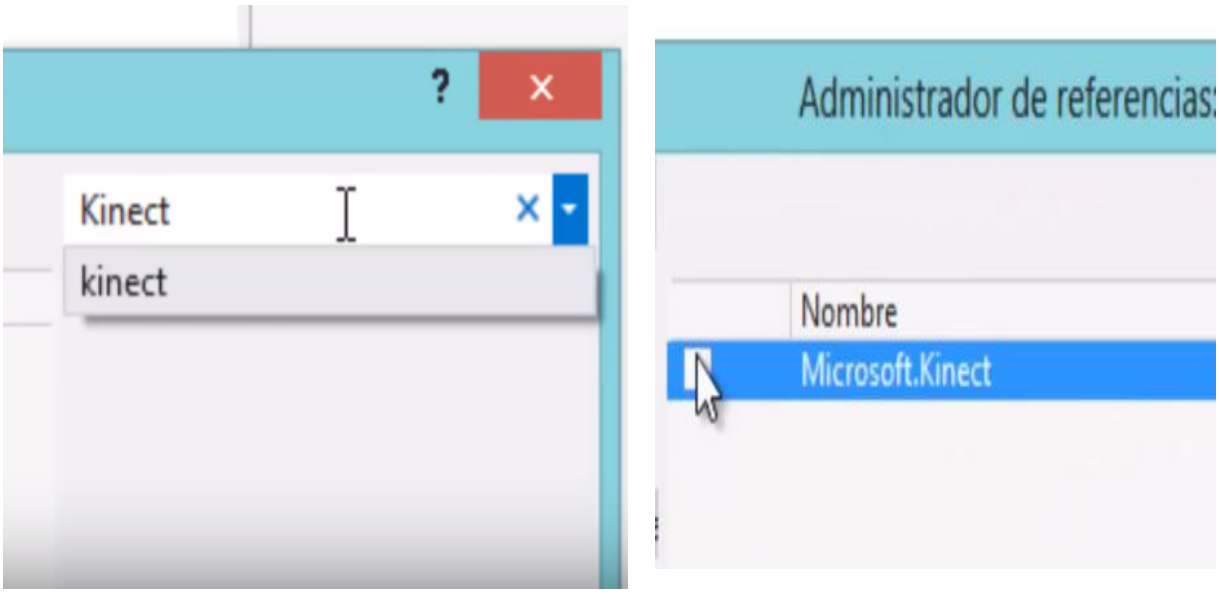

Figura 4.7 librería de Microsoft Kinect

Lo primero que aremos en el proyecto será el diseño lo que agregaremos será un control imagen en el cual mostraremos las imágenes que recibiremos de la Kinect

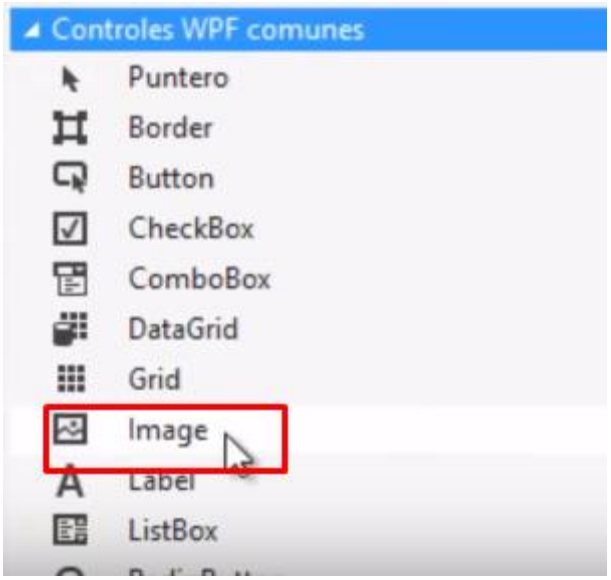

Figura 4.8 cuadro de herramientas

Arrastramos la imagen y expandimos para que severa mejor Figura 4.9

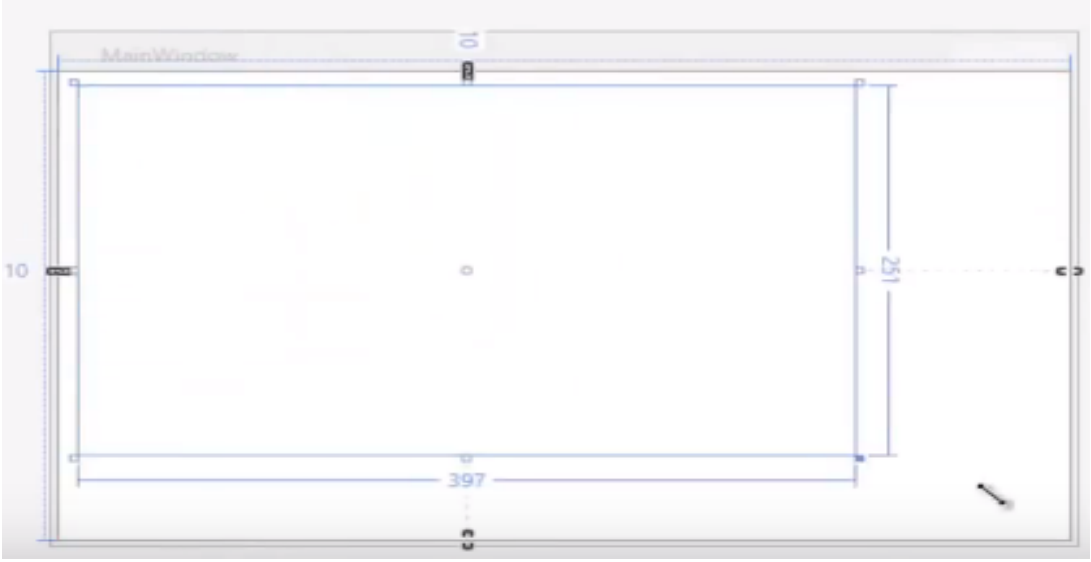

Figura 4.9 cuadro donde se colocara la imagen

Vamos lado derecho en propiedades le pondremos un nombre para poder identificarlo mostrarVideo

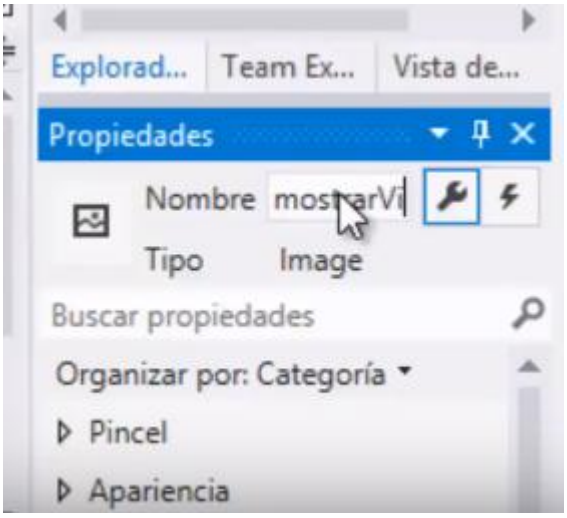

Figura 4.10 agregar nombre del Ford

Ahora vamos con la parte funcional para ejecutar el código al momento de que inicie nuestra aplicación vamos a llamar al evento loug de nuestra ventana podemos seleccionar la ventana o formulario de termo diseño o desde el modo código

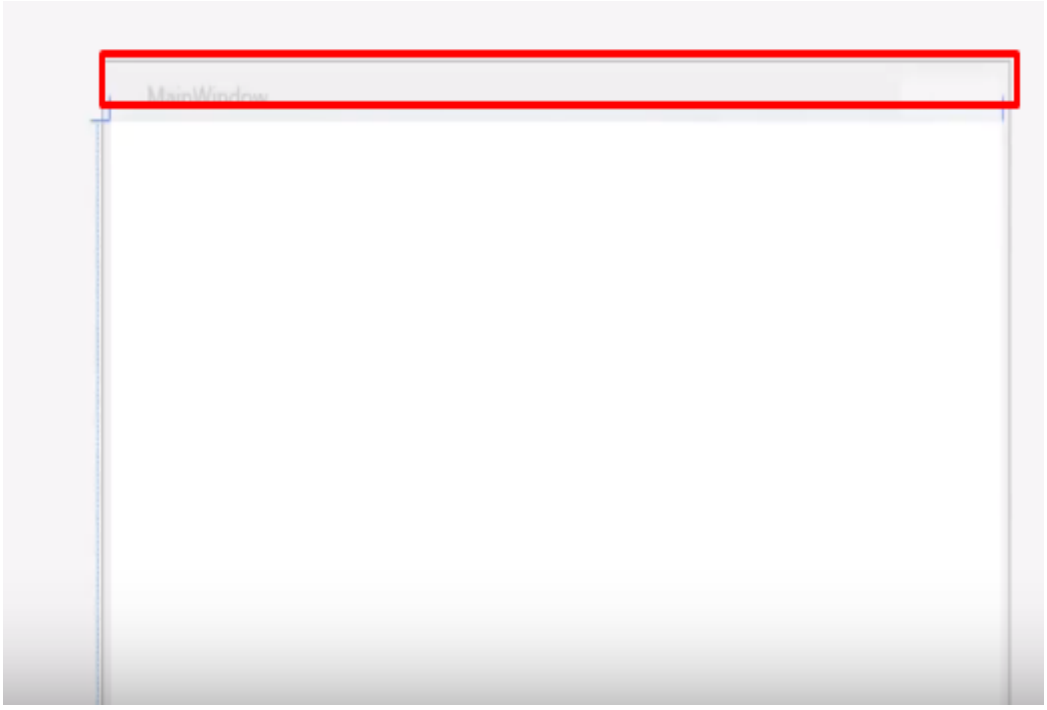

Figura 4.11 dar clic en loug para marcar la ventana

Vamos a propiedades y damos en el símbolo del rayo ya que este sombreado en color azul el bode redes

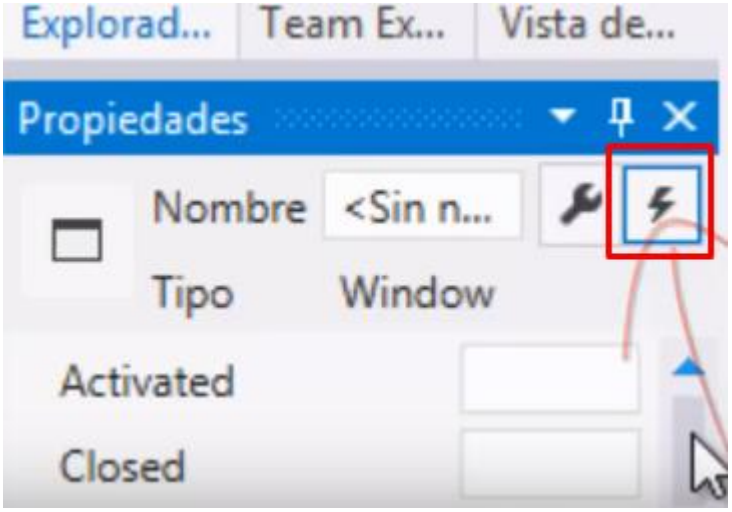

Figura 4.12 Explorador de la herramientas

Una vez que tengamos marcado el símbolo del rayo nos saldrá unas opciones de evento en esta lista buscamos el Figura 5.15 evento loaded presionamos doble clic sobre la caja de texto

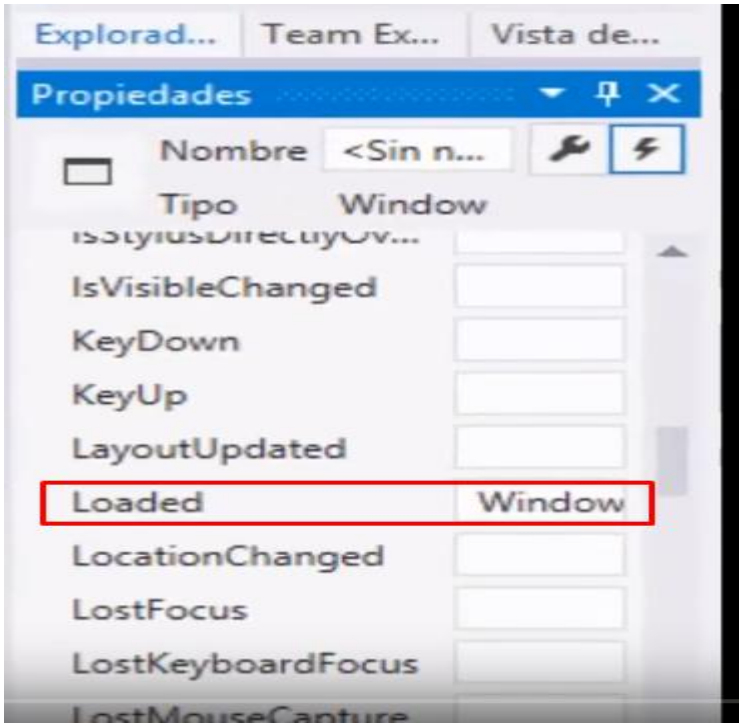

Figura 4.13 evento loaded

Luego nos madara a una ventana donde automáticamente nos mandara a nuestra ventana de código ya estamos en el código fuente de nuestra aplicación Como podrás recordar ya aviamos hecho una referencia sobre la librería de la Kinect en toses aquí en el código fuente lo síguete que debemos hacer el “using” que contiene todas las propiedad para poder controlar la Kinect escribiremos Using Microsoft. Kinect;

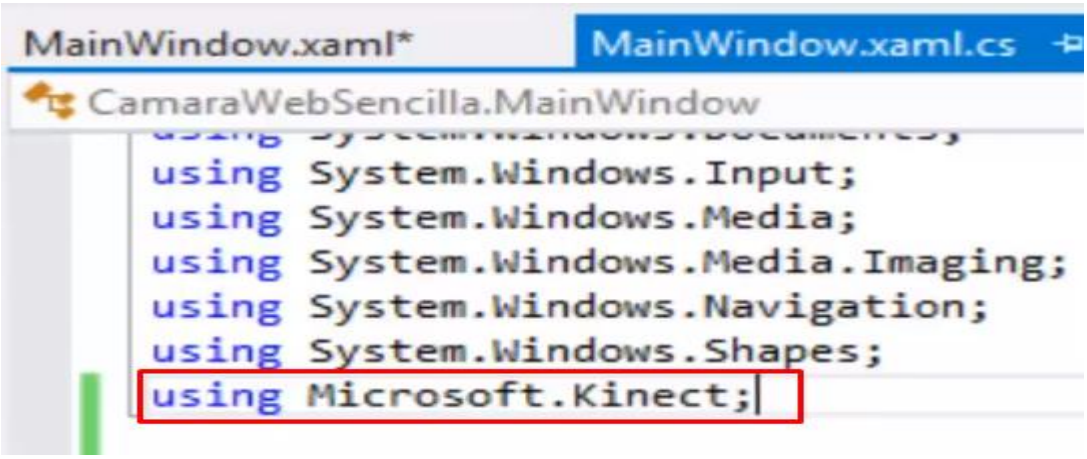

Figura 4.14 declaraciones de librerías

Para poder interactuar con la Kinect o lo primero que tenemos que hacer es obtener una estancia de la clase Kinect sensor esta clase es parte de SDK te conectara con partes de la función del Kinect escribimos dentro de nuestra “ class MaiWindow ” lo siguiente “KinectSensor mikinect;”

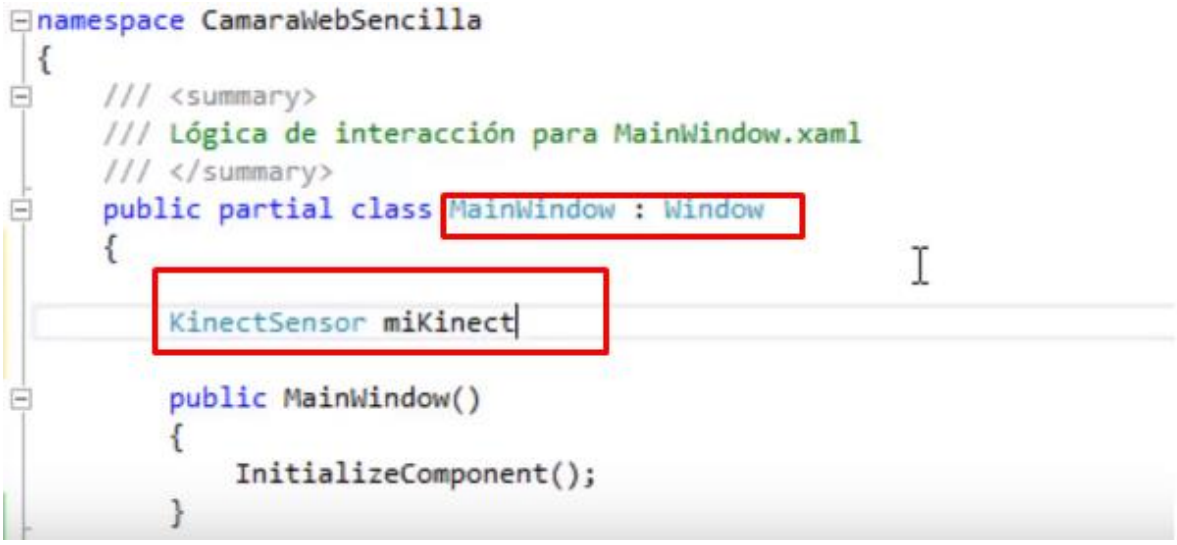

FIGURA 4.15 declaraciones de maiwindow kinect

Si vemos más abajo podemos ver el evento loaded de la ventana inicia que se creó hace rato este evento se ejecutara al momento que iniciemos el proyecto Dentro del método escribiremos esta línea de código

"miKiinect = KinectSensor.KinectSensors[0]"

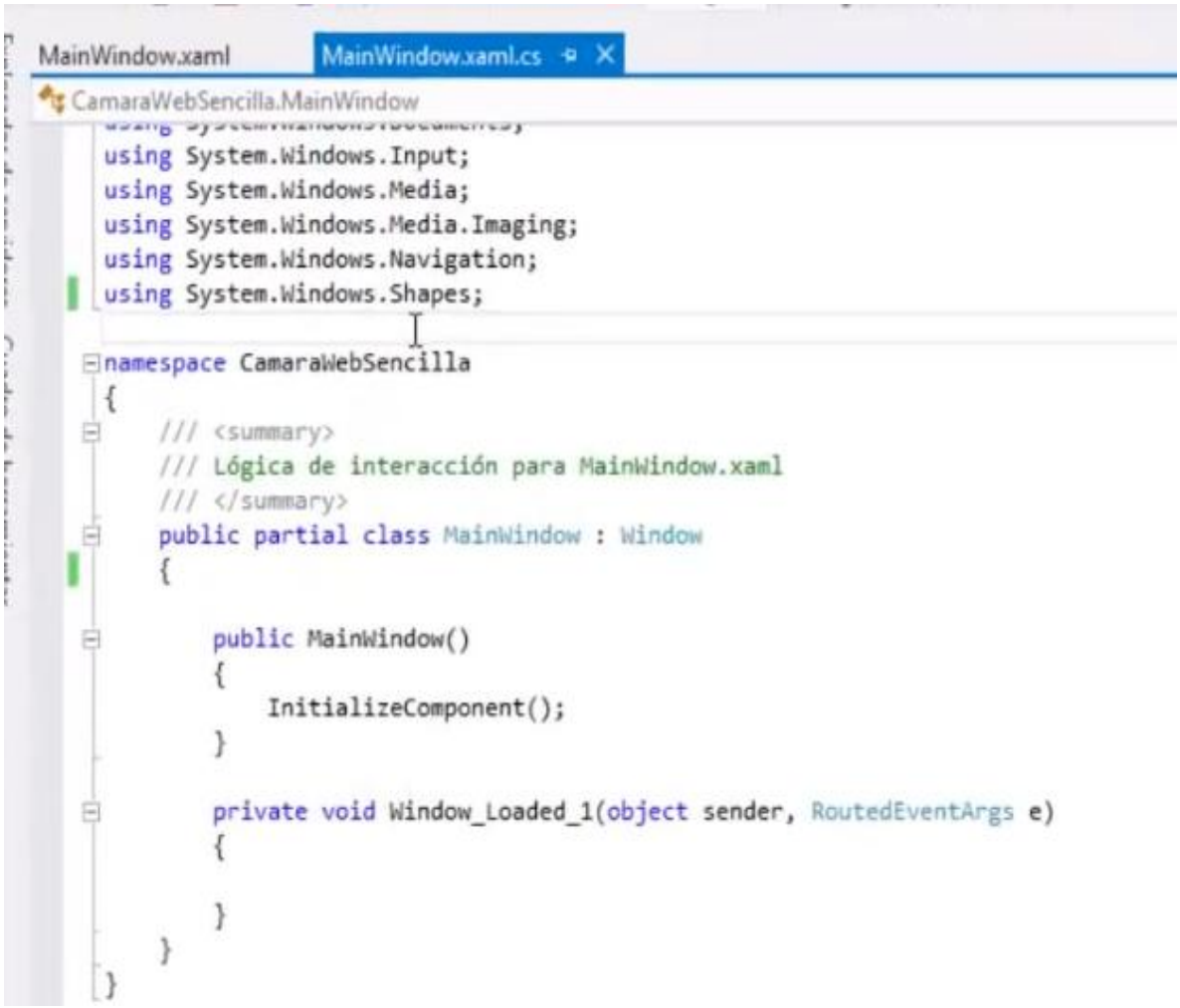

Figura 4.16 código fuente

Con este código estamos almacenado dentro de la variable de objeto un dato de nuestra Kinect a que nos referimos con esto pues es como si estuviéramos almacenado nuestra Kinect en esta variable para hací poder llamar sus funciones

Como las cámara los micrófonos o el acelerómetro los corchetes y el cero dentro indican un array el cual estamos llamando al elementos cero que en el mundo real seria el Kinect que tenemos conectado si conectáramos más de 1 Kinect y aprendiéramos a trabajar con ellos lo primero sería conocerse el cual elementos es cada uno de ellos en este array

```
public MainWindow()
{
    InitializeComponent();
}

private void Window_Loaded_1(object sender, RoutedEventArgs e)
{
    miKinect = KinectSensor.KinectSensors[0];
}
}
```

FIGURA 4.17 almacenando variables

Hasta este punto ya tenemos configurado nuestra kinect para que funciones en nuestro programa lo que tenemos que hacer es indicale que inicie usando el siguiente código

“miKinect.Start “

```
{
    InitializeComponent();
}

private void Window_Loaded_1(object sender, RoutedEventArgs
{
    miKinect = KinectSensor.KinectSensors.FirstOrDefault();
    miKinect.Start();
}
```

FIGURA 4.18 codigo para inializar el programa

Comezaremos a configurar nuestra cámara a color RGB para eso primero debemos habilitarla escribiéremos

“Mikinect.colorStream.Enable()”;

Con este colorStream.Enable damos inicio a la señal de envio de datos por parte de nuestro Kinect de la cámara de color RGB estos datos que se repiensan a mandar será utilizados como stream o flujos de datos dentro de las propiedades de enable también podemos indicarles varios tipos de formatos de imagen que nos pueden ser enviados algo que veremos después por Haro trabajaremos el que está por defecto

```
KinectSensor miKinect;

public MainWindow()
{
    InitializeComponent();
}

private void Window_Loaded_1(object sender, RoutedEventArgs e)
{
    miKinect = KinectSensor.KinectSensors.FirstOrDefault();
    miKinect.Start();
    miKinect.ColorStream.Enable();
}
```

FIGURA 4.19 agregar la camara RGB

RgbResolution640x480Fps30

Esto nos indica que la Kinect nos enviara unos datos de formato RGB con una resolución de 640 x 480 pixeles y a una velocidad de 30 FPS

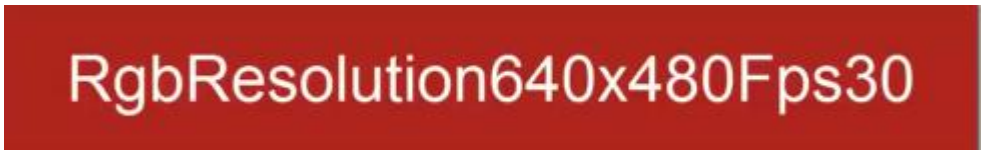

FIGURA 4.20 resolucion de camara

Seguido le indicamos a nuestra aplicación que debe hacer cuando cuando sea capturado este flujo de datos o video a color y lo haremos con un controlador de eventos escribiendo lo siguiente

“miKinect.ColorFrameReady += miKinect.ColorFrameReady;”

Podemos ver como senos creado la estructura del eventArgs una breve explicación al momento que el eventArgs captura el flujo de datos del colorStream este lo almacena en su propiedad E la cual podemos cambiar el nombre si queremos con

todo el código que ya tenemos deberíamos tener una Kinect mandado estar mandado figura 4.21 flujos de datos de video a color sin ningún erros

```
    }

    private void Window_Loaded_1(object sender, RoutedEventArgs e)
    {
        miKinect = KinectSensor.KinectSensors.FirstOrDefault();
        miKinect.Start();
        miKinect.ColorStream.Enable();
        miKinect.ColorFrameReady += miKinect_ColorFrameReady;
    }

    void miKinect_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
    {
        throw new NotImplementedException();
    }
}
```

FIGURA 4.21 flujos de datos de video a color

Pero hará debemos saber qué hacer con ese flojo de datos para convertirlo e video y mostrarlo en nuestra aplicación vamos agregando código al eventGander la sintaxis using permite eliminar un objeto el cual creamos dentro de sus paréntesis

Este objeto sra eliminado cuando search ecuntre nuestro codigo y la llave de sierre de la sintaxis using por que utilisamos esto lo utilizamos porque es importante que

se elimine el objeto de tipo “e. opencolorImagenFrame “ el cual se creara cada vez que kinect nos envíe un nuevo freind en sus flujo o de datos y como ya lo había mencionado el propiedad nos indica que nos estará mandado 30 freind cada segundo

Como vemos al tomar la propiedad enable nos estamos refieiendo al fujo de datos que enviaremos y con el método openColorImegeframe le indicamos que comience a recibir este flujo de datos y almacenarlo en la variable framesImagen para que la sintaxis using pueda eliminar y seguir recibiendo más flujo de datos bien con esta

```
        miKinect = KinectSensor.KinectSensors.FirstOrDefault();
        miKinect.Start();
        miKinect.ColorStream.Enable();
        miKinect.ColorFrameReady += miKinect_ColorFrameReady;
    }

    void miKinect_ColorFrameReady(object sender, ColorImageFrameReadyEvent
    {
        using (ColorImageFrame frameImagen = e.OpenColorImageFrame) {

        }
    }
```

Figura 4.22 para eliminar y seguir recibiendo

siguiente línea solo verificaremos framesImagen no tiene un valor nudo y si es verdadero nos retornara un valor nudo

```
    miKinect.ColorFrameReady += miKinect_ColorFrameReady;
}

void miKinect_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e
{
    using (ColorImageFrame frameImagen = e.OpenColorImageFrame()) {
        if (frameImagen == null) return;

        byte[] datosColor = new byte[frameImagen.PixelDataLength];

        frameImagen.CopyPixelDataTo(datosColor);

    }
}
```

FIGURA 4.23 verificaremos frames imágenes

Todo estos nos quiere decir que estamos recibiendo 30 imagen por segundo y nosotros tenemos que trabajar con ella digamos que tenemos almacenada la primera variable para poder mostrarla en nuestro control imesh para poder mostrar el tamaño primero obtendremos de dicha imagen para esto la clase ColorImagenFrame nos probé con una provee con una propienda llamada pixedegal la cual contiene el número de bayte y datos de video en la imagen el programa uso esto para determinar el buffer de datos de video a color

Ya por ultimo creamos solo creamos un byte con su respetivos propiedades y la colocamos en la propiedad source de nuestro control imesh que nombramos mostrar video

```
            byte[] datosColor = new byte[frameImagen.PixelDataLength];

            frameImagen.CopyPixelDataTo(datosColor);

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

FIGURA 4.24 codigo para imesh

Ya tenemos nuestra aplicación terminada podemos iniciarla esperaremos unos cuantos segundos y cómo podemos observar nuestro programa está corriendo sin errores

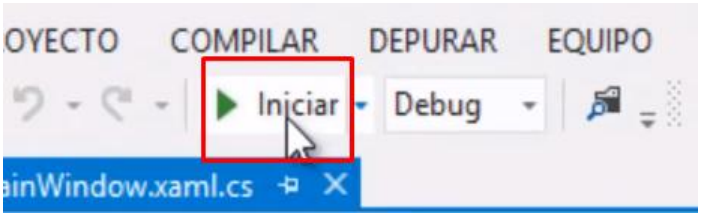

FIGURA 4.25 iniciar el programa

Agregaremos un boto a nuestra aplicación y donde poder guardad las capturas Y agregaremos unas líneas de código para poder hacer nuestras capturas y poder guardarlas

FIGURA 4.26 captura de imagen

Using System.IO;

```
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using Microsoft.Kinect;
using System.IO;
```

FIGURA 4.27 variables

Luego agregaremos una variable de tipo bool para que almacena un valor de verdadero y si no lo presionamos en paso

La segunda variable de tipo BitmapSoucer ya que al tomar la foto no es recomendable que usemos el mismo bitma que ya estábamos usado para mostrar el control imesh con esto evitaremos errores y no tener un erro al tomar la captura de imagen mediante estas líneas de código

```
bool grabarFoto;
BitmapSource bitmapImagen = null;

private void tomarFoto_Click(object sender, RoutedEventArgs e)
{
    grabarFoto = true;

    Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
    dlg.FileName = "capturaDeKinect";
    dlg.DefaultExt = ".jpg";
    dlg.Filter = "Pictures (.jpg)|*.jpg";

    if (dlg.ShowDialog() == true)
    {
        string nombreArchivo = dlg.FileName;
        using (FileStream stream = new FileStream(nombreArchivo, FileMode.Create))
        {
            JpegBitmapEncoder encoder = new JpegBitmapEncoder();
            encoder.Frames.Add(BitmapFrame.Create(bitmapImagen));
            encoder.Save(stream);
        }
```

FIGURA 4.28 codigo para captura de imagen

Dentro del evento click del botón que creamos le indicamos la variable boolean grabar foto es cambiada a truee y para checar esta variable

```
private void tomarFoto_Click(object sender, RoutedEventArgs e)
{
    grabarFoto = true;

    Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
    dlg.FileName = "capturaDeKinect";
    dlg.DefaultExt = ".jpg";
    dlg.Filter = "Pictures (.jpg)|*.jpg";
```

FIGURA 4.29 evento click del boton

Y para checar que nutro botón funciones agregaremos un condicional if

```
byte[] datosColor = new byte[framesImagen.PixelDataLength];

framesImagen.CopyPixelDataTo(datosColor);

if (grabarFoto)
{
    bitmapImagen = BitmapSource.Create(
        framesImagen.Width, framesImagen.Height, 96, 96, PixelFormats.Bgr32, null,
        datosColor, framesImagen.Width * framesImagen.BytesPerPixel);
    grabarFoto = false;
```

FIGURA 4.30 condicional if

Usando la condicional if y realizando la función y si es verdadero ara la siguiente condición con bitmapImagen y guardara la imagen que se está tomando en ese momento.

```
byte[] datosColor = new byte[framesImagen.PixelDataLength];

framesImagen.CopyPixelDataTo(datosColor);

if (grabarFoto)
{
    bitmapImagen = BitmapSource.Create(
        framesImagen.Width, framesImagen.Height, 96, 96, PixelFormats.Bgr32, null,
        datosColor, framesImagen.Width * framesImagen.BytesPerPixel);
    grabarFoto = false;
}

    colorStream.Source = BitmapSource.Create(
        framesImagen.Width, framesImagen.Height,
        96,
        96,
        PixelFormats.Gray16,
        null,
```

FIGURA 4.31 codicional bitmap

Y por ultimo ponemos la variable grabarFoto = false; para poder procesar la imagen y poder continua con el flujo de datos y poder continuar y guardando en el control imesh

```
byte[] datosColor = new byte[framesImagen.PixelDataLength];

framesImagen.CopyPixelDataTo(datosColor);

if (grabarFoto)
{
    bitmapImagen = BitmapSource.Create(
        framesImagen.Width, framesImagen.Height, 96, 96, PixelFormats.Bgr32, null,
        datosColor, framesImagen.Width * framesImagen.BytesPerPixel);
    grabarFoto = false;
}

    colorStream.Source = BitmapSource.Create(
        framesImagen.Width, framesImagen.Height,
        96,
        96,
        PixelFormats.Gray16,
```

FIGURA 4.32 procesar la imagen

Veremos cómo se guarda la imagen dentro del botón click podemos ver que llamamos a la clase 6 “SaveFileDialog” con esta línea de código se abrirá un dialogo con el cual nos especificara en formato para la imagen y donde guardarlo

```
Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
dlg.FileName = "capturaDeKinect";
dlg.DefaultExt = ".jpg";
dlg.Filter = "Pictures (.jpg)|*.jpg";

if (dlg.ShowDialog() == true)
{
    string nombreArchivo = dlg.FileName;
    using (FileStream stream = new FileStream(nombreArchivo, FileMode.Create
    {
        JpegBitmapEncoder encoder = new JpegBitmapEncoder();
        encoder.Frames.Add(BitmapFrame.Create(bitmapImagen));
        encoder.Save(stream);
    }
```

FIGURA 4.33 especificar el formato de guardar la captura de la imagen

Y de esta manera queda la sintaxis del código para dar el formato en el que se guardara.

```
dlg.FileName = "capturaDeKinect";
dlg.DefaultExt = ".jpg";
dlg.Filter = "Pictures (.jpg)|*.jpg";
```

FIGURA 4.34 espesificasion del formato de archivo

4.0.2 Creación del código de la cámara de profundidad la Kinect

Abrimos visual studio damos nuevo proyecto y seleccionamos aplicación WPF

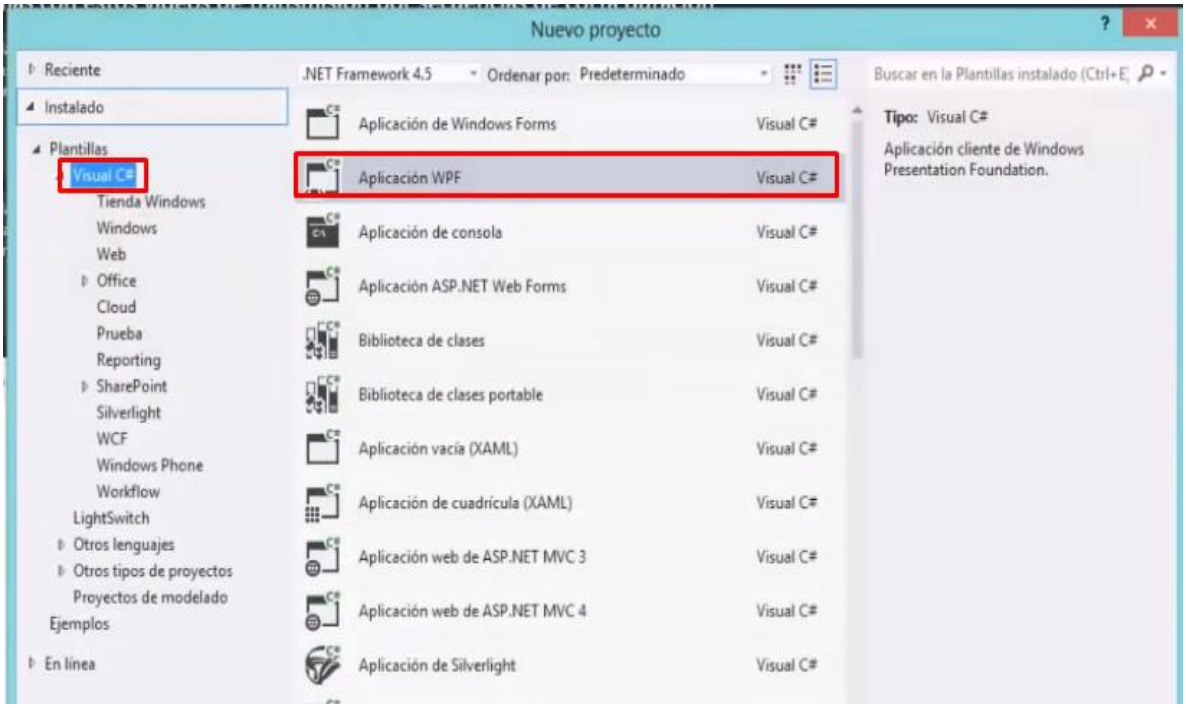

FIGURA 4.35 agregar una aplicasion WPF

Le pondremos de nombre CamaraProfundidad damos aceptar

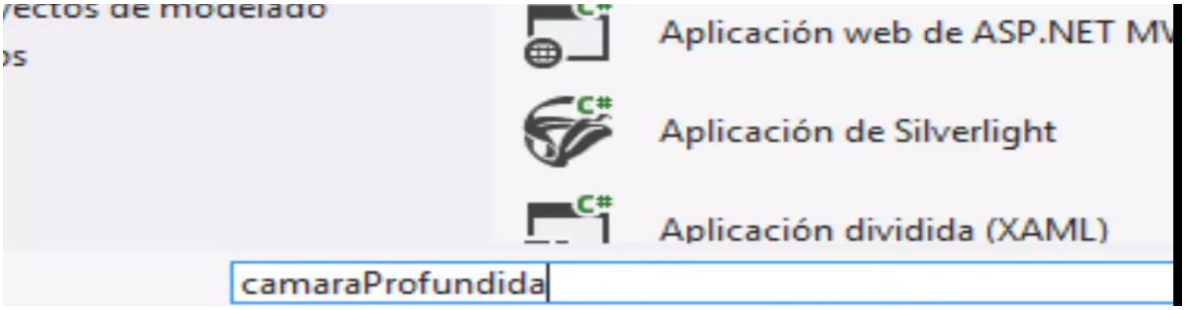

FIGURA 4.36 agregar nonbre del proyecto

Luego agregamos un control imagen para poder visualizar la profundidad de la captura de la Kinect y distinguirlas con colores

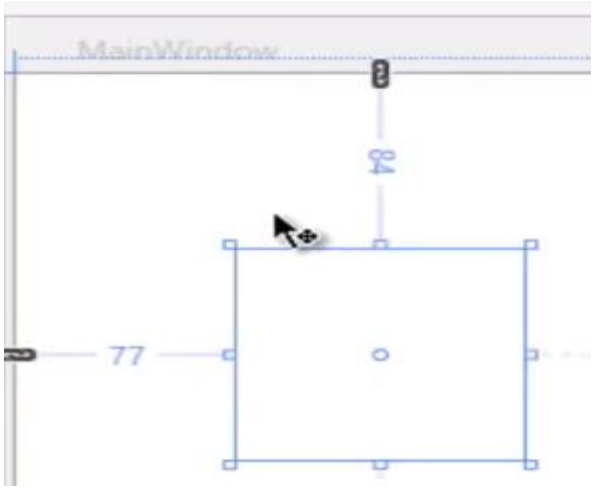

FIGURA 4.37 agregado un control de imagen

Hay que agregar su references de la librería de SDK de Kinect

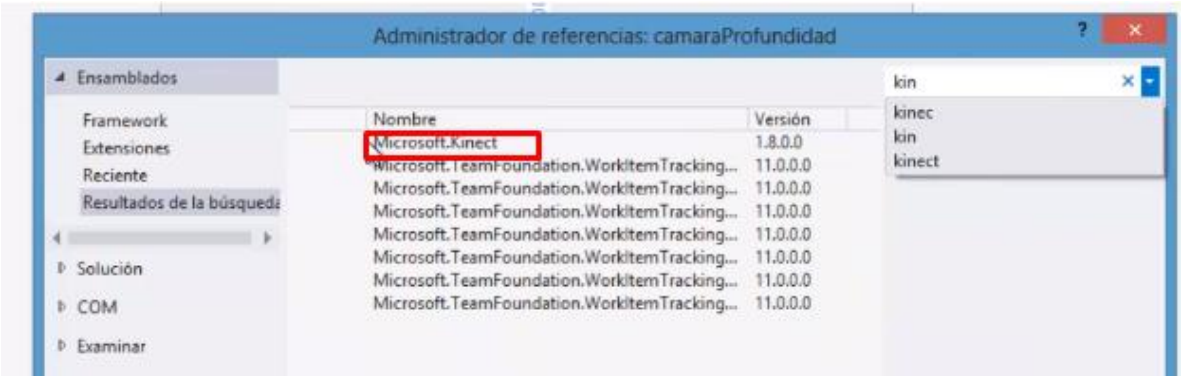

FIGURA 4.38 agregar librería de kinect

vamos al código como primero agregamos el **using Microsoft.Kinect;**

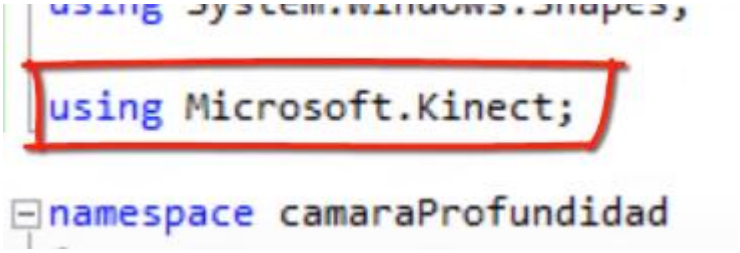

FIGURA 4.39 agregar la librería de kinect

Luego agregamos creamos una variable de la clase KinectSensor

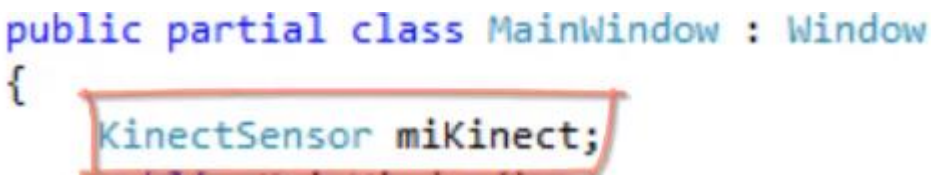

FIGURA 4.40 agregar variable sensor

Luego creamos un evento load del formulario para que al iniciar la aplicación el síguete código se ejecute código que ya sabemos cómo funciona

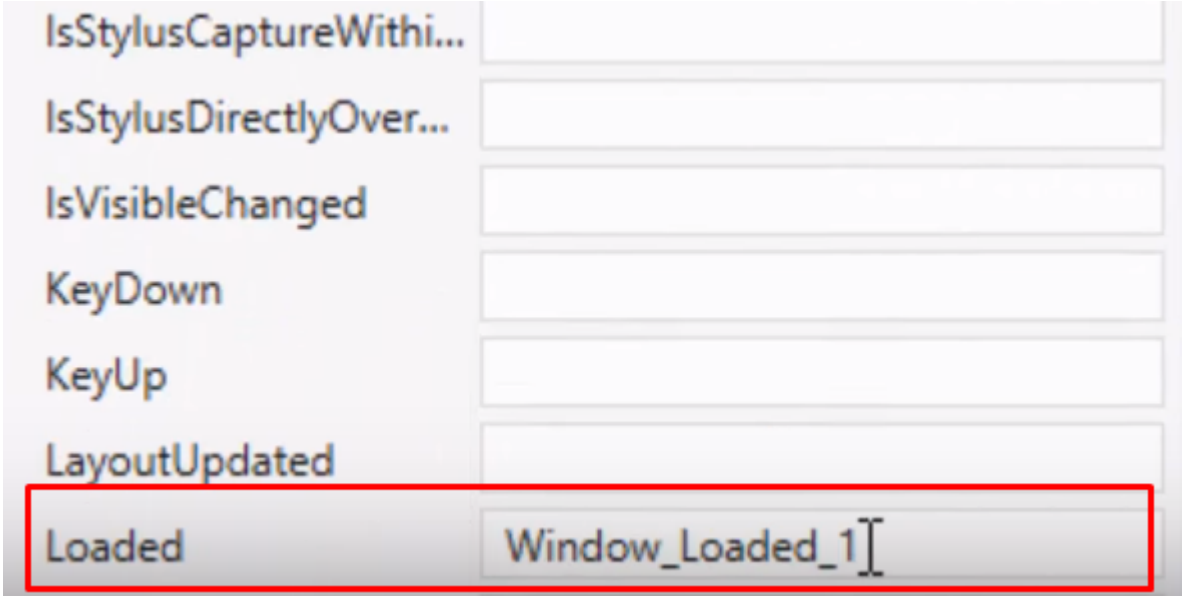

FIGURA 4.41 evento load

Para habilitar la cámara de profundidad usaremos la siguiente línea de codigo

miKinect.DepthStream.Enable();

Luego inicializamos el kineck con start

```
miKinect = KinectSensor.KinectSensors.FirstOrDefault();
miKinect.DepthStream.Enable();
miKinect.Start();
miKinect.DepthFrameReady += miKinect_DepthFrameReady;
```

FIGURA 4.42 hablilitar la camara de profundidad

Y por último crearemos la variable que usaremos para mostrar los datos de flujo para mostrar los batos de distancias y el color

Como configurar la distancias y dar la indicación de cada color esta son las líneas de codigo las cual se en cargar de captar los colores de la distancia usando las condiciones IF para dar los colores por distancias

En este código solo tenemos la configuración de color azul

```
framesDistancia.CopyPixelDataTo(datosDistancia);

int posColorImagenDistancia = 0;

for (int i = 0; i < framesDistancia.PixelDataLength; i++ )
{
    int valorDistancia = datosDistancia[i] >> 3;

    if (valorDistancia == miKinect.DepthStream.UnknownDepth) {
        colorImagenDistancia[posColorImagenDistancia] = 0;
    framesDistancia.CopyPixelDataTo(datosDistancia);
```

FIGURA 4.43 condicion if para dar los colores por dictancias

De esta manera quedara la función de else if para declarar cada color

```
for (int i = 0; i < framesDistancia.PixelDataLength; i++ )
{
    int valorDistancia = datosDistancia[i] >> 3;

    if (valorDistancia == miKinect.DepthStream.UnknownDepth) {
        colorImagenDistancia[posColorImagenDistancia++] = 0; //Azul
        colorImagenDistancia[posColorImagenDistancia++] = 0; //Verde
        colorImagenDistancia[posColorImagenDistancia++] = 255; //Rojo
    }
    else if (valorDistancia == miKinect.DepthStream.TooFarDepth) {
        colorImagenDistancia[posColorImagenDistancia++] = 255; //Azul
        colorImagenDistancia[posColorImagenDistancia++] = 0; //Verde
        colorImagenDistancia[posColorImagenDistancia++] = 0; //Rojo
    }
    else if (valorDistancia == miKinect.DepthStream.TooFarDepth)
    {
        colorImagenDistancia[posColorImagenDistancia++] = 0; //Azul
        colorImagenDistancia[posColorImagenDistancia++] = 255; //Verde
        colorImagenDistancia[posColorImagenDistancia++] = 0; //Rojo
    }
    posColorImagenDistancia++;
```

FIGURA 4.44 ejemplo de comoqueda funcion if

Y un último caso usaremos un parámetro por los 3 casos anteriores si no se cumple la condijo de unos de los 3 pondremos uno a adicional para las propiedades de distancias, no identificada la distancias y muy alejada la distancia para eso agregaremos un else con este else vamos agregar un valor de luminosidad para asignar en los pixeles y eso nos dará más iluminación para eso aremos un areglo para convertir los valor de distancia en color

```
else {
    byte byteDistancia = (byte)(255 - (valorDistancia >> 5));
    colorImagenDistancia[posColorImagenDistancia++] = byteDistancia; //Azul
    colorImagenDistancia[posColorImagenDistancia++] = byteDistancia; //Verde
    colorImagenDistancia[posColorImagenDistancia++] = byteDistancia; //Rojo
}
posColorImagenDistancia++;
```

FIGURA 4.45 agregar casos para no repetir la funcion

Listo es todo ya lo podemos ejecutar el programa

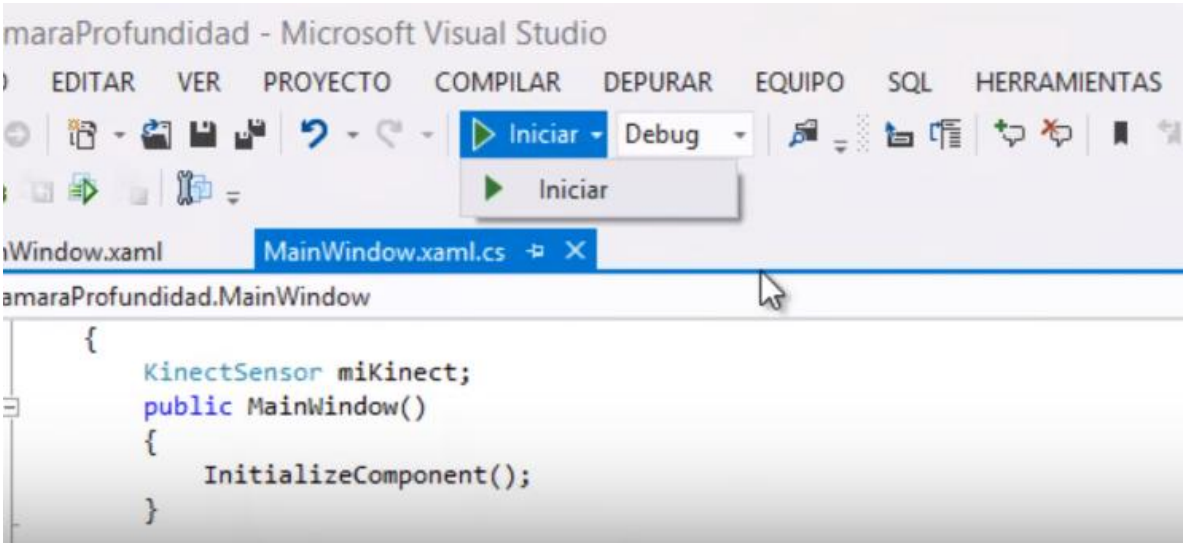

FIGURA 4.46 iniciar el programa

Y este es el resultado de la programación como se puede observar el color rojo es lo que no distingue la Kinect los pixeles verde son las distancias que están demasiada cercas del Kinect y las distancias más alejadas son color azul

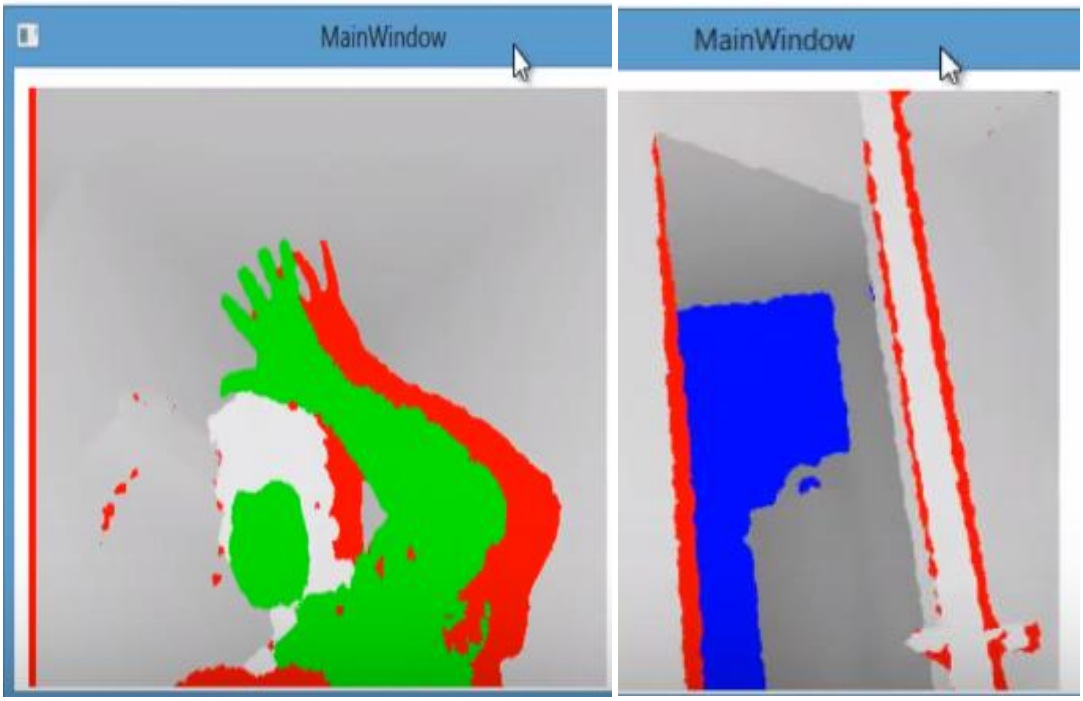

FIGURA 4.47 ejemplo de la camara de profndidad

4.0.3 Kinect - Detectando Esqueletos

Kineck nos provee con otro tipo de flujo de datos

Esta al igual que el colorStream nos estarán aviando un flujo de datos el cual será las posiciones en 3 dimensiones de una figura parecida al de un esqueleto humano estos esqueletos son creados por el SDK de Kinect en una combinación de ambas tecnologías de la cámara de color y de profundidad en especial la cámara de profundad ya que con estas es con la que capturamos las distancia de cada pixel

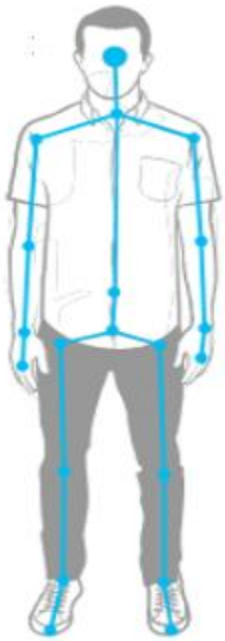

FIGURA 4.48 cuerpo humano y sus articualaciones

Skeletal tracking significa *seguimiento de esqueleto,* este está basado en un algoritmo que logra identificar partes del cuerpo, gestos y/o posturas de quienes están en el campo de visión del sensor.

Kinect es capaz de detectar dos tipos de Esqueletos, el default y el seated

Default: Es el esqueleto predeterminado en el cual Kinect identifica las 20 partes del cuerpo fundamentales en las extremidades y en el rostro del cuerpo humano.

Seated: Es el esqueleto que Kinect detecta si el usuario está sentado, solo cuenta con 10 puntos de identificación, se utiliza comúnmente cuando el usuario está

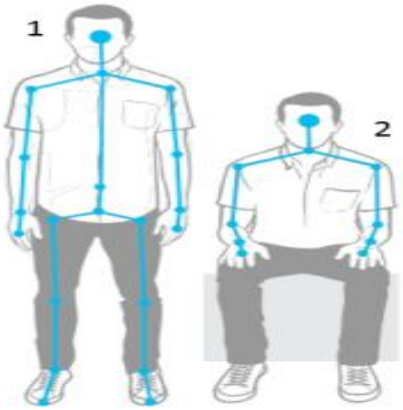

FIGURA 4.49 deteccion de 2 cuerpo humano

Ahora bien, cada punto de los esqueletos recibe un nombre, nombre con el cuál podrás indicarle a Kinect que si es detectada tal parte tome la fotografía

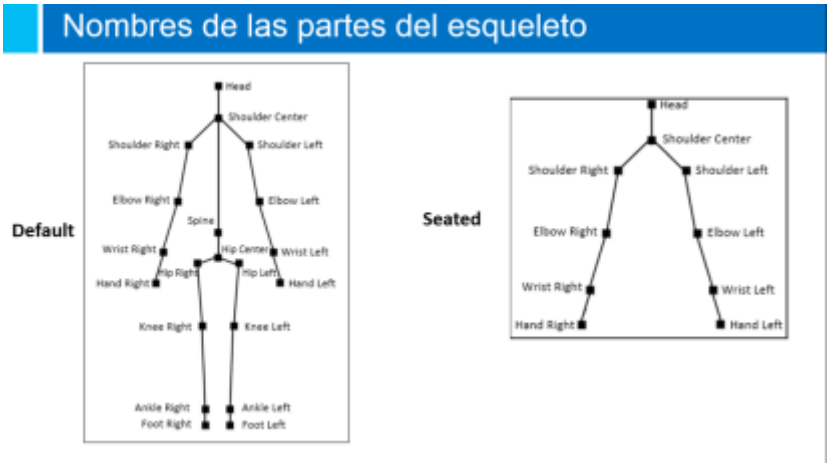

FIGURA 4.50 las articualacione de cuerpo huamano

4.0. 4Usaremos la propiedad ClippedEdges para hacer la captura de esqueleto

```
Public enum FrameEdges
{
None  = 0,
Right = 1,
Left = 2,
Top = 4,
Boottom = 8,
}
```

FIGURA 4.51 detección de cuerpo cuerpo humano

Pasos para capturar de datos

A. Habilitar el flujo de datos de esqueletos
B. Captura ese flujo de datos
C. Crear el array que almacenara los datos de esqueletos,
D. Copiar los datos a ese array de esqueletos
E. Recorrer cada uno de los frames de esqueletos con un loop.
F. Verificar que se detecte el esqueleto
G. Verificar que tenga buena calidad de captura.
H. Elegir una de las articulaciones de ese esqueleto
I. Utilizar alguna propiedad de esa articulación (posición)
J. Mostrar resultados en 3D.

Comenzamos creando un nuevo proyecto en visual studio será WPF y le

```
<Window x:Class="DeteccionEsqueletos.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Width="850" SizeToContent="Height">
    <Grid>

    </Grid>
```

FIGURA 4.52 tamaño del formulario

pondremos detector de esqueletos al formulario solo le indicaremos un width de 850 y borramos el height y agreagamos sizeToContent = “Height” para que el tamaña del contenido del formulario se igual al tamaño

Creamos un evento de loaded que nos servirá mas después

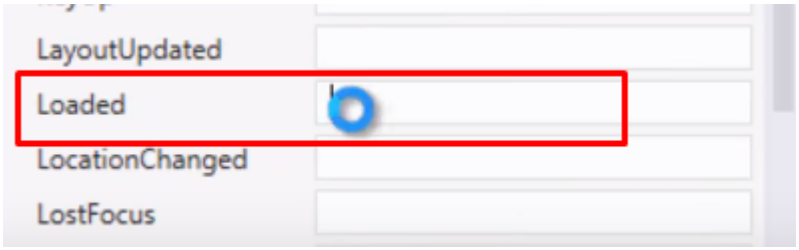

FIGURA 4.53 evento loaded

Ya adentro del contenido del formulario vamos agregar 2 elementos textblock el primer textblock mostrara si se detecta el esqueleto y la articulación que deseamos elegir y el segundo textblock va mostrar unos mensajes que nos indicaran a donde

nos tenemos que mover para que la aplicación tenga una buena captura del esqueleto

Ya que agregamos los textblock nos vamos al formulario le borraremos todas su propiedades que tenga los textblock para agregar la siguiente primero su identificador que le pondremos textblockEstatus y al segundo textBlockCaptura luego podemos ponerle una propiedad que será el text que será su contenido qe tendrá al inicio de la aplicación lo irmoes modificando como se valla enjutando en el primer text ponemos “Estaturas de esqueleto” y el segundo “captura de calidad

Por ultimo indicamos un “HorizontalAlignmant” = center para que se coloque en el centro del formulario y un fontSize de 72 para que la legreta se distinga

Quedando de esta manera las cajas de texto

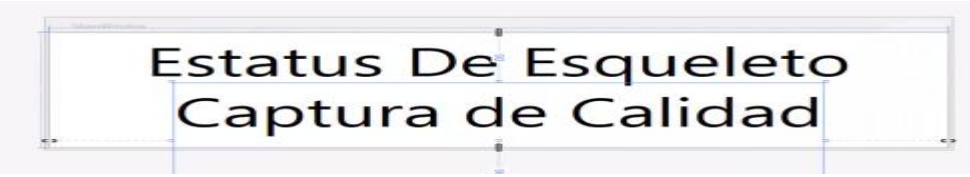

FIGURA 4.54 caja de texto

Y recuerde agregar las referencias de la Kinect bueno continuaremos con el código similares a las anteriores que hemos estado desarrollando uno de las diferencias al habilitar es que la línea de código tiene que ser como mi esqueleto

```
miKinect.SkeletonStream.Enable();
```

Luego se crea readyEventARGS Primero creamos 3 variables

La primera la llamaremos string mensaje no hay datos de esqueletos

Segunda variable será string mensajeCalidad del textblock 2

La tercera variable será una array para alamasenar los datos de skeleton [] esqueleto

```
string mensaje = "No hay datos de esqueleto";
string mensajeCalidad = "";
Skeleton[] esqueletos = null;
```

FIGURA 4.55 instrucción para la detección de cuerpo humano

Luego con seguiremos escribiendo la sintaxis de using (skeletonFrame) con esto indicamos que empieza almacenar las fren para guardarlos en dicho objeto luego hacemos una comprobación de que estamos recibiendo capturas de datos usando un FramesEsqueleto asemos una verificación de seguridad por si el array no tiene ningún valor y usaremos igual FOREACH para hacer una comparación

En el primer textblock lo utilizadores para poder agregar los vares de x y la

```
if (esqueletos == null) return;

foreach(Skeleton esqueleto in esqueletos){
    if (esqueleto.TrackingState == SkeletonTrackingState.Tracked) {
        Joint jointCabeza = esqueleto.Joints[JointType.Head];
```

FIGURA 4.56 detectado cierta parte del esqueleto humano

posición y hacer una conversión por que los valores tridimensional son muy grandes saldrán muy grandes y nosotros los usaremos en string brazo “X:{0:0.0}” Y:{0:0.0}” Z:{0:0.0}” Y le eindicamos que muestro la variable mensaje en el textblock

```
        mensaje = string.Format("Cabeza: X:{0:0.0} Y:{0:0.0} Z:{0:0.0}", posicionCa
    }
}
textBlockEstatus.Text = mensaje;
```

FIGURA 4.57 agregar la dirección y función para la detección del cabeza

Listo y podemos ejecutarlo podemos ver como se muestra que no detectado ni un dato del esqueleto

Pero si nosotros nos posicionamos en un buen lugar nos indicara que a detectado la calidad de detección

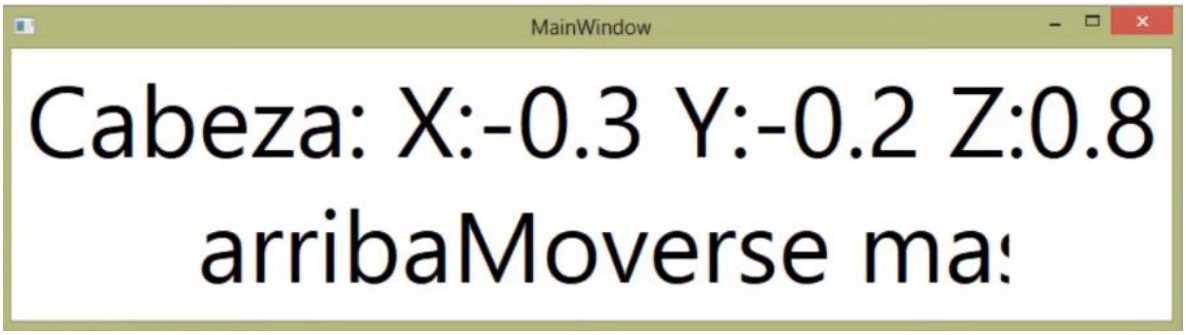

FIGURA 4.58 interfaz de la detección de la cabeza con sus coordenadas

4.0. 5 Diseño del interfaz de la aplicación

Para crear un nuevo proyecto vamos a Archivo ---> FIGURA 4.59 Nuevo Proyecto

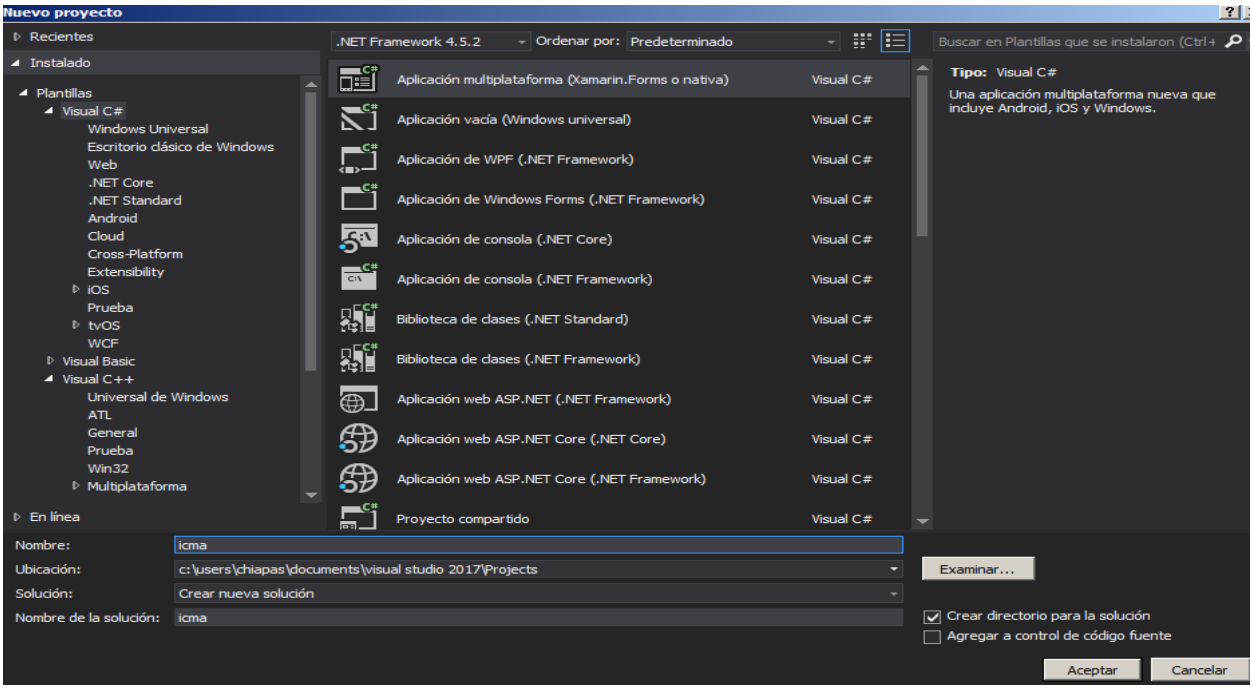

FIGURA 4.59 nuevo proyecto

Agregaremos 2 botones una ladel y una imagen

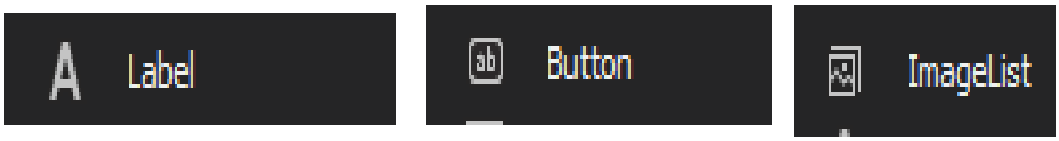

FIGURA 4.60 herramientas de visual

La caja de texto de ladel

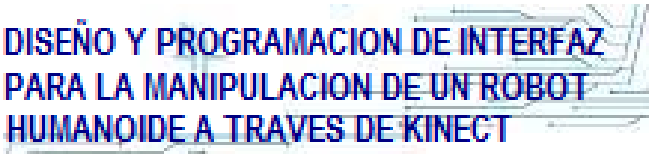

FIGURA 4.61 caja de texto

Botón de inicio para la sesión

FIGURA 4.62 botón

Codigo de botón

```
}
1 referencia
private void IniciarMa_Click(object sender, EventArgs e)
{
    Form.ActiveForm.Hide();
    Form2 a = new Form2();
    a.Show();
}
```

FIGURA 4.63 código para botón

Botón de salir

FIGURA 4.64 botón para salir

Código del botón

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    DialogResult Salir;
    Salir = MessageBox.Show("Realmente desea salir de la aplicacion?", "Salir de la Aplicacion", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
    if (Salir == DialogResult.OK)
    {
        Application.Exit();
    }
```

FIGURA 4.65 código para el botón para salir

Diseño de la interfaz

FIGURA 4.66 DISEÑO DE USUARIO

Código completo de la interfaz

```
public partial class Inicial : Form
   {
```

```
        public Inicial()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            DialogResult Salir;
            Salir = MessageBox.Show("Realmente desea salir de la aplicacion?", "Salir
de la Aplicacion", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
            if (Salir == DialogResult.OK)
            {
                Application.Exit();
            }
        }
        private void IniciarMa_Click(object sender, EventArgs e)
        {
            Form.ActiveForm.Hide();
            Form2 a = new Form2();
            a.Show();
        }
        private void pictureBox2_Click(object sender, EventArgs e)
        {
        }
        private void label1_Click(object sender, EventArgs e)
        {
        }
        private void Inicial_Load(object sender, Eventers e)
        {
        }
```

FIGURA 4.67 código compilado

Para crear un nuevo proyecto vamos a Archivo ---> Nuevo Proyecto

Luego debemos seleccionar 2 Labels. (Podemos darle doble click o arrastrarlas

TextBox. (Podemos darle doble click o arrastrarlas)

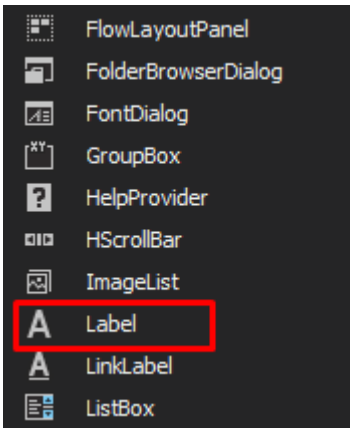

FIGURA 4.68 herramientas

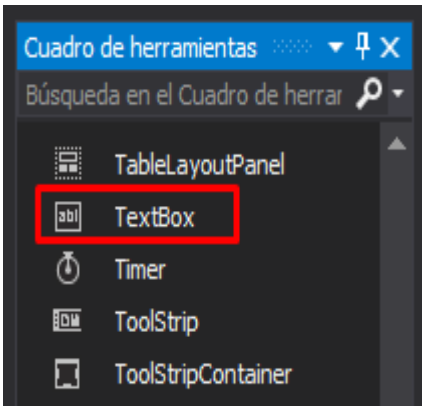

FIGURA 4.69 herramientas

1 Botón. (Podemos darle doble click o arrastrarlas)

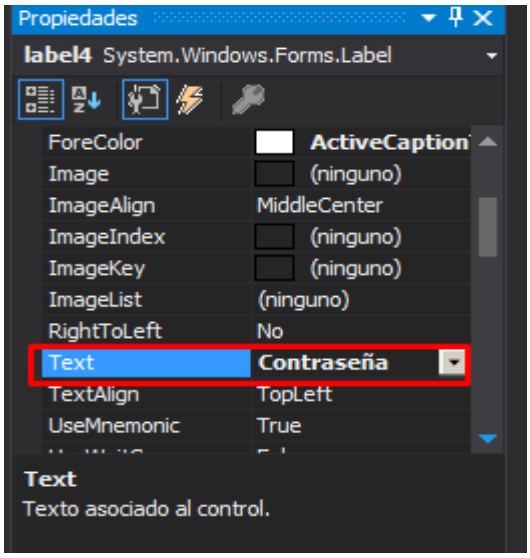

FIGURA 4.70 dar nombre a caja de texto

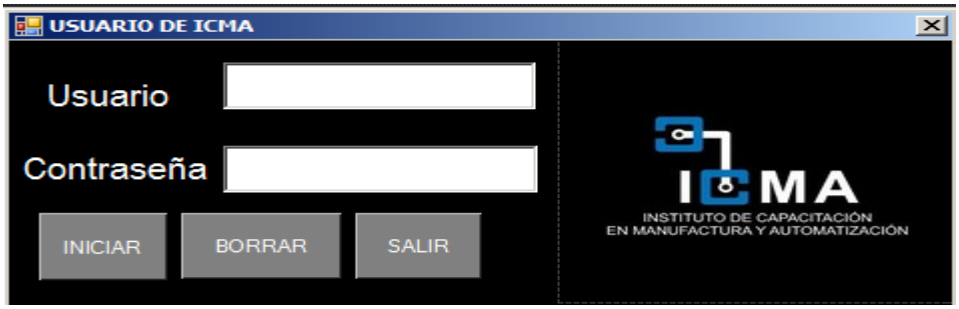

FIGURA 4.71 interfas de usuario

Nos tendría que quedar así.

Para cambiarle el texto a los Labels y al boton, lo seleccionamos y abajo a la derecha, en la ventana de propiedades nos vamos a "Text" y ahí lo cambiamos.

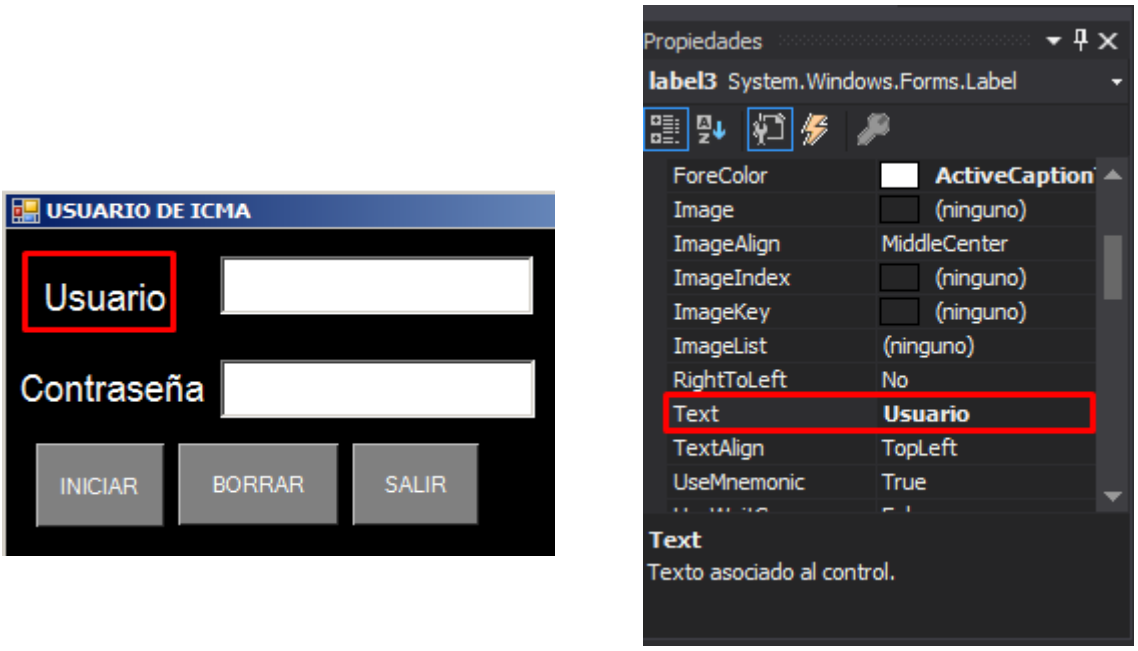

FIGURA 4.72 agregar nombre a la caja de texto

Le dan doble click al boton de inicio y les aparecerá algo asi.

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        If TextBox1.Text = "" And TextBox2.Text = "" Then
            MsgBox("Los campos se encuentran vacios")
        Else
            If TextBox1.Text = "" Then
                MsgBox("El campo USUARIO se encuentra vacio")
            Else
                If TextBox2.Text = "" Then
                    MsgBox("El campo CONTRASEÑA se encuentra vacio")
                Else
                    If TextBox1.Text = "TuUsuario" And TextBox2.Text = "TuContraseña" Then
                        Me.Hide()
                        form2.show()
                    Else
                        MsgBox("Los datos son incorrectos")
                    End If
                End If
            End If
        End If
    End Sub
End Class
```

FIGURA 4.73 linea de codigo para usuario y contraseña

Deben cambiar las partes que dicen "TuUsuario" y "TuContraseña", por su usuario y contraseña Ahora, debemos crear otro formulario, que es a donde nos llevara, luego de iniciar sesion. Para ello, damos clic derecho en "Login Básico"

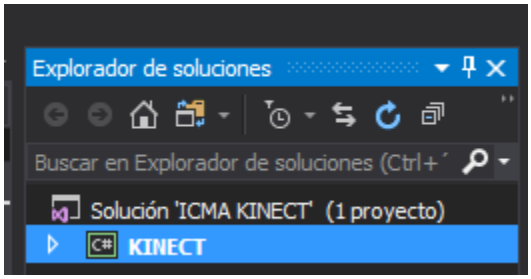

FIGURA 4.74 solucion del explorador de nuestro trabajo

Agregamos una función de time para hacer la animación de donde carga una barra

Y una función de incremento para el inicio y cargue en porcentaje

FIGURA 4.75 interfas de inializar la palicasion

La siguiente sintaxis del código nos hace la suma de + 1 en incremento al llegar al 100 % salta a nutra ventana seleccionada

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (progressBar1.Value < 100)
    {
        progressBar1.Value = progressBar1.Value + 1;
        label1.Text = "iniciando " + progressBar1.Value + " %";

    }
    else
    {
        timer1.Enabled = false;
        label1.Text = "Carga Completa";
        Form3.ActiveForm.Hide();
        Form4 d= new Form4();
        d.Show();
    }
}

1 referencia
private void button1_Click(object sender, EventArgs e)
{

    DialogResult result;
    timer1.Enabled=false;
    DialogResult Timer;
    Timer=MessageBox.Show("¿Desea cancelar la carga?", "Cancelar",MessageBoxButtons.OKCancel, MessageBoxIcon.Question );
    if (Timer == DialogResult.OK)
    {
        Form3.ActiveForm.Hide();
        Form2 A = new Form2();
        A.Show();
        MessageBox.Show("Carga Cancelada", "Cancelado", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        timer1.Enabled = true;
    }
}
```

FIGURA 4.76 lineas de codigo de intefaz cargar de barra

4.0.6 Menú de las aplicaciones de Kinect

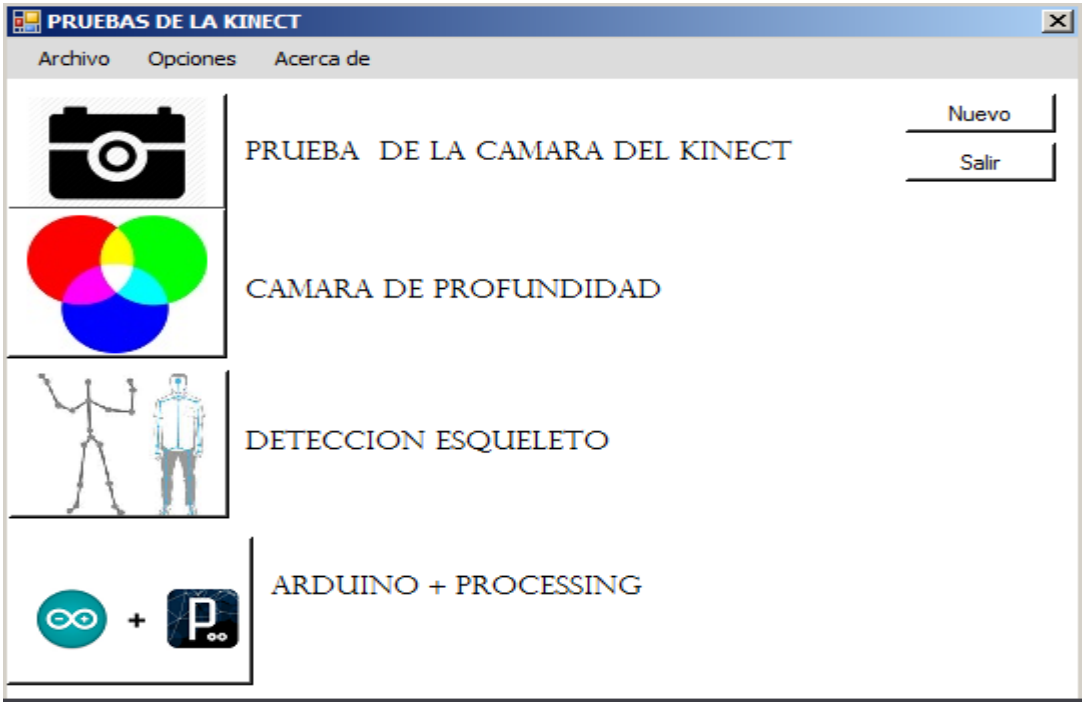

FIGURA 4.77 menu de aplicasion

FIGURA 4.78 menu cuenta con 3 opciones

Agregaremos 4 botones para nuestras aplicaciones y 2 botones más para salir y nuevo.

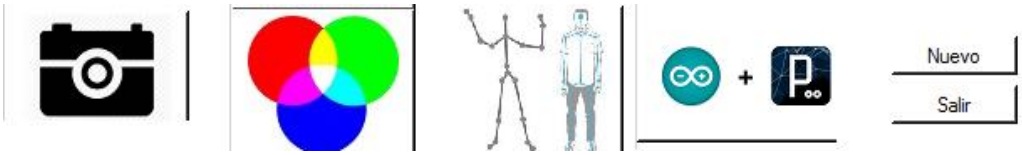

FIGURA 4.78 cuenta con 4 botones

Agregaremos 4 label para los textos donde indicamos que hacen los botones por que no importa la ilustración

Conclusiones

Dentro de las pruebas se observó que el movimiento del robótico puede ser un poco brusco, debido a que las coordenadas entregadas por el Kinect no son del todo constantes dentro de cada imagen entregada por el sensor, aun así, se pudieron tomar algunas piezas de pequeño tamaño y peso ligero, demostrando que el robót podría llegar a transportar piezas u objetos pequeños como los utilizados en las pruebas los cuales tienen un ancho menor a la apertura total de la pinza

El tiempo promedio para que un usuario que desconoce el funcionamiento del prototipo pueda utilizar el robot, depende la habilidad de la persona para acostumbrarse al control del robot.

En cuanto a la respuesta del sistema se obtuvieron resultados favorables, el robot se mueve a una velocidad que asimila ser paralela a la velocidad con la que el usuario se mueve.

Una de las limitaciones del proyecto, como se ha mencionado anteriormente, es que ninguno de las articulaciones de libertad alcanza el máximo movimiento permitido por los servomotores (180 grados), debido a que la flexibilidad en los movimientos del robot de la mayoría de las personas, no se ajusta a dicho movimiento, delimitando como consecuencia el movimiento del robot.

Finalmente, este trabajo de pasante produjo un una interfaz para un robot humanoide con ciertos grados de libertad, que puede ser controlado de forma natural por cualquier persona que cuente con las extremidades requeridas, sin necesidad de utilizar palancas o botones.

Observaciones

El cuarto donde se utilice el sensor es de gran importancia, se debe procurar que el sensor proyecte sobre una superficie rígida y no transparente o reflejante (vidrios, etc.), además de que la habitación debe ser un espacio en donde no afecte la luz solar de forma directa, por ejemplo colocar el sensor cerca de una ventana o puerta. La altura donde se coloque el sensor debe estar dentro de los rangos que indica el fabricante (0.60m a 1.50m), ya que al no cumplirse esta especificación, el sensor adquiere y entrega los datos de forma incorrecta debido a la mala proyección que ejecuta el sensor infrarrojo sobre la superficie cuando se coloca el sensor en una altura fuera del rango especificado.