CENTRO DE INGENIERIA Y DESARROLLO INDUSTRIAL

PROYECTO INDUSTRIAL TERMINAL

POSICIONAMIENTO TRIDIMENSIONAL DE OBJETO VIRTUAL CON VISIÓN ARTIFICIAL

TESIS

PARA OBTENER LA ESPECIALIDAD DE:

TECNÓLOG EN MECATRÓNICA

PRESENTA

Emmanuel Rodríguez Díaz

TUTOR ACADÉMICO Y DE PLANTA

Dr. Leonardo Barriga Rodríguez

CONACYT

Santiago de Querétaro, Qro.2017.

CONACYT

CIDESI

CENTRO DE INGENIERÍA Y DESARROLLO INDUSTRIAL

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN TECNOLÓGICA

AUTORIZACIÓN
PUBLICACIÓN EN FORMATO ELECTRÓNICO DE TESIS

Fecha: _____

El que suscribe
Alumno (a) Emmanuel Rodríguez Díaz……

CURP RODE940226HPLDZM00…… CVU 758733……..…

ORCID 0000-0002-0064-8005…….……

Correo electrónico (opcional)
emmanuel.rodriguez.diaz94@gmail.com.....……

Egresado (a) de Universidad Politécnica de Querétaro……

……

Autor de la Tesis titulada
"Posicionamiento tridimensional de objeto virtual con visión artificial

Por medio del presente documento autorizo[1] en forma gratuita y permanente a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita el acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en Página Web.
La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

Emmanuel Rodríguez Díaz

Nombre y firma del tesista

_____

[1] Ley Federal de Derechos de Autor
Para obtener tu ORCID regístrate en: **https://orcid.org/register**

Av. Playa Pie de la Cuesta No. 702 Desarrollo San Pablo. C. P. 76125
Santiago de Querétaro, Qro., México México.
Tel. 01 (442) 211 98 40
E-mail: jvite@cidesi.edu.mx

Índice

Índice de figuras

1. Resumen

El proyecto que se detalla a continuación es una aplicación diseñada para obtener coordenadas en tres dimensiones del cuerpo humano con el objetivo de poder posicionar en un espacio tridimensional el efector final de cualquier robot manipulador, es por eso que la aplicación desarrollada permite configuraciones de acuerdo al espacio de trabajo del robot a manipular, adicionalmente la persona que se encuentra trabajando con la aplicación puede determinar la apertura del efector final en caso de ser requerido así como también el estado del efector final ya sea activado o desactivado. Se detalla también como se lleva acabo la comunicación entre el robot y la aplicación para tener la mejor respuesta posible por parte del robot, siendo la aplicación o el centro de control del robot la que le indica a la aplicación aquí desarrollada cuando debe recibir información.

La obtención de puntos en tres dimensiones se lleva acabo mediante el sensor Kinect V1, mismo que es programado en Visual C# por medio de su kit de desarrollo de software (SDK).

2. Planteamiento del problema

Se requiere manipular la posición en un espacio tridimensional del efector final de un robot manipulador mediante la detección de puntos en tres dimensiones de un brazo humano mediante visión artificial, de esta manera en condiciones óptimas el efector final del robot manipulador será capaz de imitar el punto alcanzado por la mano de un humano.

3. Justificación

Debido a la necesidad de controlar un brazo robótico en un robot submarino en donde la única retroalimentación que tiene el operador es una imagen del espacio en el que esta operando el robot se requiere de un control remoto del brazo robótico para poder manipular objetos.

4. Objetivo general

Diseñar una aplicación en Visual C# que sea capaz de reconocer un brazo humano en tres dimensiones para con dichas coordenadas posicionar el efector final de un robot manipulador.

4.1. Objetivos específicos

1. Diseñar una aplicación en Visual C# que reconozca puntos en tres dimensiones de una mano humana de forma continua con la ayuda del sensor Kinect V1.
2. Obtener coordenadas en tres dimensión en función del espacio de trabajo del robot a manipular.
3. Implementar un protocolo de comunicación para mandar las coordenadas de la aplicación en Visual C# al robot.
4. Escribir un documento técnico sobre e proyecto realizado.

5. Antecedentes

La manipulación a distancia de robots es de gran utilidad en varias áreas de la ingeniería ya que se presenta como una oportunidad para aislar al ser humano de tareas potencialmente peligrosas o simplemente imposibles de realizar para un ser humano.

La manipulación remota de un robot involucra una parte indispensable que es la técnica utilizada para manipular el robot o en palabras mas simples como decirle al robot en donde se va a posicionar,dicha técnica puede ser mediante controles manuales como un Joystick, o como en este caso con imitación de movimientos mediante procesamiento de imágenes y detección de profundidad para el caso de 3 dimensiones.

La aplicación tecnológica para la cual esta diseñado este proyecto es la manipulación remota de un brazo robótico que se encuentra acoplado a un robot submarino, esto con el propósito de poder manipular objetos bajo el mar, sin embargo no es la aplicación en si lo que se trata dentro de este reporte técnico sino la aplicación que se desarrollo para poder manipular el efector final robot. Para poner en contexto la viabilidad de este proyecto a continuación se muestran algunas aplicaciones tecnológicas para controlar robots a distancia que se han implementado con éxito.

En la planta nuclear de Bugey ubicada en Lyon, Francia la alta radiación hace imposible que el ser humano lleve acabo tareas de mantenimiento como hacer inspección del reactor nuclear desde el interior por la radiación térmica, es por eso que un robot manipulado a distancia lleva acabo esta tarea como se puede ver en la figura que se muestra a continuación:

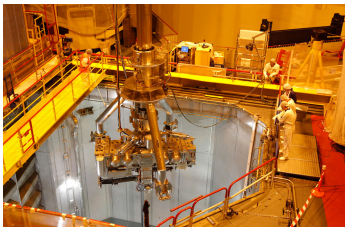

Figura 1: Robot para inspección del reactor de la planta nuclear Bugey

En cuanto a una aplicación más cercana a la planteada en este reporte tenemos el robot comercial ROV 110 el cual cuanta con un brazo robótico al frente para poder manipular o tomar objetos que se encuentren en la superficie sobre la cual se encuentre el submarino como se muestra en la siguiente imagen:

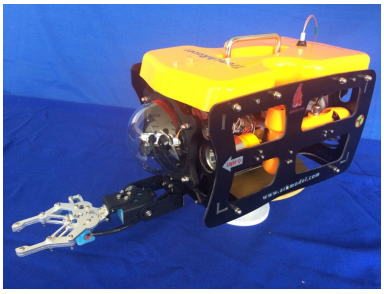

Figura 2: Robot submarino ROV110

Otro ejemplo de éxito de la manipulación remota de robots se encuentra en el área de la medicina, tal es el caso del Sistema Quirúrgico Da Vinci el cuál es un robot manipulador con tres brazos seriales y efectores lineales los cuales le permiten al doctor que este llevando acabo la cirugía hacer tareas de gran precisión y sin la necesidad de estar en el lugar donde se lleva acabo la operación, dichos brazos se controlan mediante el posicionamiento de un sistema mecánico que manda al robot la posición de las manos del cirujano, dichos brazos detectan la posición deseada mediante sensores de posición angular(encoder) ubicados en sus articulaciones lo cuál permite tener una gran precisión y estabilidad de la señal que se manda al robot.

Figura 3: Sistema Quirúrgico Da Vinci

Por otro lado es importante conocer la técnicas de reconocimiento de cuerpos en 3 dimensiones mediante visión artificial que se han desarrollado e implementado exitosamente en aplicaciones de ingeniería, por lo tanto a continuación se detallan algunos de los métodos de visión artificial más completos y eficientes.

Red de cámaras de alta velocidad : Este sistema utiliza como su nombre lo dice una serie de cámaras ubicadas en puntos estratégicos para poder determinar la ubicación de objetos en tres dimensiones, la razón por la que se utiliza más de una cámara es porque con una sola cámara es muy complicado determinar la profundidad de un objeto, por lo tanto al analizar la imagen de dos cámaras tal y como lo hace este sistema es posible hacer unas estimación bastante precisa sobre la posición de un objeto en tres dimensiones, dicha técnica es ampliamente utilizada en aplicaciones como la ubicación de drones en espacios tridimensionales, ubicación de objetos en deportes como el tenis o recientemente el fútbol soccer con el llamado .ºjo de halcón".

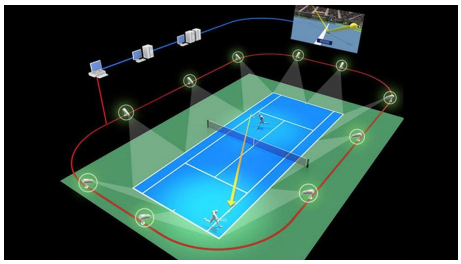

Figura 4: Red de cámaras de alta velocidad en cancha de tenis

Acelerómetro y giroscopio : Existen circuitos integrados que contienen tanto un giroscopio como un acelerómetro, lo cuál permite mediante una serie de filtros digitales saber cuanto avanza un objeto en 3 dimensiones y como se encuentra orientado dicho objeto, y aunque este método no implica visión artificial si ha sido utilizado para la manipulación de robots en cuanto a su orientación al igual que para mover robots móviles.Al momento de estabilizar la señal del dispositivo se pueden obtener datos como los que se muestran a continuación:

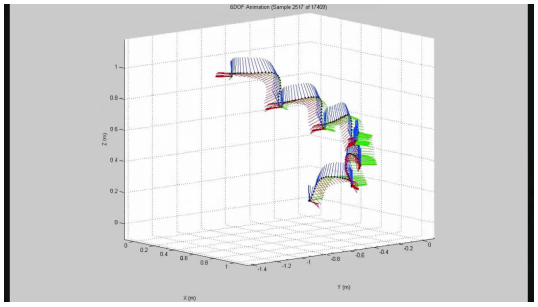

Figura 5: Red de cámaras de alta velocidad en cancha de tenis

6. Fundamentación

El método utilizado en este proyecto para poder detectar puntos en tres dimensiones es mediante el dispositivo de Microsoft Kinect en su versión 1,por lo tanto es preciso entender el funcionamiento de dicho dispositivo,el funcionamiento de sus componentes de forma individual se describe a continuación:

1. Cámara RGB- Con resolución de 1280x960 pixeles.
   Dicha cámara no tiene ningún propósito al momento de detectar el cuerpo humano, simplemente se utiliza para entretenimiento, al tomar fotos y vídeos durante un videojuego.

2. Emisor y receptor infrarrojo- Con la ayuda de estos sensores se crea un vector de puntos que contiene la información de la distancia entre el sensor y la persona en cada uno de los puntos que el sensor proyecta hacia el exterior así como también cuál es la posición en tiempo real de la persona que se encuentra enfrente del sensor Kinect y con esto obtener los puntos en tres dimensiones.

3. Motor de inclinación- Este servomotor sirve para cambiar la ventana de visión del sensor Kinect al cambiar su inclinación.

4. Adicionalmente el sensor Kinect cuanta con dos sensores mas los cuales son una cadena de micrófonos y un acelerómetro de 3 ejes el cuál sirve para detectar la inclinación del mismo Kinect.

La ubicación de los componentes dentro del Kinect se muestra en la Figura 6

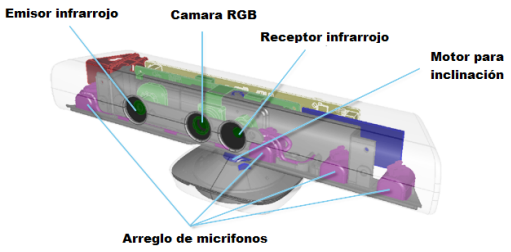

Figura 6: Partes del sensor Kinect V1

Mediante el emisor infrarrojo y el receptor infrarrojo, el sensor Kinect manda una serie de datos a la computadora que después de ser procesados a través de una serie de algoritmos se puede conocer la posición en tres dimensiones de puntos determinados del cuerpo humano en tiempo real, dichas coordenadas son obtenidas en metros considerando como el eje 'z'la profundidad del objeto, como eje 'y'la altura del objeto o desplazamiento vertical y como eje 'xél desplazamiento horizontal. A continuación se muestran los puntos que se pueden seguir con el Kinect V1:

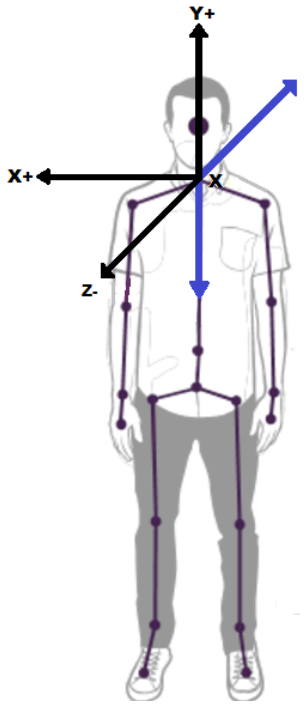

Figura 7: Cuerpo escaneado por Kinect V1

La Figura 7 muestra el sistema coordenado que maneja el sensor Kinect, posteriormente se explica cuál es la relación establecida entre el sistema coordenado del sensor Kinect y el sistema coordenado del robot a manipular así como otros aspectos importantes a considerar.

El sensor Kinect es capaz de detectar hasta 6 personas al mismo tiempo como se muestra en la Figura 8, sin embargo solamente puede estimar la posición de dos personal al mismo tiempo, pero por

motivos de seguridad en esta aplicación es estrictamente requerido que solo una persona se encuentre enfrente del Kinect ya que aunque solamente se estarán siguiendo los movimientos de una persona no se tiene control mediante programación para decidir o tratar de saber cuál es la persona que esta manipulando el efector final del robot.

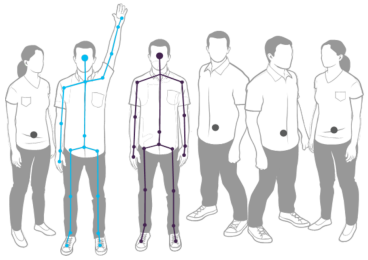

Figura 8: Número máximo de personas detectadas por el Kinect V1

Otro aspecto importante a considerar es la orientación de la persona con respecto al sensor Kinect ya que para obtener un cuerpo como el que se muestra en la Figura 7 es necesario estar mirando de frente al Kinect, de lo contrario el Kinect podría llegar a perder de vista a la persona o malinterpretar su posición, la orientación correcta se muestra en la Figura 9.

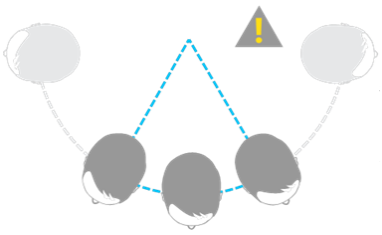

Figura 9: Orientación correcta para detección de posición

Así como es de vital importancia considerar la orientación también lo es considerar el rango de visión del Kinect, ya que este es limitado y es necesario trabajar dentro de dichos limites, el primero a considerar es la cercanía que se debe de tener con el Kinect la cuál es 1.2m a 3.5m de manera óptima y de 0.8m a 4m para poder reconocer el cuerpo, cabe mencionar que el Kinect V2 tiene otro modo de funcionamiento para poder detectar a la persona de modo

cercano, dicho rango trabaja en los limites de 0.4m a 3m de manera óptima y de 0.8m a 2.5m para poder reconocer el cuerpo.

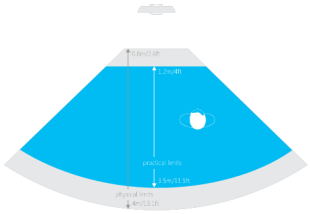

Figura 10: Distancia óptima en modo normal del Kinect

El último limite a considerar es el rango de visión de altura, el cual es el más amplio ya que se puede ajustar mediante el motor de inclinación, sin embargo de manera general si una persona se posiciona dentro la cercanía óptima ya antes mencionada no existirá problema alguno con la altura.

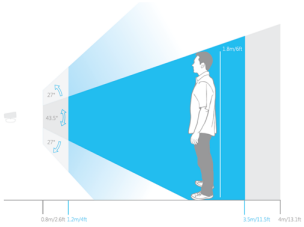

Figura 11: Altura óptima en modo normal del Kinect

El sensor Kinect fue diseñado para la consola de videojuegos Xbox, sin embargo al ser Microsoft el creador del Kinect también se desarrolló un kit de desarrollo de software(SDK por sus siglas en inglés) para poder programar aplicaciones en la plataforma de Visual Studio la cual es la plataforma o compilador que se utiliza para el desarrollo de este proyecto.

Par el desarrollo de una aplicación en C#, F#, C++ o algún otro paradigma de programación es necesario instalar el SDK o "Kit de Desarrollo de Software.[el] cuál dependerá según sea la versión del Kinect, para la V1 del Kinect la cuál se utiliza en este proyecto es requerido instalar la versión 1.8 y para la V2 es necesario instalar la versión 2.0 en adelante, es importante mencionar que con el SDK 2.0 no es posible realizar aplicaciones para la versión 1 del Kinect

ya que el sensor cambio su hardware.

Para poder tener un rendimiento óptimo al momento de realizar una aplicación con el Kinect V1 es necesario cumplir con los siguientes requerimientos:

1. Sistema operativo: Windows 7, Windows 8, Windows 8.1 o Windows 10
2. Procesador: 32-bit (x86) o 64-bit (x86), 2.66-GHz o mayor.
3. USB 2.0 o 3.0.
4. Memoria RAM mayor o igual a 2GB.

En cuanto a la versión de Visual Studio utilizada es la 2015 con programación en C#, cabe mencionar que con tan solo tener instalado un sistema operativo de Windows es posible descargar cualquier versión de Visual Studio de manera completamente gratuita.

El proyecto se puede dividir en dos partes, la obtención de las coordenadas en tres dimensiones y la manipulación del efector final del robot que consiste en asignarle una posición en 3 dimensiones teniendo como referencia las coordenadas obtenidas por el sensor Kinect, dicha tarea se puede realizar de dos maneras, la primera es mediante cinemática directa, que consiste en asignarle un valor a cada efector del robot ya sea prismático o de revoluta y mediante esos valores determinar la posición del efector final y la segunda opción es mediante cinemática inversa que consiste en darle un posición al efector final y mediante esta posición determinar el valor angular o lineal de cada uno de los efectores del robot o grados de libertad, de tal forma que si tenemos un robot de 5 grados de libertad la cinemática inversa nos arrojaría 5 valores diferentes.

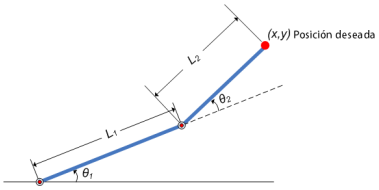

Figura 12: Robot serial de 2 grados de libertad (2GDL)

En la figura 12 se puede observar un robot de dos grados de libertad, mediante la cinemática directa obtendríamos la posición x, y al definir los valores de θ_1 y θ_2 y por el contrario con la cinemática inversa obtendríamos los valores de θ_1 y θ_2 al definir los valores de x y y.

Cada configuración distinta de un robot ya sea un robot serial, un robot paralelo,un robot móvil o cualquier otra configuración para un robot manipulador siempre tendrá una solución matemática diferente, sin embargo algo tienen en común todas las configuraciones distintas de robots manipuladores lo cual es que todos y cada uno de ellos puede ser manipulado al indicarle al robot cual debe ser la posición del efector final, por lo tanto se llegó a la conclusión de desarrollar la aplicación de una manera universal, es decir para manipular el efector final de cualquier robot sin importar su configuración.

El software de la aplicación desarrollada no se encuentra en la misma computadora que el sistema de control del robot a manipular,por eso es indispensable un protocolo de comunicación, y dado que la comunicación serial, más particularmente hablando del protocolo RS232 es muy sencillo de implementar y se encuentra en la mayoría de los dispositivos móviles programables, es este el protocolo que se utilizará para la comunicación entre el robot y la aplicación. En la comunicación serial existen dos configuraciones principales, el puerto con el qe se va trabajar y la velocidad de transmisión del puerto, con dichos parámetros definidos se puede establecer una comunicación entre cualquier par de puertos seriales.

7. Procedimientos

7.1. Estrategia de control

La parte principal del proyecto es plantear el método que se va a implementar para la manipulación del brazo robótico, en donde hay dos puntos importantes a considerar, la manipulación de la posición efector final y el control del efector final en caso de que este pueda ser controlado en apertura y cerradura. La estrategia planteada es la siguiente:

1. Mano derecha controla posición de efector final
2. Mano izquierda controla estado de efector final y detiene el programa..

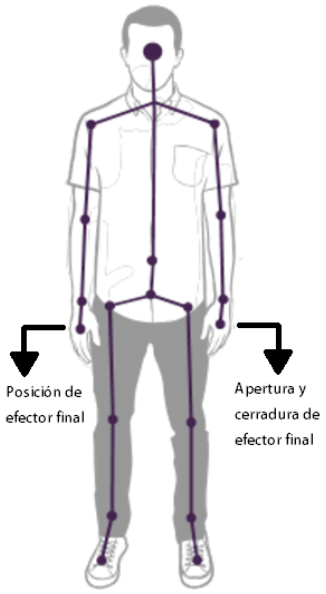

Figura 13: Estrategia de control

Como se puede observar en la figura 13 las dos manos son requeridas para lograr el control total del robot y como más adelante se explicará para definir el espacio de trabajo del Kinect.

7.2. Configuración inicial

Antes de obtener la coordenadas en tres dimensiones por parte del Kinect es necesario realizar ciertas configuraciones iniciales tales como la resolución o los cuadros por segundo a los cuales estará trabajando la aplicación,estas configuraciones nos permitirán adaptar las características del Kinect para obtener el mejor desempeño posible, dichas configuraciones se explican a continuación:

Código 1: Configuraciones iniciales

```
//Usar el primer sensor Kinect detectardo
kSensor = KinectSensor.KinectSensors[0];
//Inicializar el sensor Kinect
kSensor.Start();
//Habilitar la imagen de profundidad
kSensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
//Evento para realizar operaciones las coordenas del cuerpo estan listas
kSensor.SkeletonFrameReady += KSensor_SkeletonFrameReady;
//Habilitar la detección del cuerpo
kSensor.SkeletonStream.Enable();
//Habilitar la forma de rastreo (sentado o parado=default)
kSensor.SkeletonStream.TrackingMode = SkeletonTrackingMode.Default;
```

Adjunto al código 1 la palabra kSensor se define como un objeto para el sensor Kinect de la siguientes forma: "private KinectSensor kSensor;", este objeto nos permite tener acceso a diferentes funciones tales como encenderlo, apagarlo o cambiar su grado de inclinación como se muestra a continuación:

Código 2: Accseso a funciones del sensor Kinect

```
//Inicializar el sensor Kinect
kSensor.Start();
//Detener el sensor Kinect
kSensor.Stop();
//Obtener o cambiar el ángulo de elevación
kSensor.ElevationAngle();
//Saber si el sensor Kinect se encuantra encendido
kSensor.IsRunning
```

Es indispensable configurar el Kinect antes de empezar a utilizarlo ya que aunque algunas de las configuraciones se pueden omitir estas adquieren los valores predefinidos que pueden ser o no ser las configuraciones óptimas para la aplicación.

Con dichas configuraciones se puede empezar a recibir información por parte del sensor Kinect de cada una de las extremidades que el sensor puede rastrear,sin embargo no todas son de nuestro interés, por lo tanto las extremidades que utilizaremos son las siguientes:

Figura 14: Puntos de interés

7.3. Obtención de coordenadas en tres dimensiones

La parte central de la aplicación se basa en lo que se hará con las coordenadas en tres dimensiones de la articulaciones del cuerpo, como se mencionó anteriormente los puntos de control son la mano izquierda y la mano derecha, ya que con dichos puntos se controla el efector final así como también su posición respectivamente.

Mediante programación no se puede acceder a dichas coordenadas en cualquier momento es por eso que es necesario primero definir el evento que nos dice en que momento se puede hacer uso de dichas coordenadas, tal y como se mencionó en el código 1 primero es necesario declarar el evento para poder hacer uso de el y posteriormente codificar dentro de dicho evento los algoritmos necesarios para obtener las coordenadas en tres dimensiones de las articulaciones deseadas tal y como se muestra en el código 3.

Código 3: Uso de evento para obtención de coordenadas

```
kSensor.SkeletonFrameReady += KSensor_SkeletonFrameReady;

private void KSensor_SkeletonFrameReady(SkeletonFrameReadyEventArgs e)
{
  using (var frame = e.OpenSkeletonFrame())
  {
    if (frame != null)
    {
      var skeletons = new Skeleton[frame.SkeletonArrayLength];
      frame.CopySkeletonDataTo(skeletons);
      var TrackedSeleton = skeletons.FirstOrDefault();
      if (TrackedSeleton != null)
      {
        GetSkeleton(TrackedSeleton);
      }
    }
  }
}
```

En el código 3 se puede observar al principio la declaración del evento que permite acceder a la posición en tres dimensiones de las articulaciones del cuerpo, posteriormente se atiende dicho evento en donde se obtienen las coordenadas del primer cuerpo detectado ya que como se menciono anteriormente el Kinect es capaz de detectar a más de un cuerpo dentro de su campo de visión, y una vez que ya se detecta que el Kinect se encuentra siguiendo las posición de cuerpo ya se puede acceder a las coordenadas en tres dimensiones como se muestra en el código 4:

Código 4: Obtención de coordenadas

```
private void GetSkeleton(Skeleton skl)
{
  //Obtener puntos de las articulaciones del cuerpo
  SkeletonPoint RH = skl.Joints[JointType.HandRight].Position;
  SkeletonPoint LH = skl.Joints[JointType.HandLeft].Position;
  SkeletonPoint CS = skl.Joints[JointType.ShoulderCenter].Position;

  // Puntos de la mano derecha
  x1 = RH.X * 100;
  y1 = RH.Z * 100;
  z1 = RH.Y * 100;
  //Puntos de la mano izquierda
  x2 = LH.X * 100;
  y2 = LH.Z * 100;
  z2 = LH.Y * 100;
  //Puntos del hombro central
  x = CS.X * 100;
  y = CS.Z * 100;
  z = CS.Y * 100;
}
```

Como se muestra en el código 4 se obtienen coordenadas de tres puntos de interés, los cuales sn la mano izquierda como "LH", la mano derecha como RHz el centro de la clavícula como ÇSza que este último servirá como punto de referencia como se explica más

adelante. El formato de las coordenadas esta en metros, es por eso que cada uno de los puntos es multiplicado por 100, ya que por practicidad en la aplicación es mejor manejar las coordenadas en centímetros.

Como se puede apreciar en el código 4 la coordenada y es intercambiada con la coordenada z para poder obtener un sistema cartesiano convencional, de esta manera se realiza el siguiente cambio:

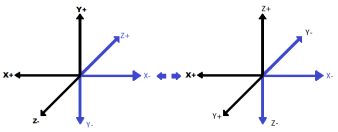

Figura 15: Cambio de sistema coordenado

7.4. Definición del espacio de trabajo del Kinect y del robot a manipular

Para determinar el espacio de trabajo del Kinect es necesario también definir un punto de referencia para poder obtener valores absolutos, esto con el propósito obtener los mismos valores en x,y y z sin importar a que distancia se encuentre la persona del Kinect, de esta manera el punto de referencia queda definido de la siguiente manera:

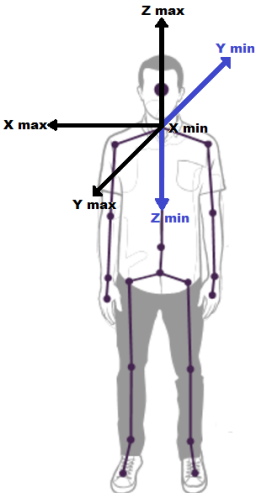

Figura 16: Sistema de referencia ajustado

Con este sistema de referencia los valores de x,y y z se obtienen de la siguiente forma:

Código 5: Sistema coordenado ajustado

```
xabs = x1 - x;
yabs = (y1 - y)*(-1);
zabs = z1 -z;
```

Con este método se pueden obtener valores de la posición de la mano derecha con respecto a un sistema coordenado, sin embargo aun se tienen valores que varían de acuerdo a la distancia a la que uno se encuentre del Kinect, es por esta razón que en necesario definir previamente cuales son los valores máximos y mínimos dentro de los cuales se va a estar trabajando, al tener los valores máximos es posible implementar un algoritmo que permita que el robot reciba coordenadas fuera de los limites preestablecidos, este paso se logra de la siguiente manera:

Código 6: Definición de valores máximos y mínimos

```
if (xabs > xmax) { xabs = xmax; }
if (yabs > ymax) { yabs = ymax; }
if (zabs > zmax) { zabs = zmax; }
if (xabs < xmin) { xabs = xmin; }
if (yabs < ymin) { yabs = ymin; }
if (zabs < zmin) { zabs = zmin; }
```

La definición del espacio de trabajo del Kinect consta de dos partes, la primera es definir los limites máximos y mínimos del espacio de trabajo del robot, para lograr esta tarea es necesario declarar variables que representen los valores que delimiten el espacio de trabajo del robot y la segunda parte es trasladar el espacio de trabajo del Kinect al espacio de trabajo del Robot para esto se usara el método de interpolación o triángulos semejantes de la siguiente manera:

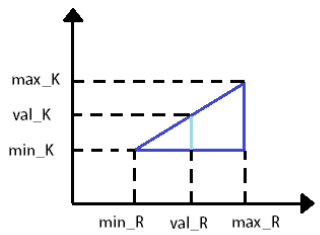

Figura 17: Método de interpolación

De la variables que se muestran en la Figura 17 la que nos interesa

calcular es "val_K", misma variable que representa el valor de la posición del robot en un eje dado, para encontrar dicho valor se usa la siguiente ecuación;

$$val_R = (max_R - min_R)(val_R - min_K)/(max_k - min_k) + min_R$$

(Interpolación)

En la ecuación de interpolación los subindicies Rz "Krepresentan los espacios de trabajo del Robot y del Kinect respectivamente.

La interpolación se aplica a cada uno de los ejes, es decir que en necesario interpolar el $ejex$, $ejey$ y $ejez$ en cada nueva iteración de la aplicación, para ello es necesario implementar el siguiente algoritmo:

Código 7: Interpolación de ejes x,y,z

```
//Interpolar eje x
Scale_x = (xabs - xmin)/(xmax - xmin);
X_robot = (Max_Space_X - Min_Space_X) * Scale_x + Min_Space_X;
//Interpolar eje y
Scale_y = (yabs - ymin) / (ymax - ymin);
Y_robot = (Max_Space_Y - Min_Space_Y) * Scale_y + Min_Space_Y;
//Interpolar eje z
Scale_z = (zabs - zmin) / (zmax - zmin);
Z_robot = (Max_Space_Z - Min_Space_Z) * Scale_z + Min_Space_Z;
```

7.5. Estabilización de señal del sensor Kinect

Es necesario determinar en que punto la señal del Kinect es estable,la señal que entrega el Kinect se encuentra con la siguiente exactitud :0,0000001m , es decir que la exactitud es de un micrómetro lo cual no significa que a un micrómetro la señal sea estable, y como estable se define el punto en el que mientras la mano se encuentra estática la señal también se encuentre estática punto que se puede determinar mediante las siguientes gráficas:

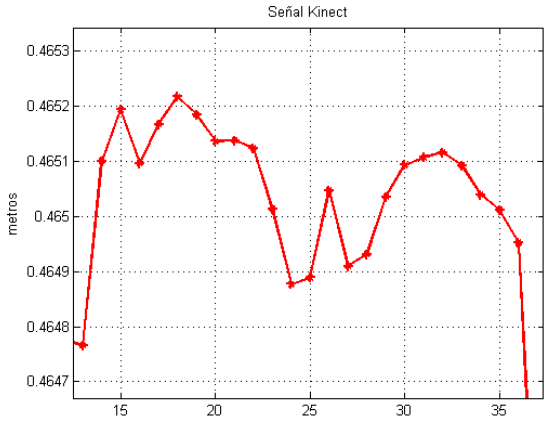

Figura 18: Señal Kinect eje x

Como se puede apreciar en la Figura 18 las décimas, centímetros y milímetros son estables, es por eso que las señal se debe redondear hasta tres decimales para evitar inestabilidad en la señal.A pesar de que la señal en milímetros entregada por el Kinect es estable existen situaciones en las que puede haber un cambio de valores repentino debido a factores externos tales como eso excesivo de memoria RAM o una mala ubicación enfrente del Kinect, tal y como se muestra en la siguiente gráfica:

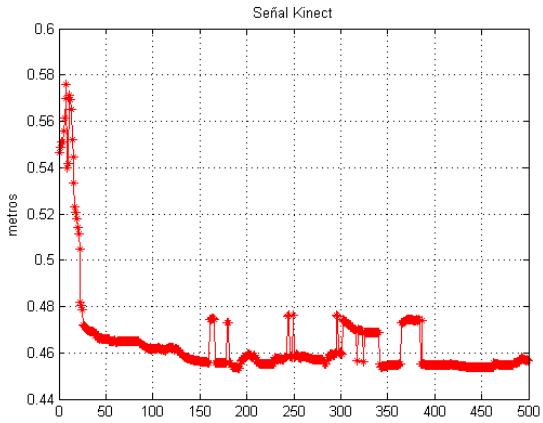

Figura 19: Señal Kinect eje x

Ya que existen cambios en la señal por factores externos es nece-

sario convertir la señal del Kinect en un punto de referencia y crear una nueva señal estable o que no tenga dichos cambios repentinos, para eso se crea una señal que siga a los valores de la señal otorgada por el Kinect, para crear dicha señal estable se requiere del siguiente algoritmo:

Código 8: Señal se seguimiento

```
if (X_fill<X_robot){if(countx1==speed){X_fill+=0.1;}else{countx1++;}}
if (X_fill>X_robot){if(countx2==speed){X_fill-=0.1;}else{countx1++;}}
if (Y_fill<Y_robot){if(county1==speed){Y_fill+=0.1;}else{county1++;}}
if (Y_fill>Y_robot){if(county2==speed){Y_fill-=0.1;}else{county2++;}}
if (Z_fill<Z_robot){if(countz1==speed){Z_fill+=0.1;}else{countz1++;}}
if (Z_fill>Z_robot){if(countz2==speed){Z_fill-=0.1;}else{countz2++;}}
```

En el código 8 la variable *speed* es una variable definida por el usuario que define la velocidad de seguimiento, misma que se puede configurar para aumentar o disminuir dicha velocidad.De esta manera tenemos una posición deseada otorgada por el sensor Kinect y una variable estabilizada que realiza el seguimiento a la variable otorgada por el Kinect como se muestra a continuación:

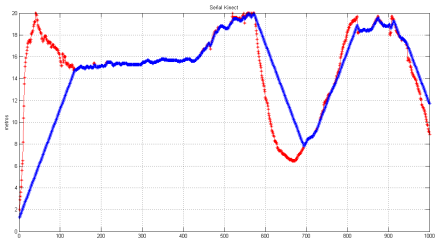

Figura 20: Variable de seguimiento en eje x

La variable de seguimiento nos permite evitar cambios inesperados o no deseados en la señal del Kinect.Como se muestra en la Figura 20 es necesario esperar a que la variable de seguimiento alcancé el valor de la señal del Kinect lo cual también se puede traducir como el robot alcanzando la posición deseada, en el segmento de la muestra N°800 a la muestra N°1000 se puede apreciar claramente como la variable de seguimiento que se muestra de color azul nos permite evitar los cambios inesperados en la señal tal y como se muestra con más detalle a continuación:

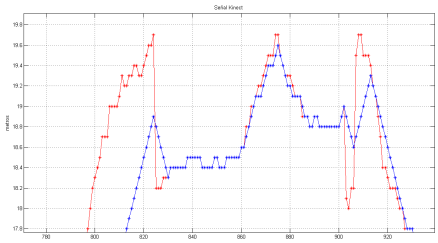

Figura 21: Corrección de cambios inesperados

Tanto la gráfica 20 como la gráfica 21 se fueron muestreadas con la mayor velocidad de seguimiento posible, que también se puede traducir como la velocidad máxima del procesador de la computadora. La señal del sensor Kinect en 3 dimensiones se puede observar de la siguiente manera:

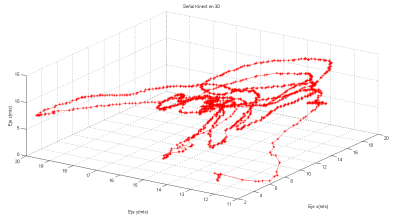

Figura 22: Señal 3D del sensor Kinect

Mientras que la señal de seguimiento se puede observar de la siguiente manera:

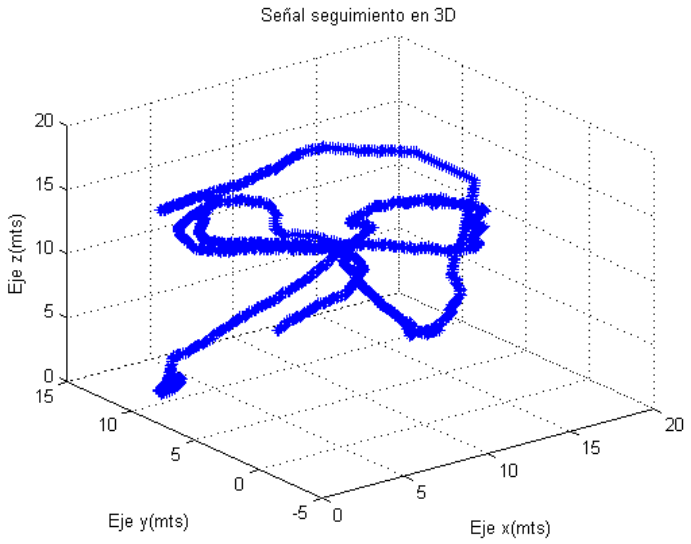

Figura 23: Señal 3D de la señal de seguimiento

A pesar de la las Figuras 22 y 23 muestran dos señales distintas la señal de seguimiento presenta más estabilidad mientras que la señal directa del Kinect se encuentra con pequeñas perturbaciones.

8. Resultados

La aplicación que se desarrolló en el entorno de Visual C# cuenta con una ventana principal o ventana de control en donde el usuario tiene acceso a configurar parámetros como el puesto serial y el espacio de trabajo del robot, dicha ventana se muestra a continuación:

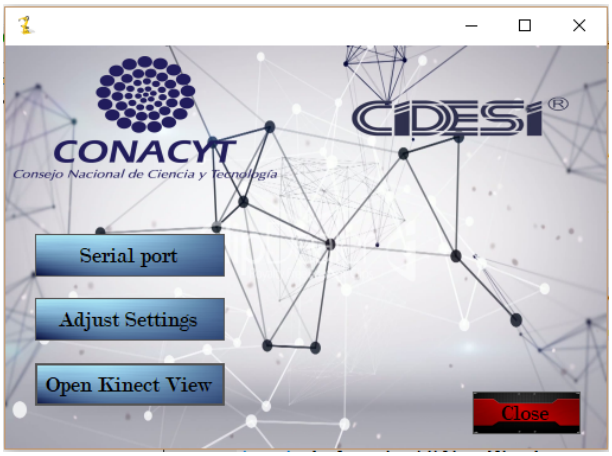

Figura 24: Ventana de control

En la ventana de control es necesario declarar el puesto serial así como también las configuraciones del espacio de trabajo del robot y la velocidad de seguimiento antes de abrir el Kinect, si dichas configuraciones no se definen antes de utilizar el Kinect la aplicación no funcionará al 100 % y por lo tanto no se mandarán las coordenadas por el puesto serial. La ventana de configuración del puerto serial es la siguiente:

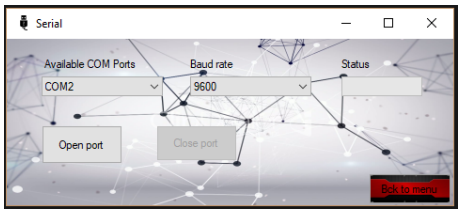

Figura 25: Ventana de configuración del puesto serial

Como se puede observar en la Figura 25 es necesario declarar la velocidad de transmisión así como también el puerto COM por

el cuál se llevará a cabo la comunicación, una vez que estos dos parámetros quedan definidos solo resta abrir el puerto serial mediante el botón .ºpen", y el puerto se puede cerrar mediante esta misma ventana o al cerrar la aplicación.

La siguiente configuración implica definir el espacio de trabajo del robot manipulador a utilizar en sus tres ejes, así como también definir la velocidad de las variables de seguimiento y una velocidad adicional que se utiliza para el control del efector final como se muestra en la Figura 26

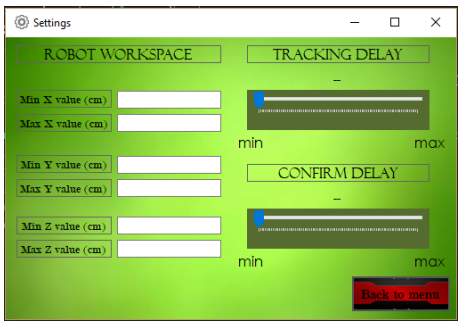

Figura 26: Ventana de configuraciones generales

En la ventana de configuraciones generales la aplicación puede detectar algunas incongruencias en la declaración del espacio de trabajo del robot, como definir el valor máximo de un eje menor que el valor mínimo o introducir caracteres inválidos.

Es importante mencionar que la forma correcta de cerrar la ventana es presionando el botón de "Back.ᵉⁿ lugar de cerrar la ventana, ya que si se cierra la ventana los datos anteriormente guardados se perderán. Una vez que dichos parámetros has sido definidos se puede empezar a trabajar con el Kinect al presionar el botón .ºpen Kinect View.ᵉⁿ la ventana de control en donde el primer paso de ejecución es definir los valores máximos y mínimos del espacio de trabajo del Kinect.

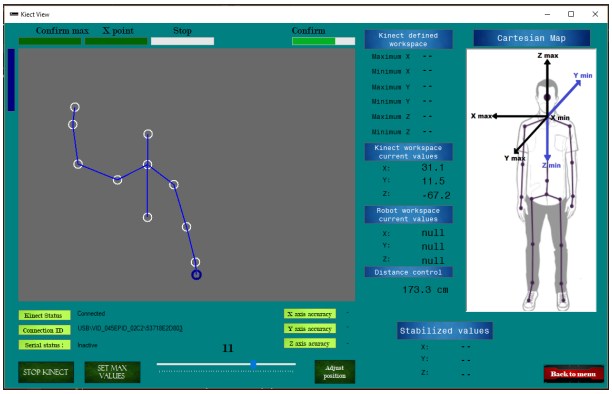

Figura 27: Ventana de trabajo

Una vez que la barra de progreso de confirmación se llena se le asignara el valor en el que se encuentre la mano derecha en la posición indicada de acuerdo al espacio de trabajo que se muestra en la parte derecha de la aplicación, una vez que todas las coordenadas hayan sido definidas y el espacio de trabajo del robot este definido los indicadores de la parte superior izquierda funcionaran para controlar el efector final como se muestra a continuación:

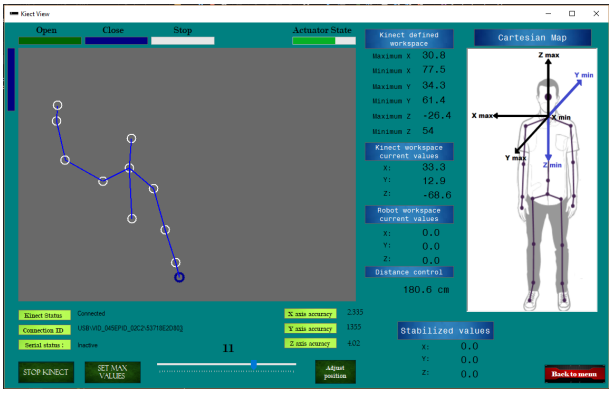

Figura 28: Ventana de trabajo con espacio de trabajo definido

Si todos los parámetros están definidos y el puerto serial se encuentra activo la aplicación se estará lista para mandar las variables correspondientes a los ejes x , y , z y los valores correspondientes al efector final, los cuales son un estado de 1 o 0 para indicar si se encuentra activado o desactivado y un valor de 0 a 100 que indica

el porcentaje de apertura del efector final, todos estos valores son concatenados en un cadena que tiene el siguiente formato:

0.0x0.0y0.0z0S0F

Figura 29: Formato de concatenación del puerto serial

En una iteración normal se podría apreciar la siguiente concatenación:

14.3x9.2y1.8z1S85F

Figura 30: Formato de concatenación del puerto serial en caso aleatorio

En el caso que se muestra en la Figura 30 se mandan los valores de las coordenadas x , y , z y el actuador se encuentra activado con un 85 % de apertura. Para poder tener acceso a los datos ya antes mencionado se requiere manda un valor de 1 a través del puerto serial e inmediatamente la aplicación mandará y los valores ya antes mencionados.

9. Conclusiones

Con la realización de dicho proyecto es posible concluir que la manipulación de objetos virtuales hace posible la conexión con robots manipuladores que requieren de realizar tareas poco accesibles para los humanos.

10. Recomendaciones

Se recomienda traspasar la interfaz realizada a la nueva versión del sensor Kinect (V2.0) para así obtener una mayor precisión y estabilidad de las coordenadas obtenidas.

11. Referencias bibliográficas

Referencias

[1] Leslie Lamport, *Robótica industrial prototipo y sistemas de visión artificial*, Editorial académica española 1ra edición, 2012.

[2] Cortés, F. R., *Robótica*, Alfaomega 1ra edición, 2013.

[3] Microsoft., *Microsoft Developer Network*, msdn.microsoft.com.

[4] OSTAR, *ROV training Center*, 2016, http://www.rovs.es.

12. Anexos

Código 9: Código para ventana de control

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace App1_Kinect
{
public partial class Acceso : Form
{
public  class FormProvider
{
public static Control wincontrol
{
get
{
if (mainMenu == null)
{
mainMenu = new Control();
}
return mainMenu;
}
}
private static Control mainMenu;
public static Settings winset
{
get
{
if (set == null)
{
set = new Settings();
}
return set;
}
}
private static Settings set;
public static Serial serial
{
get
{
if (winserial == null)
{
winserial = new Serial();
}
return winserial;
}
}
private static Serial winserial;
}
public Acceso()
{
InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
Application.Exit();
}
private void btnOpenK_Click(object sender, EventArgs e)
{
this.Hide();
try
{
FormProvider.wincontrol.Show();
}
catch { new Control().Show();}
}
private void Acceso_FormClosed(object sender, FormClosedEventArgs e)
{
Application.Exit();
}
private void btnSettings_Click(object sender, EventArgs e)
{
this.Hide();
try
{
FormProvider.winset.Show();
}
catch { new Settings().Show(); }
}
private void btnSerial_Click(object sender, EventArgs e)
```

```
{
this.Hide();
try
{
FormProvider.serial.Show();
}
catch { new Serial().Show(); }
}
}
}
```

Código 10: Código para ventana de configuración del puerto serial

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;


namespace App1_Kinect
{
public partial class Serial : Form
{
public class FormProvider
{
public static Acceso Acceso
{
get
{
if (winAcceso == null)
{
winAcceso = new Acceso();
}
return winAcceso;
}
}
private static Acceso winAcceso;
}
public Serial()
{
InitializeComponent();
GetAvailablePorts();
}
private void button2_Click(object sender, EventArgs e)
{
this.Hide();
FormProvider.Acceso.Show();
}
void GetAvailablePorts()
{
String[] ports = SerialPort.GetPortNames();
cbports.Items.AddRange(ports);
}
private void button5_Click(object sender, EventArgs e)
{
if (cbports.Text == "" || cbbaud.Text == "")
{
MessageBox.Show("Please select port settings");
}
else
{
try
{
serialPort1.PortName = cbports.Text;
serialPort1.BaudRate = Convert.ToInt16(cbbaud.Text);
serialPort1.Open();
Acceso.FormProvider.wincontrol.lblSerial.Text = "Active";
btnOpen.Enabled = false;
btnclose.Enabled = true;
pbStatus.Value = 100;

}
catch(Exception ex)
{
MessageBox.Show("The port is already open or name does not exist");
btnclose.Enabled = false;
btnOpen.Enabled = true;
pbStatus.Value = 0;
Acceso.FormProvider.wincontrol.lblSerial.Text = "Inctive";
serialPort1.Close();
pbStatus.Value = 0;
}
}
```

```
}
private void btnclose_Click(object sender, EventArgs e)
{
serialPort1.Write("C");
serialPort1.Close();
pbStatus.Value = 0;
btnclose.Enabled = false;
btnOpen.Enabled = true;
Acceso.FormProvider.wincontrol.lblSerial.Text = "Inactive";
}
private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
char prueba = new char();
prueba= (char)serialPort1.ReadChar();
}
private void Serial_FormClosing(object sender, FormClosingEventArgs e)
{
if(serialPort1.IsOpen){ serialPort1.Close(); }
Acceso.FormProvider.wincontrol.lblSerial.Text = "Inactive";
FormProvider.Acceso.Show();
}
}
}
```

Código 11: Código para ventana de configuraciones generales

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace App1_Kinect
{
public partial class Settings : Form
{
public static double Min_Space_X;
public static double Max_Space_X;
public static double Min_Space_Y;
public static double Max_Space_Y;
public static double Min_Space_Z;
public static double Max_Space_Z;
public static bool complete = false;
public static int speed = 0;
public static int ConfirmD = 0;

public Settings()
{
InitializeComponent();
}
public class FormProvider
{
public static Acceso acceso1
{
get
{
if (acceso == null)
{
acceso = new Acceso();
}
return acceso;
}
}
private static Acceso acceso;
}
private void btnMenu_Click(object sender, EventArgs e)
{
this.Hide();
FormProvider.acceso1.Show();
if (tbMinX.Text != null && tbMaxX.Text != null &&
tbMinY.Text != null && tbMaxY.Text != null &&
tbMinZ.Text != null && tbMaxZ.Text != null)
{
complete = true;
if (Control.flag == 5)
{
Acceso.FormProvider.wincontrol.lblXacu.Text = Convert.ToString((Control.xmax - Control.xmin) /
 (Max_Space_X - Min_Space_X));
Acceso.FormProvider.wincontrol.lblYacu.Text = Convert.ToString((Control.ymax - Control.ymin) /
 (Max_Space_Y - Min_Space_Y));
Acceso.FormProvider.wincontrol.lblZacu.Text = Convert.ToString((Control.zmax - Control.zmin) /
 (Max_Space_Z - Min_Space_Z));
}
}
else { complete = false; }
```

```
}
private void tbMinX_Leave(object sender, EventArgs e)
{
try
{
Min_Space_X= Convert.ToDouble(tbMinX.Text);
}
catch
{
MessageBox.Show("The input most be a number");
tbMinX.Text = null;
}

}
private void Settings_FormClosing(object sender, FormClosingEventArgs e)
{
FormProvider.acceso1.Show();
}
private void tbMaxX_Leave(object sender, EventArgs e)
{
try
{
Max_Space_X = Convert.ToDouble(tbMaxX.Text);
if (Convert.ToDouble(tbMinX.Text) >= Convert.ToDouble(tbMaxX.Text))
{
MessageBox.Show("Max value most be bigger than min value");
tbMaxX.Text = null;
tbMinX.Text = null;
}
}
catch
{
MessageBox.Show("The input most be a number");
tbMaxX.Text = null;
}
}
private void tbMinY_Leave(object sender, EventArgs e)
{
try
{
Min_Space_Y = Convert.ToDouble(tbMinY.Text);
}
catch
{
MessageBox.Show("The input most be a number");
tbMinY.Text = null;
}
}
private void tbMaxY_Leave(object sender, EventArgs e)
{

try
{
Max_Space_Y = Convert.ToDouble(tbMaxY.Text);
if (Convert.ToDouble(tbMinY.Text) >= Convert.ToDouble(tbMaxY.Text))
{
MessageBox.Show("Max value most be bigger than min value");
tbMaxY.Text = null;
tbMinY.Text = null;
}
}
catch
{
MessageBox.Show("The input most be a number");
tbMaxY.Text = null;
}
}
private void tbMinZ_Leave(object sender, EventArgs e)
{
try
{
Min_Space_Z = Convert.ToDouble(tbMinZ.Text);
}
catch
{
MessageBox.Show("The input most be a number");
tbMinZ.Text = null;
}
}
private void tbMaxZ_Leave(object sender, EventArgs e)
{
try
{
Max_Space_Z = Convert.ToDouble(tbMaxZ.Text);
if (Convert.ToDouble(tbMinZ.Text) >= Convert.ToDouble(tbMaxZ.Text))
{
MessageBox.Show("Max value most be bigger than min value");
tbMaxZ.Text = null;
tbMinZ.Text = null;
}
}
```

```
catch
{
MessageBox.Show("The input most be a number");
tbMaxZ.Text = null;
}
}
private void trackBar1_ValueChanged(object sender, EventArgs e)
{
lblSpeed.Text = tbDelay.Value.ToString();
speed = tbDelay.Value;
}
private void Settings_Click(object sender, EventArgs e)
{
tbDelay.Focus();
}
private void Settings_Load(object sender, EventArgs e)
{
tbDelay.Focus();
}

private void tbConfirm_ValueChanged(object sender, EventArgs e)
{
lblConfirm.Text = tbConfirm.Value.ToString();
ConfirmD = tbConfirm.Value;
}
}
}
```

Código 12: Código para ventana trabajo

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using System.Windows.Forms;
using Microsoft.Kinect;
using System.Media;

namespace App1_Kinect
{
public partial class Control : Form
{

//Global variables
Graphics draw;
private KinectSensor kSensor;
int xold_RH = 0; int yold_RH = 0; int xold_LH = 0; int yold_LH = 0;
int xold_HE = 0; int yold_HE = 0; int xold_CS = 0; int yold_CS = 0;
int xold_RS = 0; int yold_RS = 0; int xold_LS = 0; int yold_LS = 0;
int xold_RE = 0; int yold_RE = 0; int xold_LE = 0; int yold_LE = 0;
int xold_RW = 0; int yold_RW = 0; int xold_LW = 0; int yold_LW = 0;
int xold_SP = 0; int yold_SP = 0;

uint countx = 0; uint county = 0; uint countz = 0;
uint countOC = 0;
int Actuator = 0;
int function = 1;
int speed = 0;
public static int flag = 0;
int countmax = 0;
int stopcount = 0;
double xabs;
double yabs;
double zabs;
//Define variables for maximum minimum values for x,y and z
public static double xmax = 0.00;
public static double xmin = 0.00;
public static double ymax = 0.00;
public static double ymin = 0.00;
public static double zmax = 0.00;
public static double zmin = 0.00;
double X_fill = 0.0;
double Y_fill = 0.0;
double Z_fill = 0.0;
bool kinect_state = false;
byte flagA = 0;

public class FormProvider
{
public static Acceso acceso1
{
get
{
if (acceso == null)
```

```
{
acceso = new Acceso();
}
return acceso;
}
}
private static Acceso acceso;
}
public Control()
{
InitializeComponent();
}
private void bntStream_Click(object sender, EventArgs e)
{
if (btnStream.Text == "Start Kinect")
{
if (KinectSensor.KinectSensors.Count > 0)
{
this.btnStream.Text = "Stop Kinect";
kSensor = KinectSensor.KinectSensors[0];//Use the first kinect because more than one can be connected.
kSensor.Start();
this.lblConnectionID.Text = kSensor.DeviceConnectionId;
kSensor.SkeletonStream.Enable();
kSensor.SkeletonFrameReady += KSensor_SkeletonFrameReady;
KinectSensor.KinectSensors.StatusChanged += KinectSensors_StatusChanged;
lblStatus.Text = kSensor.Status.ToString();
start();
}
else
{
MessageBox.Show("Kinect not found");
}
}
else
{
if (kSensor != null && kSensor.IsRunning)
{
kSensor.Stop();
this.btnStream.Text = "Start Kinect";
this.pbSkeleton.Image = null;//clear the image once you stop the program
btnAdjust.Enabled = false;
tbAngle.Enabled = false;
kinect_state = true;
}
}
}
private void start()
{
btnAdjust.Enabled = true;
tbAngle.Enabled = true;
tbAngle.Maximum = kSensor.MaxElevationAngle;
tbAngle.Minimum = kSensor.MinElevationAngle;
lblAngle.Text = kSensor.ElevationAngle.ToString();
tbAngle.Value = kSensor.ElevationAngle;
kinect_state = true;
}
private void KSensor_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
{
using (var frame = e.OpenSkeletonFrame())
{
if (frame != null)
{
var skeletons = new Skeleton[frame.SkeletonArrayLength];
frame.CopySkeletonDataTo(skeletons);
var TrackedSeleton = skeletons.FirstOrDefault(s => s.TrackingState == SkeletonTrackingState.Tracked);
if (TrackedSeleton != null)
{
GetSkeleton(TrackedSeleton);
}
}
}
}
private void KinectSensors_StatusChanged(object sender, StatusChangedEventArgs e)
{
this.lblStatus.Text = kSensor.Status.ToString();
}
private void trackBar1_Scroll(object sender, EventArgs e)
{
lblAngle.Text = tbAngle.Value.ToString();
}
private void btnAdjust_Click(object sender, EventArgs e)
{
if (kSensor != null && kSensor.IsRunning)
{
kSensor.ElevationAngle = (int)tbAngle.Value;

}
else
{
MessageBox.Show("The Kinect most be connected in order to adjust sensor");
}
```

```
}
private void button1_Click(object sender, EventArgs e)
{
this.Hide();
FormProvider.acceso1.Show();
}
private void trackBar1_ValueChanged(object sender, EventArgs e)
{
lblAngle.Text = tbAngle.Value.ToString();
}
private void DrawEllipse(int x, int y, int xold, int yold)
{
try
{
if (xold != x || yold != y)
{
Pen Pen = new Pen(Color.WhiteSmoke, 2);
Pen oldPen = new Pen(Color.DimGray, 2);
draw.DrawEllipse(oldPen, xold - 8, yold - 8, 16, 16);
draw.DrawEllipse(Pen, x - 8, y - 8, 16, 16);
}

}
catch { }
}
private void DrawEllipseHand(int x, int y, int xold, int yold)
{
try
{
if (xold != x || yold != y)
{
Pen Pen = new Pen(Color.DarkBlue, 4);
Pen oldPen = new Pen(Color.DimGray, 4);
draw.DrawEllipse(oldPen, xold - 8, yold - 8, 16, 16);
draw.DrawEllipse(Pen, x - 8, y - 8, 16, 16);
}

}
catch { }
}
private void DrawLine(int x, int y, int xold, int yold, int x1, int y1, int xold1, int yold1)
{
try
{
if (xold != x || yold != y || x1 != xold1 || y1 != yold1)
{
//Graphics draw1 = pbSkeleton.CreateGraphics();
draw = pbSkeleton.CreateGraphics();
Pen Pen = new Pen(Color.Blue, 2);
Pen oldPen = new Pen(Color.DimGray, 2);
Point p1 = new Point(x, y);
Point p2 = new Point(x1, y1);
Point p1old = new Point(xold, yold);
Point p2old = new Point(xold1, yold1);
draw.DrawLine(oldPen, p1old, p2old);
draw.DrawLine(Pen, p1, p2);
}
}
catch { }
}
private void OpenClose(int x, int y)
{
int CD = Settings.ConfirmD;
if (x <= 120 && y <= 120)
{
flagA = 1;
if (pbOpen.BackColor != Color.DarkGreen) pbOpen.BackColor = Color.DarkGreen;
if (pbClose.BackColor != Color.Navy) pbClose.BackColor = Color.Navy;
if (Actuator < 100)
{
if (countOC >= CD)
{
Actuator++;
countOC = 0;
pbActuator.Value = Actuator;
}
else { countOC ++; }

}
}
else if (x >= 126 && x <= 246 && y <= 120)
{
flagA = 0;
if (pbClose.BackColor != Color.DarkGreen) pbClose.BackColor = Color.DarkGreen;
if (pbOpen.BackColor != Color.Navy) pbOpen.BackColor = Color.Navy;
if (Actuator > 0)
{
if (countOC >= CD)
{
Actuator--;
countOC = 0;
```

```
pbActuator.Value = Actuator;
}
else { countOC ++; }

}
}
else
{
if (pbClose.BackColor != Color.Navy) pbClose.BackColor = Color.Navy;
if (pbOpen.BackColor != Color.Navy) pbOpen.BackColor = Color.Navy;
}
}
private void Control_Load(object sender, EventArgs e)
{
lblAngle.Text = tbAngle.Value.ToString();
btnAdjust.Enabled = false;
tbAngle.Enabled = false;
if (Acceso.FormProvider.serial.serialPort1.IsOpen) { lblSerial.Text = "Active"; }
else { lblSerial.Text = "Inactive"; }
}
private void DrawSkeleton(ColorImagePoint RH_POINT, ColorImagePoint LH_POINT, ColorImagePoint
 HE_POINT, ColorImagePoint CS_POINT, ColorImagePoint RS_POINT, ColorImagePoint LS_POINT
 , ColorImagePoint RE_POINT, ColorImagePoint LE_POINT, ColorImagePoint RW_POINT, ColorImagePoint
  LW_POINT, ColorImagePoint SP_POINT,double distance)
{
if(distance>=155)
{
//In this section all pionts from spine to head are drawn as circle pionts//

DrawEllipseHand(RH_POINT.X, RH_POINT.Y, xold_RH, yold_RH);//Draw right hand
DrawEllipse(LH_POINT.X, LH_POINT.Y, xold_LH, yold_LH);     //Draw left hand
DrawEllipse(HE_POINT.X, HE_POINT.Y, xold_HE, yold_HE);     //Draw head
DrawEllipse(CS_POINT.X, CS_POINT.Y, xold_CS, yold_CS);     //Draw center shoulder
DrawEllipse(RS_POINT.X, RS_POINT.Y, xold_RS, yold_RS);     //Draw right shoulder
DrawEllipse(LS_POINT.X, LS_POINT.Y, xold_LS, yold_LS);     //Draw left shoulder
DrawEllipse(RE_POINT.X, RE_POINT.Y, xold_RE, yold_RE);     //Draw right elbow
DrawEllipse(LE_POINT.X, LE_POINT.Y, xold_LE, yold_LE);     //Draw left elbow
DrawEllipse(RW_POINT.X, RW_POINT.Y, xold_RW, yold_RW);     //Draw right wrist
DrawEllipse(LW_POINT.X, LW_POINT.Y, xold_LW, yold_LW);     //Draw left wrist
DrawEllipse(SP_POINT.X, SP_POINT.Y, xold_SP, yold_SP);     //Draw spine

//In this section all pionts from spine to head are drawn as line pionts//

//Draw lime fom right hand to right wrist
DrawLine(RH_POINT.X, RH_POINT.Y, xold_RH, yold_RH, RW_POINT.X, RW_POINT.Y, xold_RW, yold_RW);
//Draw line from right wrist to right elbow
DrawLine(RW_POINT.X, RW_POINT.Y, xold_RW, yold_RW, RE_POINT.X, RE_POINT.Y, xold_RE, yold_RE);
//Draw line from right elbow to right shoulder
DrawLine(RE_POINT.X, RE_POINT.Y, xold_RE, yold_RE, RS_POINT.X, RS_POINT.Y, xold_RS, yold_RS);
//Draw line from right shoulder to center shoulder
DrawLine(RS_POINT.X, RS_POINT.Y, xold_RS, yold_RS, CS_POINT.X, CS_POINT.Y, xold_CS, yold_CS);
//Draw line from left hand to left wrist
DrawLine(LH_POINT.X, LH_POINT.Y, xold_LH, yold_LH, LW_POINT.X, LW_POINT.Y, xold_LW, yold_LW);
//Draw line from left wrist to left elbow
DrawLine(LW_POINT.X, LW_POINT.Y, xold_LW, yold_LW, LE_POINT.X, LE_POINT.Y, xold_LE, yold_LE);
//Draw line from left elbow to left shoulder
DrawLine(LE_POINT.X, LE_POINT.Y, xold_LE, yold_LE, LS_POINT.X, LS_POINT.Y, xold_LS, yold_LS);
//Draw line from left shoulder to center shoulder
DrawLine(LS_POINT.X, LS_POINT.Y, xold_LS, yold_LS, CS_POINT.X, CS_POINT.Y, xold_CS, yold_CS);
//Draw line from center shoulder to head
DrawLine(CS_POINT.X, CS_POINT.Y, xold_CS, yold_CS, HE_POINT.X, HE_POINT.Y, xold_HE, yold_HE);
//Draw line from center shoulder to spine
DrawLine(CS_POINT.X, CS_POINT.Y, xold_CS, yold_CS, SP_POINT.X, SP_POINT.Y, xold_SP, yold_SP);

//Update old positions
xold_RH = RH_POINT.X;
yold_RH = RH_POINT.Y;
xold_LH = LH_POINT.X;
yold_LH = LH_POINT.Y;
xold_HE = HE_POINT.X;
yold_HE = HE_POINT.Y;
xold_CS = CS_POINT.X;
yold_CS = CS_POINT.Y;
xold_RS = RS_POINT.X;
yold_RS = RS_POINT.Y;
xold_LS = LS_POINT.X;
yold_LS = LS_POINT.Y;
xold_RE = RE_POINT.X;
yold_RE = RE_POINT.Y;
xold_LE = LE_POINT.X;
yold_LE = LE_POINT.Y;
xold_RW = RW_POINT.X;
yold_RW = RW_POINT.Y;
xold_LW = LW_POINT.X;
yold_LW = LW_POINT.Y;
xold_SP = SP_POINT.X;
yold_SP = SP_POINT.Y;
}

}
private bool CheckConfig(int x, int y)
```

```
{
int CD = Settings.ConfirmD;
if (x <= 246 && y <= 120)
{
if (pbOpen.BackColor != Color.DarkGreen) pbOpen.BackColor = Color.DarkGreen;
if (pbClose.BackColor != Color.DarkGreen) pbClose.BackColor = Color.DarkGreen;
if (lblActuator.Text != "Confirm") lblActuator.Text = "Confirm";
if (countmax == 100)
{
countmax = 0;
return true;
}
else
{
if(countOC >= CD)
{
countOC = 0;
countmax++;
pbActuator.Value = countmax;
return false;
}
else
{
countOC++;
return false;
}
}
}
else
{
if (pbClose.BackColor != Color.Navy) pbClose.BackColor = Color.Navy;
if (pbOpen.BackColor != Color.Navy) pbOpen.BackColor = Color.Navy;
countmax = 0;
pbActuator.Value = 0;
return false;
}
}
private void SetConfig(int x, int y, double xabs, double yabs, double zabs)
{
if (flag == 0)
{
lblOpen.Text = "Confirm max";
lblClose.Text = "X point";
if (CheckConfig(x, y))
{
SystemSounds.Beep.Play();
xmax = Math.Round(Convert.ToDouble(lblxabs.Text),1);
lblmaxX.Text = xmax.ToString();
flag = 1;
}
}
else if (flag == 1)
{
lblOpen.Text = "Confirm min";
lblClose.Text = "X point";
if (CheckConfig(x, y))
{
SystemSounds.Beep.Play();
xmin = Math.Round(Convert.ToDouble(lblxabs.Text),1);
lblminX.Text = xmin.ToString();
flag = 2;
}
}
else if (flag == 2)
{
lblOpen.Text = "Confirm max";
lblClose.Text = "Y point";
if (CheckConfig(x, y))
{
SystemSounds.Beep.Play();
ymax = Math.Round(Convert.ToDouble(lblyabs.Text),1);
lblmaxY.Text = ymax.ToString();
flag = 3;
}
}
else if (flag == 3)
{
lblOpen.Text = "Confirm min";
lblClose.Text = "Y point";
if (CheckConfig(x, y))
{
SystemSounds.Beep.Play();
ymin = Math.Round(Convert.ToDouble(lblyabs.Text),1);
lblminY.Text = ymin.ToString();
flag = 4;
}
}
else if (flag == 4)
{
lblOpen.Text = "Confirm max";
lblClose.Text = "Z point";
```

```
if (CheckConfig(x, y))
{
SystemSounds.Beep.Play();
zmax = Math.Round(Convert.ToDouble(lblzabs.Text),1);
lblmaxZ.Text = zmax.ToString();
flag = 5;
}
}
else if (flag == 5)
{
lblOpen.Text = "Confirm min";
lblClose.Text = "Z point";
if (CheckConfig(x, y))
{
SystemSounds.Beep.Play();
zmin = Math.Round(Convert.ToDouble(lblzabs.Text),1);
lblminZ.Text = zmin.ToString();
lblOpen.Text = "Open";
lblClose.Text = "Close";
lblActuator.Text = "Actuator State";
pbActuator.Value = 0;
function = 0;
if (Settings.complete == true)
{
lblXacu.Text = Convert.ToString(Math.Abs((xmax - xmin) / (Settings.Max_Space_X -
 Settings.Min_Space_X)));
lblYacu.Text = Convert.ToString(Math.Abs((ymax - ymin) / (Settings.Max_Space_Y -
 Settings.Min_Space_Y)));
lblZacu.Text = Convert.ToString(Math.Abs((zmax - zmin) / (Settings.Max_Space_Z -
 Settings.Min_Space_Z)));
}
}
}
}
private void GetCords(SkeletonPoint RH, SkeletonPoint CS)
{
double x1; double y1; double z1;
double x2; double y2; double z2;
double Scale_x ; double Scale_y ; double Scale_z ;
double X_robot ; double Y_robot ; double Z_robot ;
// Right hand coordenates
x1 = RH.X * 100;
y1 = RH.Z * 100;
z1 = RH.Y * 100;
// Center Shoulder coordenates
x2 = CS.X * 100;
y2 = CS.Z * 100;
z2 = CS.Y * 100;
//Calculate the diference between shoulder an right hand
xabs = x1 - x2;
yabs = (y1 - y2)*(-1);
zabs = z1 -z2;

lblxabs.Text = String.Format("{0:0.0}", xabs);
lblyabs.Text = String.Format("{0:0.0}", yabs);
lblzabs.Text = String.Format("{0:0.0}", zabs);

if (y2<155 && lblDistance.ForeColor != Color.Red)
{
lblDistance.ForeColor = Color.Red;
pbSkeleton.Image = null;
}
else if(y2 >= 155 && lblDistance.ForeColor != Color.Black) { lblDistance.ForeColor = Color.Black; }

lblDistance.Text = String.Format("{0:0.0 cm}", y2);
if (Settings.complete == true && flag == 5 && y2 >= 155)
{
speed = Settings.speed;
if (xabs > xmax) { xabs = xmax; }
if (yabs > ymax) { yabs = ymax; }
if (zabs > zmax) { zabs = zmax; }
if (xabs < xmin) { xabs = xmin; }
if (yabs < ymin) { yabs = ymin; }
if (zabs < zmin) { zabs = zmin; }

//Convert from kinect x workspace to x robot workspace
Scale_x = (xabs - xmin) / (xmax - xmin);
X_robot = (Settings.Max_Space_X - Settings.Min_Space_X) * Scale_x + Settings.Min_Space_X;
//Convert from kinect y workspace to y robot workspace
Scale_y = (yabs - ymin) / (ymax - ymin);
Y_robot = (Settings.Max_Space_Y - Settings.Min_Space_Y) * Scale_y + Settings.Min_Space_Y;
//Convert from kinect z workspace to z robot workspace
Scale_z = (zabs - zmin) / (zmax - zmin);
Z_robot = (Settings.Max_Space_Z - Settings.Min_Space_Z) * Scale_z + Settings.Min_Space_Z;

lblxRobot.Text = String.Format("{0:0.0}", X_robot);
lblyRobot.Text = String.Format("{0:0.0}", Y_robot);
lblzRobot.Text = String.Format("{0:0.0}", Z_robot);

X_robot = Math.Round(X_robot, 1);
Y_robot = Math.Round(Y_robot, 1);
```

```
Z_robot = Math.Round(Z_robot, 1);

if (X_fill < X_robot) {if (countx >= speed) { X_fill += 0.1; countx = 0; } else { countx++; }}
if (X_fill > X_robot) {if (countx >= speed) { X_fill -= 0.1; countx = 0; } else { countx++; }}
if (Y_fill < Y_robot) {if (county >= speed) { Y_fill += 0.1; county = 0; } else { county++; }}
if (Y_fill > Y_robot) {if (county >= speed) { Y_fill -= 0.1; county = 0; } else { county++; }}
if (Z_fill < Z_robot) {if (countz >= speed) { Z_fill += 0.1; countz = 0; } else { countz++; }}
if (Z_fill > Z_robot) {if (countz >= speed) { Z_fill -= 0.1; countz = 0; } else { countz++; }}

lblXsta.Text =String.Format("{0:0.0}",X_fill);
lblYsta.Text =String.Format("{0:0.0}",Y_fill);
lblZsta.Text =String.Format("{0:0.0}",Z_fill);
if (Acceso.FormProvider.serial.serialPort1.IsOpen)
{
Acceso.FormProvider.serial.serialPort1.Write(lblXsta.Text + "x"
+ lblYsta.Text + "y"
+ lblZsta.Text + "z"
+ flagA + "S"
+ Actuator + "F");
}
}
else
{
if (lblxRobot.Text != "null")
{
lblxRobot.Text = "null";
lblyRobot.Text = "null";
lblzRobot.Text = "null";
}
}
}
private void StopKinect(int x, int y)
{
if (x >= 252 && x <= 372 && y <= 120)
{
if (stopcount < 100)
{
stopcount++;
pbStop.Value = stopcount;
}
else
{
kSensor.Stop();
pbSkeleton.Image = null;
btnStream.Text = "Start Kinect";
pbStop.Value = 0;
stopcount = 0;
SystemSounds.Beep.Play();
}
}
else
{
pbStop.Value = 0;
stopcount = 0;
}
}
private void GetSkeleton(Skeleton skl)
{
//Get skeleton points
SkeletonPoint RH = skl.Joints[JointType.HandRight].Position;
SkeletonPoint LH = skl.Joints[JointType.HandLeft].Position;
SkeletonPoint HE = skl.Joints[JointType.Head].Position;
SkeletonPoint CS = skl.Joints[JointType.ShoulderCenter].Position;
SkeletonPoint RS = skl.Joints[JointType.ShoulderRight].Position;
SkeletonPoint LS = skl.Joints[JointType.ShoulderLeft].Position;
SkeletonPoint RE = skl.Joints[JointType.ElbowRight].Position;
SkeletonPoint LE = skl.Joints[JointType.ElbowLeft].Position;
SkeletonPoint RW = skl.Joints[JointType.WristRight].Position;
SkeletonPoint LW = skl.Joints[JointType.WristLeft].Position;
SkeletonPoint SP = skl.Joints[JointType.Spine].Position;

//Convert skeleton points to specifc screen scale
var CM = new CoordinateMapper(kSensor);
var RH_POINT = CM.MapSkeletonPointToColorPoint(RH, ColorImageFormat.RgbResolution640x480Fps30);
var LH_POINT = CM.MapSkeletonPointToColorPoint(LH, ColorImageFormat.RgbResolution640x480Fps30);
var HE_POINT = CM.MapSkeletonPointToColorPoint(HE, ColorImageFormat.RgbResolution640x480Fps30);
var CS_POINT = CM.MapSkeletonPointToColorPoint(CS, ColorImageFormat.RgbResolution640x480Fps30);
var RS_POINT = CM.MapSkeletonPointToColorPoint(RS, ColorImageFormat.RgbResolution640x480Fps30);
var LS_POINT = CM.MapSkeletonPointToColorPoint(LS, ColorImageFormat.RgbResolution640x480Fps30);
var RE_POINT = CM.MapSkeletonPointToColorPoint(RE, ColorImageFormat.RgbResolution640x480Fps30);
var LE_POINT = CM.MapSkeletonPointToColorPoint(LE, ColorImageFormat.RgbResolution640x480Fps30);
var RW_POINT = CM.MapSkeletonPointToColorPoint(RW, ColorImageFormat.RgbResolution640x480Fps30);
var LW_POINT = CM.MapSkeletonPointToColorPoint(LW, ColorImageFormat.RgbResolution640x480Fps30);
var SP_POINT = CM.MapSkeletonPointToColorPoint(SP, ColorImageFormat.RgbResolution640x480Fps30);

if (function == 0)
{
GetCords(RH, CS);
OpenClose(LH_POINT.X, LH_POINT.Y);
StopKinect(LH_POINT.X, LH_POINT.Y);
}
```

```
else if (function == 1)
{
GetCords(RH, CS);
SetConfig(LH_POINT.X, LH_POINT.Y, xabs, yabs, zabs);
StopKinect(LH_POINT.X, LH_POINT.Y);
}
//Function to draw skeleton based on scaled pionts
DrawSkeleton(RH_POINT, LH_POINT, HE_POINT, CS_POINT,
RS_POINT, LS_POINT, RE_POINT, LE_POINT, RW_POINT,
LW_POINT, SP_POINT, CS.Z * 100);
}
private void btnSet_Click(object sender, EventArgs e)
{
function = 1;
flag = 0;
if (lblmaxX.Text != "null")
{
lblmaxX.Text = "null";
lblminX.Text = "null";
lblmaxY.Text = "null";
lblminY.Text = "null";
lblmaxZ.Text = "null";
lblminZ.Text = "null";
}
if (kinect_state == false)
{
MessageBox.Show("The kinect is not open");
}
}
private void Control_FormClosing(object sender, FormClosingEventArgs e)
{
this.Hide();
btnStream.Text = "Start Kinect";
FormProvider.acceso1.Show();
}
}
}
```