Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DETECCIÓN Y RASTREO DEL MOVIMIENTO DE LAS ARTICULACIONES ÓSEAS CON EL KINECT DE LA CONSOLA XBOX360 GESTIONADO CON PROCESSING 2 EN LINUX UBUNTU 14.04 LTS

Jorge Estuardo Toledo Morales
Asesorado por el Ing. Byron Odilio Arrivillaga Méndez

Guatemala, abril de 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DETECCIÓN Y RASTREO DEL MOVIMIENTO DE LAS ARTICULACIONES ÓSEAS CON EL KINECT DE LA CONSOLA XBOX360 GESTIONADO CON PROCESSING 2 EN LINUX UBUNTU 14.04 LTS

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JORGE ESTUARDO TOLEDO MORALES
ASESORADO POR EL ING. BYRON ODILIO ARRIVILLAGA MÉNDEZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, ABRIL DE 2016

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Raúl Eduardo Ticún Córdova
VOCAL V	Br. Henry Fernando Duarte García
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Carlos Eduardo Guzmán Salazar
EXAMINADOR	Ing. Julio Rolando Barrios Archila
EXAMINADORA	Inga. María Magdalena Puente Romero
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DETECCIÓN Y RASTREO DEL MOVIMIENTO DE LAS ARTICULACIONES ÓSEAS CON EL KINECT DE LA CONSOLA XBOX360 GESTIONADO CON PROCESSING 2 EN LINUX UBUNTU 14.04 LTS

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 8 de agosto de 2014.

Jorge Estuardo Toledo Morales

Guatemala 4 de febrero de 2016

Ingeniero Carlos Eduardo Guzmán Salazar
Coordinador del área de Electrónica
Escuela de Mecánica Eléctrica
Facultad de Ingeniería
USAC

Estimado Ingeniero

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante de la carrera de ingeniería electrónica JORGE ESTUARDO TOLEDO MORALES con carnet 200512142, titulado: **"Detección y rastreo del movimiento de las articulaciones óseas con el Kinect de la consola XBOX360 gestionado con Processing 2 en Linux Ubuntu 14.04 LTS"**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo según el protocolo, por lo cual como asesor apruebo su contenido.

Sin otro particular me despido, atentamente.

Ing. Byron Arrivillaga Méndez
Col. 5217

Ing. Byron Odilio Arrivillaga Méndez
Col: 5217

Byron Odilio Arrivillaga Méndez
Ing. Electrónico
Colegiado no.5217
Asesor

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 12 . 2016.
Guatemala, 15 de FEBRER0 2016.

Señor Director
Ing. Francisco Javier González López
Director Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: DETECCIÓN Y RASTREO DEL MOVIMIENTO DE LAS ARTICULACIONES ÓSEAS CON EL KINECT DE LA CONSOLA XBOX360 GESTIONADO CON PROCESSING 2 EN LINUX UBUNTU 14.04 LTS, del estudiante Jorge Estuardo Toledo Morales, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

Ing. Carlos Eduardo Guzmán Salazar
Coordinador Área Electrónica

sro

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 12. 2016.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto bueno del Coordinador de Área, al trabajo de Graduación del estudiante; JORGE ESTUARDO TOLEDO MORALES Titulado: DETECCIÓN Y RASTREO DEL MOVIMIENTO DE LAS ARTICULACIONES ÓSEAS CON EL KINECT DE LA CONSOLA XBOX360 GESTIONADO CON PROCESSING 2 EN LINUX UBUNTU 14.04 LTS, procede a la autorización del mismo.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

Ing. Francisco Javier González López

GUATEMALA, 3 DE MARZO 2016.

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.D.195-2016

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica al trabajo de graduación titulado: **DETECCIÓN Y RASTREO DEL MOVIMIENTO DE LAS ARTICULACIONES ÓSEAS CON EL KINECT DE LA CONSOLA XBOX360 GESTIONADO CON PROCESSING 2 EN LINUX UBUNTU 14.04 LTS**, presentado por el estudiante universitario: **Jorge Estuardo Toledo Morales**, y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
Decano

DECANO
FACULTAD DE INGENIERÍA

Guatemala, abril de 2016

/cc

ACTO QUE DEDICO A:

Dios	Padre omnipotente y misericordioso de quien proviene toda sabiduría.
María Auxiliadora	Intercesora amorosa y fiel ante los ojos de su hijo Jesucristo y nuestro Padre Dios.
Mi madre	Norma Leticia Toledo Morales, por su apoyo y consejos.
Mis amigos	Por su solidaridad y apoyo.

.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por su labor de formación académica superior, por medio de la cual somos profesionales al servicio de la patria.
Facultad de Ingeniería	Por los conocimientos prodigados a lo largo de la carrera.
Obra Salesiana de Don Bosco	Por brindarme la educación media y su importante aporte pedagógico a Guatemala en la formación de jóvenes.
Laboratorio de Electrónica	Por la oportunidad de adquirir experiencia llevando a la práctica los conocimientos teóricos aprendidos.
Ing. Byron Arrivillaga	Por asesorarme y compartir conmigo su conocimientos en diversas áreas académicas y profesionales.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

LISTA DE SÍMBOLOS

Símbolo	Significado
cm	Centímetro
IR	Luz infrarroja
m	Metro
mm	Milímetro
ms	Milisegundo
ft	Pies
in	Pulgada
s	Segundo

GLOSARIO

Adafruit Empresa con sede en Nueva York que vende kits para proyectos de hardware de código abierto.

Bluetooth Especificación industrial para redes inalámbricas de área personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz.

Callback function Devolución de llamada o retrollamada (en inglés: *callback*) es una función "A" que se usa como argumento de otra función "B". Cuando se llama a "B", esta ejecuta "A".

Canonical Ltd. Empresa de software de ordenadores con base en el Reino Unido fundada y financiada por el empresario sudafricano Mark Shuttleworth para la venta de soporte comercial y servicios relacionados con Ubuntu y otros proyectos afines.

Centro de masa Promedio de las masas, factorizadas por sus distancias a un punto de referencia. En un plano, es como el punto de equilibrio o de pivote de un balancín con respecto de los pares producidos.

Compilador Programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación. Usualmente el segundo lenguaje es lenguaje de máquina.

Dan Shiffman Profesor en el Programa de Telecomunicaciones Interactivas de la NYU, fue parte del proyecto OpenKinect para crear una librería para trabajar con el Kinect en Processing.

GLCD Pantalla plana formada por una matriz de píxeles monocromos colocados delante de una fuente de luz o reflectora (acrónimo del inglés *Graphic liquid crystal display*).

IR Tipo de radiación electromagnética y térmica, de mayor longitud de onda que la luz visible, pero menor que la de las microondas (acrónimo del inglés R*adiant energy*).

Josh Blake Director técnico del Grupo de Tecnología Avanzada InfoStrat y el fundador de la comunidad OpenKinect que desarrolla y mantiene el código abierto libfreenect controladores para el Kinect. Él es un MVP de Microsoft Surface y es el autor de Interfaces de Usuario Naturales.

Librería	Conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.
Linux Ubuntu	Sistema operativo basado en GNU/Linux y que se distribuye como software libre.
NITE	Módulo externo que no está disponible bajo una licencia de código abierto. Se trata de un producto comercial que pertenece a PrimeSense (acrónimo del inglés *Natural interaction*).
Oliver Kreylos	Científico de la computación en la UC Davis. Su trabajo se centra en el uso de la realidad virtual para mejorar las interfaces para la visualización de datos en la investigación científica.
OpenGL	Especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.
Paquetes de software	Colección de archivos de código fuente o binarios con un conjunto de archivos de instrucciones que especifican qué hacer con cada uno de ellos.
Periférico de entrada	Dispositivo utilizado para proporcionar datos y señales de control a la unidad central de procesamiento de una computadora.

Píxel	La menor unidad homogénea en color que forma parte de una imagen digital.
Processing	Cuaderno de bocetos de software flexible y un lenguaje para aprender a codificar en el contexto de las artes visuales.
Puerto	Interfaz a través de la cual se pueden enviar y recibir los diferentes tipos de datos.
Puerto serie	Interfaz de comunicaciones de datos digitales, frecuentemente utilizado por computadoras y periféricos, donde la información es transmitida bit a bit, enviando un solo bit a la vez.
RFCOMM	Conjunto simple de protocolos de transporte, construido sobre el protocolo L2CAP y que proporciona sesenta conexiones simultáneas para dispositivos bluetooth emulando puertos serie RS-232 (término inglés *Radio frequency communication*).
USB	Bus estándar industrial que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos (término en inglés: Universal Serial Bus).

RESUMEN

El presente trabajo de graduación estudia la obtención de datos de la cámara de profundidad y de color del Kinect, con el fin de realizar aplicaciones interactivas para el usuario. Para ello se investigaron temas tales como: conceptualización y marco histórico del Kinect, los elementos que lo integran, la instalación del sistema operativo, las librerías, drivers y el software requerido para la captura de datos con el Kinect.

Se detectan y rastrean los movimientos del esqueleto virtual dentro de las limitaciones del Kinect, utilizando la capacidad de OpenNI para procesar la información de la imagen de profundidad.

OBJETIVOS

General

Trabajar con software libre para la obtención de datos del Kinect del XBOX360 en el sistema operativo Linux Ubuntu 14.04LTS.

Específicos

1. Implementar los *drivers* y librerías necesarias para controlar el Kinect del XBOX360 en el sistema operativo Linux Ubuntu 14.04 LTS.

2. Gestionar el software de programación para capturar los datos del Kinect del XBOX360 en el sistema operativo Linux Ubuntu 14.04LTS.

3. Detectar y rastrear los movimientos de las articulaciones del cuerpo humano dentro de las posibilidades y limitaciones del Kinect del XBOX360.

INTRODUCCIÓN

El trabajo de graduación tiene como objetivo implementar los *drivers*, librerías, gestionar el software de programación necesario para detectar y rastrear las articulaciones del cuerpo humano dentro de las posibilidades y limitaciones del Kinect en Linux Ubuntu.

En el primer capítulo se explica: qué es el Kinect; sus antecedentes históricos, las generalidades de su funcionamiento y una breve descripción de los elementos que lo conforman.

El segundo capítulo trata sobre la instalación del sistema operativo Linux Ubuntu 14.04, las librerías, los *drivers* y el software requerido para llevar a cabo la programación necesaria y adquirir los datos del Kinect del XBOX360.

En el tercer capítulo se utilizó la cámara de profundidad del Kinect, que captura, por medio de una imagen, la distancia de los objetos y personas delante de él. La imagen capturada corresponde a una imagen de profundidad y cada píxel de esta registra la distancia del objeto con respecto al Kinect.

El cuarto capítulo se inicia con una orientación en 3D, posiciones de ejes, traslaciones, rotaciones en 3D. Asimismo, la búsqueda y maneras de convertir píxeles bidimensionales de la cámara de profundidad en una nube de puntos en el espacio tridimensional.

El capítulo final es concluyente y sumamente importante, ya que demuestra cómo se detecta y rastrea el movimiento del esqueleto virtual dentro

de las limitaciones del Kinect y utilizando la capacidad de OpenNI para procesar la imagen de profundidad. Permite obtener la posición, reconocer a las personas y saber dónde se ubican en el espacio. Puede accederse a la ubicación de cada parte del cuerpo de un usuario en 3D y obtener la posición exacta de las manos, la cabeza, los codos, los pies, entre otros; como también de cada una de las articulaciones (uniones) visibles del usuario: cabeza, cuello, rodillas, pies.

.

1. EL KINECT

Es un dispositivo que posee una cámara de profundidad. Las cámaras normales transforman la luz en una imagen y muestran lo que enfocan, en cambio, el Kinect con su cámara de profundidad obtiene la distancia de los objetos que se encuentran frente a él, esto lo logra mediante la utilización de una luz infrarroja, y además registra la ubicación del objeto en el espacio.

El Kinect se conecta y se comunica con la computadora a través del puerto USB, al igual que otros dispositivos como los teclados, *mouse*, entre otros. Todos los dispositivos que se comunican por el puerto USB requieren de software "*drivers*" para poder funcionar. Los controladores o drivers de los dispositivos son piezas especiales de software que se ejecutan en el equipo y que se comunican con un hardware externo. Cuando el *driver* está instalado en la computadora no se necesita de otro software para comunicarse con el dispositivo, debido a que su software no necesita conocer la marca en particular del dispositivo, porque este tiene un driver que lo hace funcional para todos los programas.

Microsoft solo destina el Kinect para trabajar con la consola Xbox 360, por lo que no dio a conocer los conductores que permiten a los programas de acceso a la Kinect en ordenadores personales normales; pero mediante la creación de un controlador de código abierto para el Kinect se logró que este Kinect sea accesible a todos los programas en todos los sistemas operativos.

1.1. Cómo funciona el Kinect

El Kinect cuenta con un proyector IR, una cámara RGB, una cámara IR, cuatro micrófonos, un motor pequeño con una serie de engranes con un límite de 30 grados de movimiento para abajo o arriba y un acelerómetro para el control del motor.

Figura 1. **Kinect sin case**

Fuente: GREG Borenstein. *Making Things See*. p. 23.

1.1.1. Proyector de luz infrarroja (IR)

El proyector de luz infrarroja o emisor de rayos infrarrojos dispara una cuadrícula de puntos infrarrojos sobre todo lo que esté delante de él, estos puntos no pueden ser percibidos por el ojo humano, pero es posible capturar una fotografía de ellos con la ayuda de una cámara o una videocámara de infrarrojos.

Figura 2. **Fotografía de la cuadrícula de puntos infrarrojos**

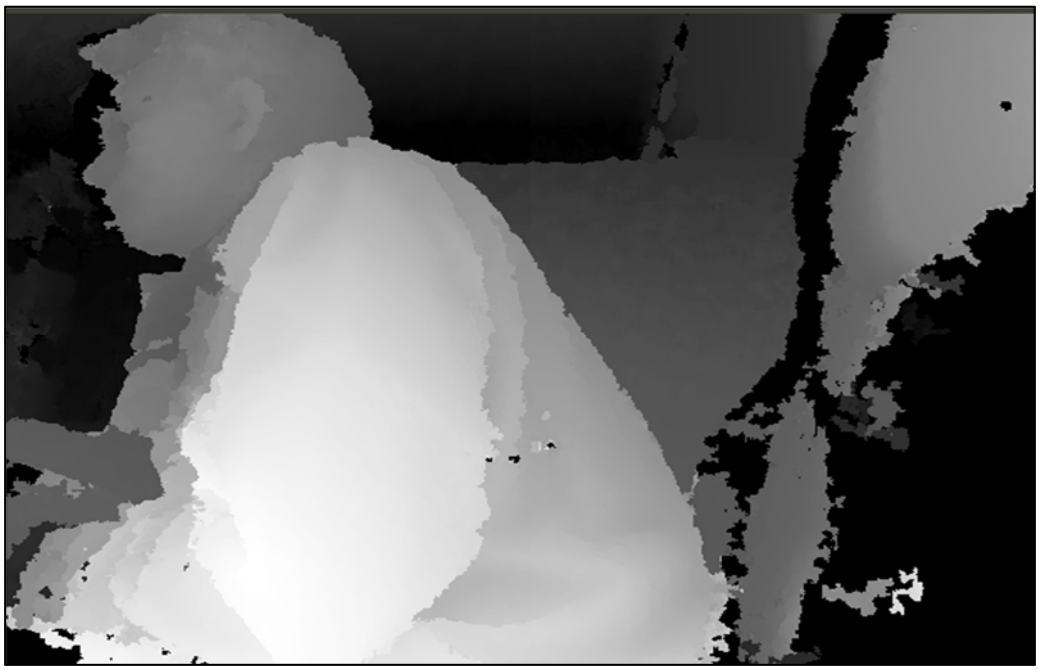
Fuente: Gadgetoblog. http://goo.gl/y55rtU. Consulta: 6 de julio de 2015.

1.1.2. Cámara de luz infrarroja (IR)

Los rayos de luz infrarroja o IR emitidos por el proyector son captados mediante la reflexión o refracción de la luz, mediante el sensor de infrarrojos que es la cámara IR, también llamada cámara de profundidad debido a que la luz captada por esta se convierte en la información necesaria para medir la distancia entre los objetos y el Kinect.

Esta información de profundidad es útil para detectar el movimiento de las personas, ya que actúa como un escáner 3D.

Figura 3. **Fotografía de la cámara de profundidad**

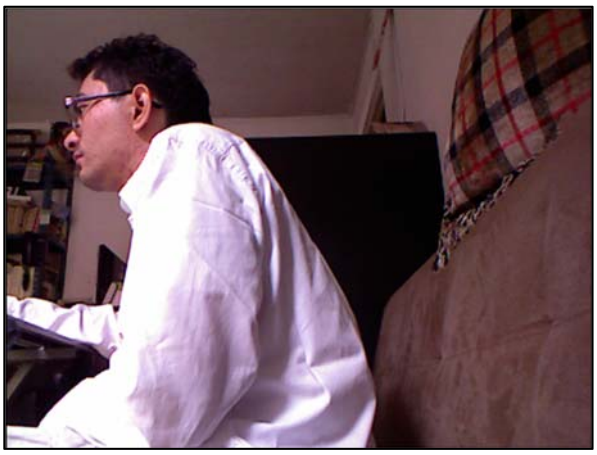


Fuente: elaboración propia.

1.1.3. Cámara de color (RGB)

Es una cámara web pequeña con resolución de 640x480 píxeles. La cámara de color sirve para crear exploraciones en 3D o entornos virtuales con color realista al combinar su información con la cámara de profundidad, pero ello implica una carga más alta de procesamiento de datos, por lo que hay que limitar los puntos de profundidad que se usarán para crear los entornos realistas en 3D.

Figura 4. **Fotografía de la cámara de color**



Fuente: elaboración propia.

1.1.4. Micrófonos

El Kinect contiene una serie de cuatro micrófonos en fila para capturar los sonidos en todas las direcciones. Este tipo de micrófono es capaz de separar los sonidos que hay justo adelante del resto de sonidos del entorno. En cada uno el sonido llega en diferentes instantes de tiempo, por lo cual se usa para localizar el sonido. Debido a las limitaciones de la librería no es posible utilizar la matriz de los cuatro micrófonos.

1.1.5. Motor

El motor del Kinect es pequeño y está conectado con una serie de engranes, lo cual le proporciona al Kinect un límite de 30 grados de movimiento para abajo o arriba, con lo que el Kinect tiene la capacidad de apuntar al punto donde se encuentre el usuario, debido a que usa los micrófonos para localizarlo.

Al igual que con la matriz de micrófonos no se puede hacer uso de este hardware debido a las limitaciones de la librería.

1.1.6. Acelerómetro

El acelerómetro que posee el Kinect sirve para el control del motor.

1.2. Antecedentes históricos

El Kinect es una herramienta producto de muchos años de investigación académica realizada tanto por Microsoft (en su división Microsoft Research) y en comunidades especializadas en la visión por computadora para usarse en la consola de video juegos XBOX y su desarrollo fue tan grande que se ha liberado para ser usado en diferentes sistemas operativos como Windows, Linux, entre otros.

El Kinect es un periférico de entrada para la consola de videojuegos Xbox 360 de Microsoft. No obstante Microsoft no construyó el Kinect en su totalidad, es el producto de muchos años de investigación de la división Microsoft Research de Microsoft Academic Search y de las comunidades de visión por computador. Por ejemplo; Richard Szeliski de Microsoft Research creó un libro de texto basado en los cursos de visión por computador que impartió en la Universidad de Washington, el cual abarca muchos de los recientes avances que llevaron a la visión por computador.

Una parte del hardware de Kinect fue desarrollado por PrimeSense, una compañía israelí que construyó anteriormente otras cámaras de profundidad utilizando la misma técnica básica de proyección IR. PrimeSense ha trabajado con Microsoft para producir una cámara de profundidad para que funcione con

el software de Microsoft, que desarrolló en sus investigaciones. PrimeSense licenció el diseño de hardware de Microsoft para crear el Kinect, pero todavía es propietaria de la tecnología básica.

Tan pronto como el Kinect fue lanzado, los programadores de todo el mundo comenzaron a trabajar en la creación de drivers de código abierto.

Josh Blake, un programador de la Natural User Interfaces, creó la comunidad OpenKinect para reunir a personas que trabajan en el proyecto de creación de controladores de código abierto en un esfuerzo de colaboración.

Héctor Martín Cantero creó la primera versión pública de un controlador que podía acceder a la imagen de profundidad del Kinect y obtuvo el premio de Adafruit. Luego se unió al proyecto OpenKinect, el cual continúa trabajando en mejorar y mantener los controladores hasta la presente fecha.

Uno de los primeros proyectos interesantes que demostró las posibilidades del Kinect, gracias a la creación de controladores de código abierto, fue el de Oliver Kreylos, investigador informático en UC Davis. Kreylos había trabajado extensamente con la realidad virtual y sistemas de presencia remota, por lo que la capacidades de escaneo 3D del Kinect encajaron perfectamente con su trabajo y demostró rápidamente las aplicaciones sofisticadas que utiliza el Kinect para reconstruir una escena en 3D a color, incluyendo la integración de modelos en 3D animados y controles de punto de vista. Las demostraciones de Kreylos estimularon la imaginación de muchas personas, por lo que dichas demostraciones ocuparon un lugar destacado en la presentación del artículo del New York Times sobre los principios del Kinect *hacks*.

Dan Shiffman, profesor de Telecomunicaciones Interactivas de la NYU, construyó con base en el proyecto OpenKinect una librería para usar el Kinect en Processing, el cual constituye un conjunto de herramientas de software que se utiliza para enseñar los fundamentos de programación de computadoras a los artistas y diseñadores.

En consideración de todas las implementaciones de drivers, librerías y aplicaciones para la utilización del Kinect, PrimeSense lanzó su software para trabajar con el Kinect. Además de los drivers que permitían a los programadores acceder a la información de la cámara de profundidad del Kinect, PrimeSense incluyó un software más sofisticado para procesar la imagen de profundidad para detectar usuarios y localizar la posición de las articulaciones en tres dimensiones. Llamaron a su sistema OpenNI, por Natural Interaction. OpenNI representó un gran avance para la comunidad interesada por trabajar con el Kinect. Por primera vez el Kinect se utilizó para la construcción de proyectos interactivos debido a que se puso a disposición la creación de códigos de proyectos. Lo anterior se logró gracias a que el proyecto OpenKinect estimuló el interés en el Kinect y creó muchas de las aplicaciones que demostraron las posibilidades del dispositivo.

Los datos de usuario proporcionados por OpenNI proporcionan una información precisa de la posición de las articulaciones de los usuarios (cabeza, hombros, codos, muñecas, pecho, caderas, rodillas, tobillos y pies) y en todo momento mientras estos estén usando la aplicación. Esta información es de gran importancia para las aplicaciones interactivas, ya que permite que los programadores puedan controlar sus aplicaciones a través de gestos con las manos o movimientos de baile.

2. INSTALACIÓN

Este capítulo trata acerca de los pasos para la instalación del sistema operativo Linux Ubuntu 14.04 LTS, las librería SimpleOpenNI, OpenNI, NITE, el driver SensorKinect y el software de programación Processing para poder utilizar el Kinect del XBOX360, en el sistema operativo antes mencionado.

2.1. Licencia GNU *general public license*

El acrónimo de GNU es un acrónimo recursivo *gnu not unix* que significa "GNU no es unix".

El proyecto GNU fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completo libre: el sistema GNU.

La licencia pública general de GNU o más conocida por su nombre en inglés GNU *general public license* (o simplemente sus siglas del inglés GNU GPL) es la licencia más ampliamente usada en el mundo del software y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios. Esta licencia fue creada originalmente por Richard Stallman, fundador de la Free Software Foundation (FSF) para el proyecto GNU.

2.2. Linux Ubuntu 14.04 LTS

El sistema operativo basado en GNU/Linux se distribuye como software libre, tiene su propio escritorio de trabajo denominado Unity, cuyo nombre proviene de la ética homónima, en la que se habla de la existencia de uno mismo como cooperación de los demás.

Linux Ubuntu está compuesto de múltiples software, los cuales, generalmente están licenciados como libres de código abierto.

El patrocinador de Ubuntu es una compañía británica denominada Canonical y es propiedad del empresario sudafricano Mark Shuttleworth. El ofrece un sistema operativo de manera gratuita y se financia por medio de servicios vinculados al sistema operativo y vendiendo soporte técnico. Debido a que es sistema es libre y gratuito la empresa aprovecha a toda una comunidad global de para mejorar los componentes y extraoficialmente la comunidad de desarrolladores brinda soporte para otras derivaciones de Ubuntu que utilizan otros entornos gráficos, como KUbuntu, XUbuntu, Ubuntu MATE, EdUbuntu, Ubuntu Studio, Mythbuntu, Ubuntu GNOME y LUbuntu.10.

Canonical, además de mantener Ubuntu para computadoras de escritorio (Ubuntu Desktop), también provee de una versión orientada a servidores (Ubuntu Server), una versión para empresas (Ubuntu Business Desktop Remix), una para televisores (Ubuntu TV), una versión para tablets (Ubuntu Tablet 11), una para teléfonos inteligentes (Ubuntu Phone) y una para teléfonos con Android (Ubuntu for Android).

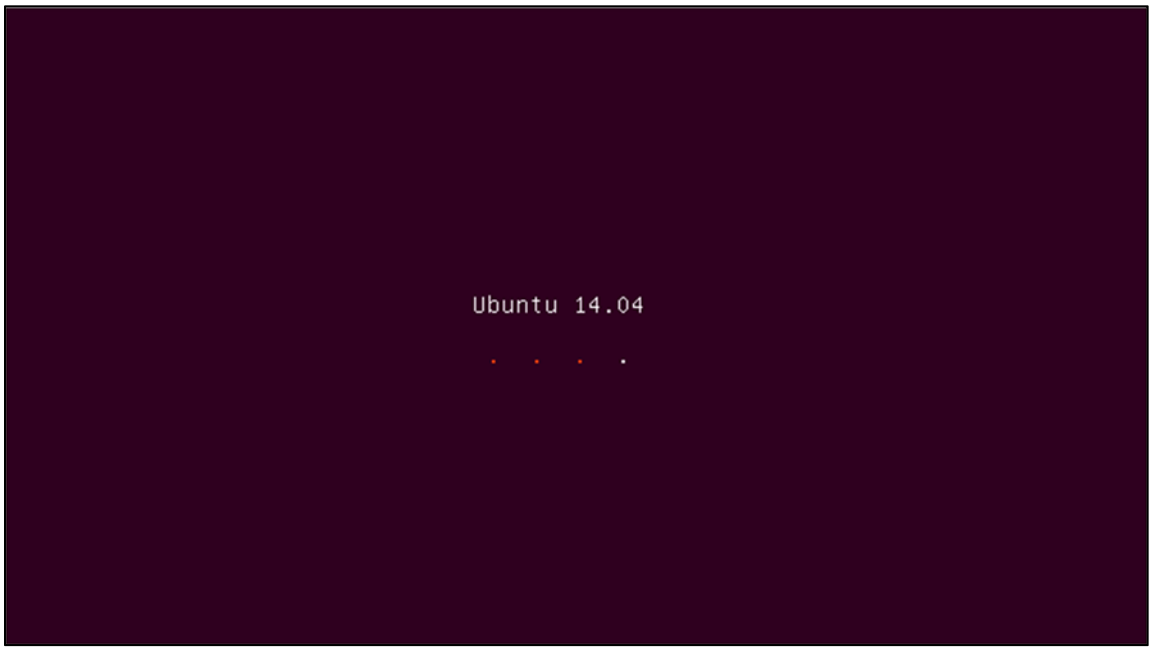
Hay versiones para computadoras de escritorio denominadas LTS (Long Term Support), las cuales son liberadas cada dos años y reciben soporte de

actualizaciones de seguridad durante cinco años en los sistemas de escritorio y servidor.

2.2.1. Instalación

A continuación se muestra con capturas de pantalla la instalación del Sistema Operativo Linux Ubuntu 14.04 LTS.

Figura 5. **Paso 1: inicio**

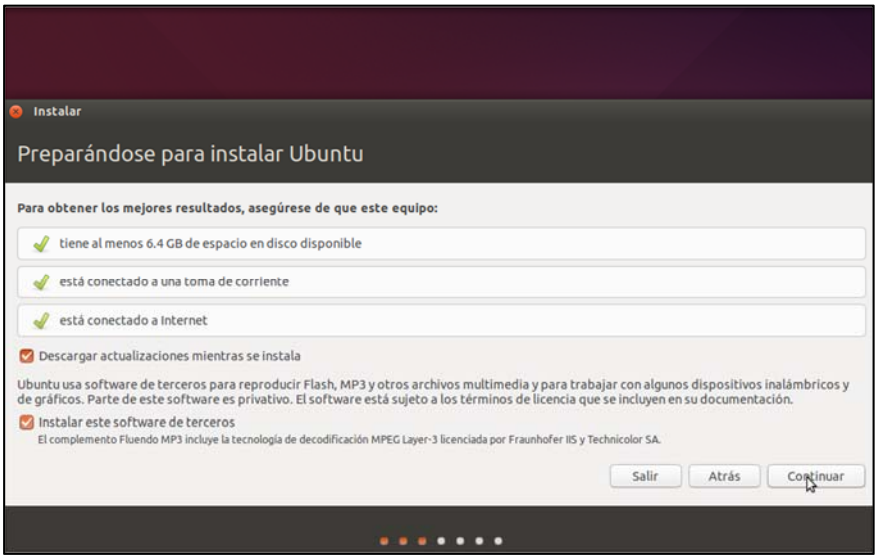


Fuente: elaboración propia.

Figura 6. **Paso 2: idioma**

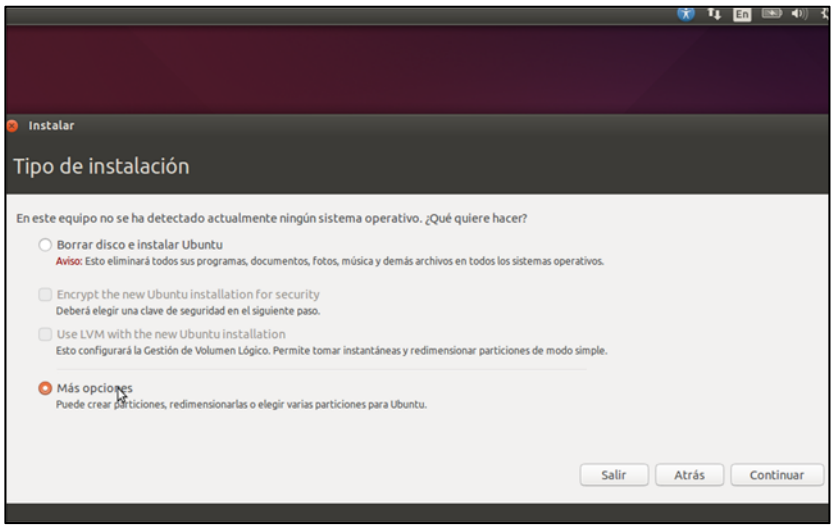
Fuente: elaboración propia.

Figura 7. **Paso 3: verificación**



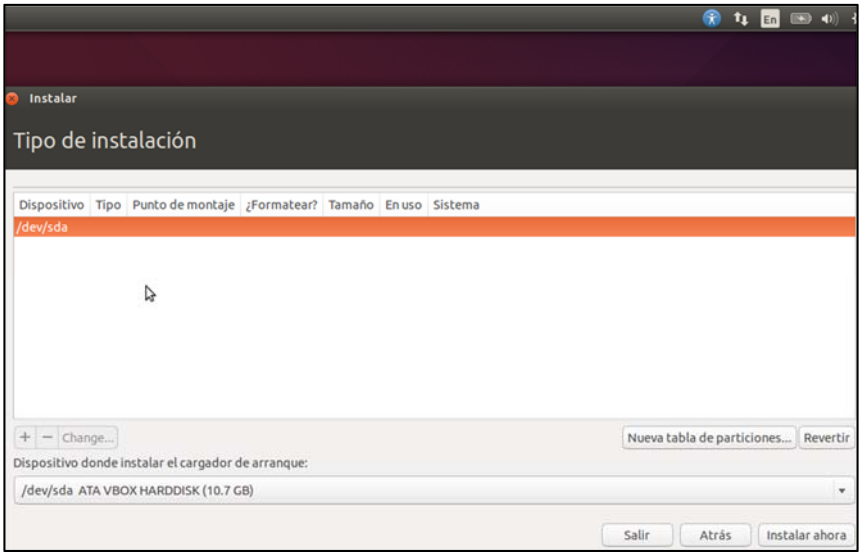
Fuente: elaboración propia.

Figura 8. **Paso 4: tipo de instalación**



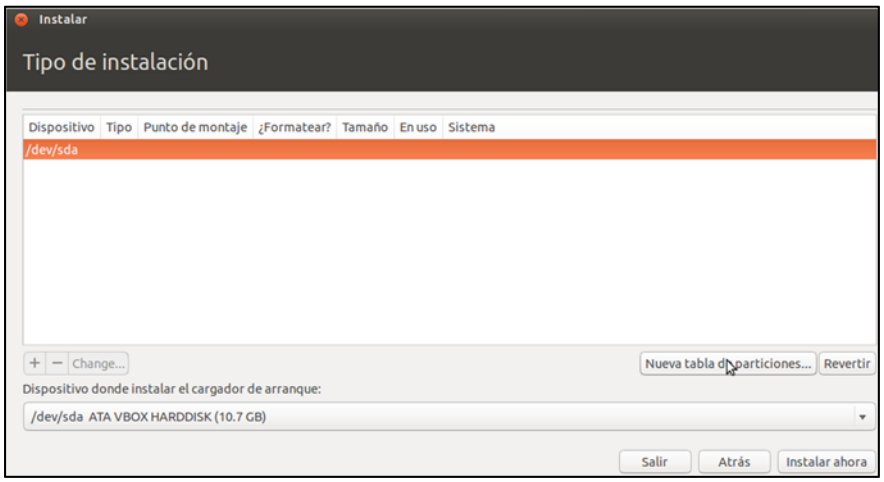
Fuente: elaboración propia.

Figura 9. **Paso 5: particiones**



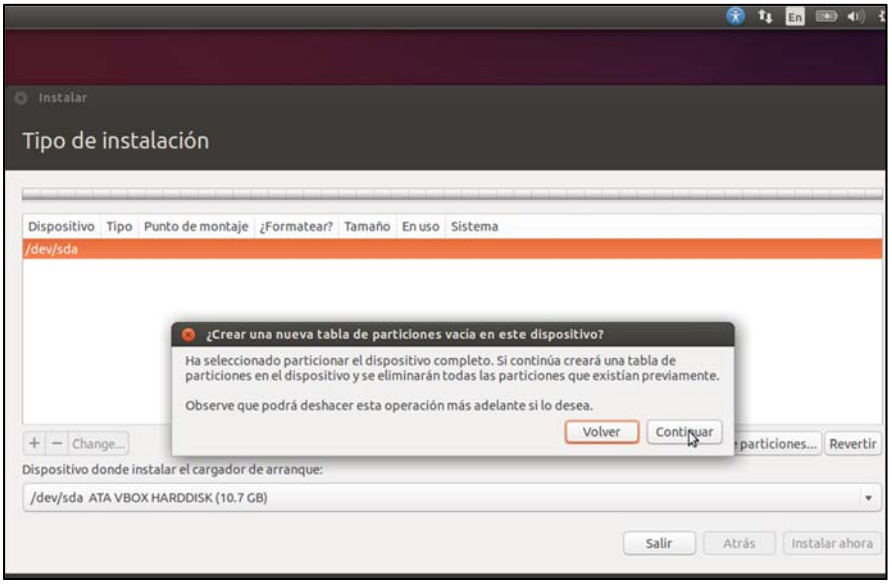
Fuente: elaboración propia.

Figura 10. **Paso 6: nueva tabla de particiones**



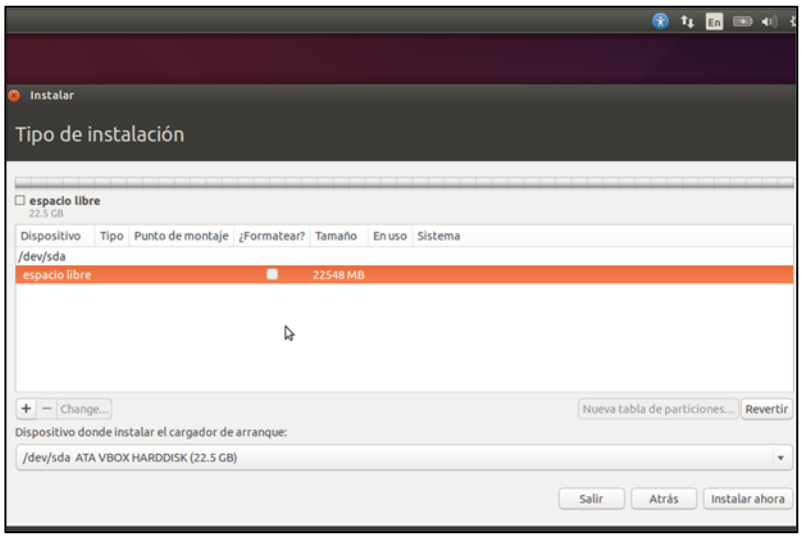
Fuente: elaboración propia.

Figura 11. **Paso 7: aceptar nueva tabla de particiones**



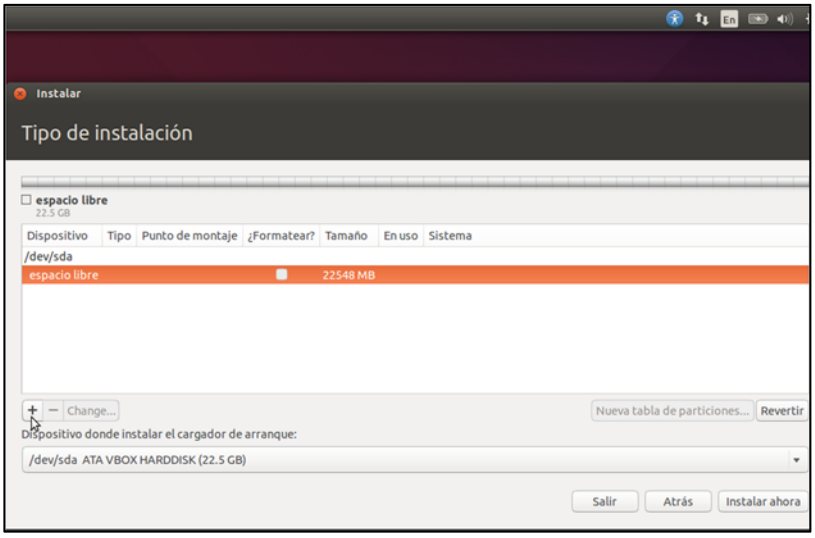
Fuente: elaboración propia.

Figura 12. **Paso 8: espacio libre del disco duro**



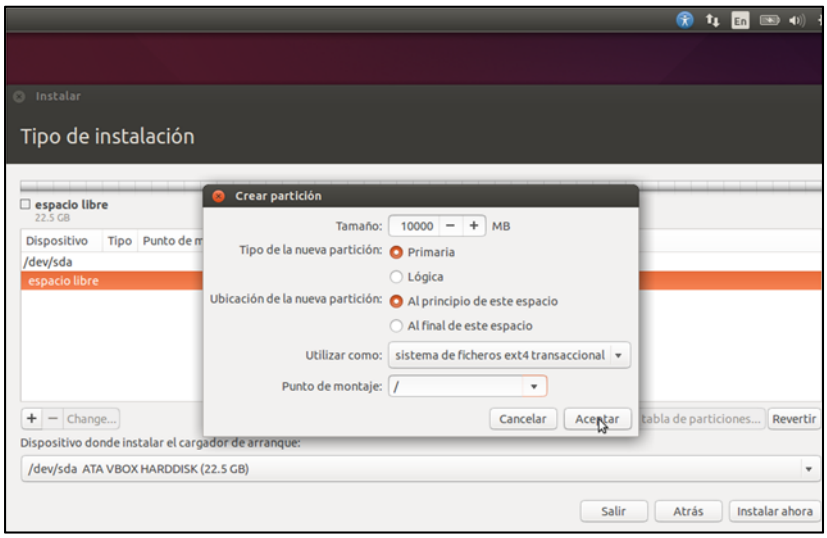
Fuente: elaboración propia.

Figura 13. **Paso 9: crear nuevas particiones**



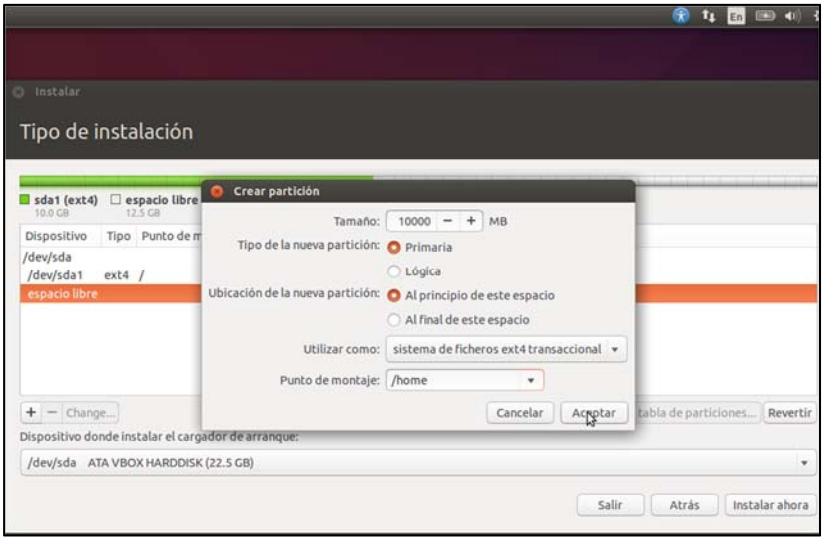
Fuente: elaboración propia.

Figura 14. **Paso 10: partición raíz**



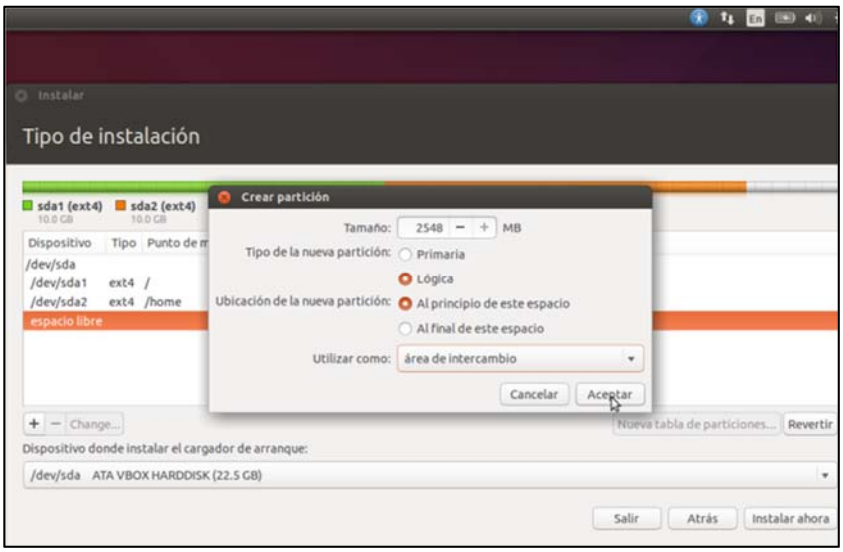
Fuente: elaboración propia.

Figura 15. **Paso 11: partición *home***



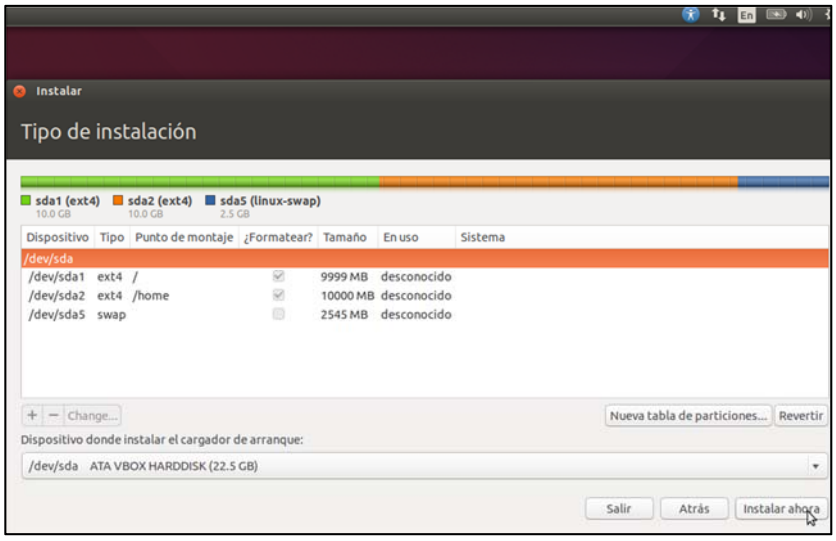
Fuente: elaboración propia.

Figura 16. **Paso 12: partición área de intercambio**



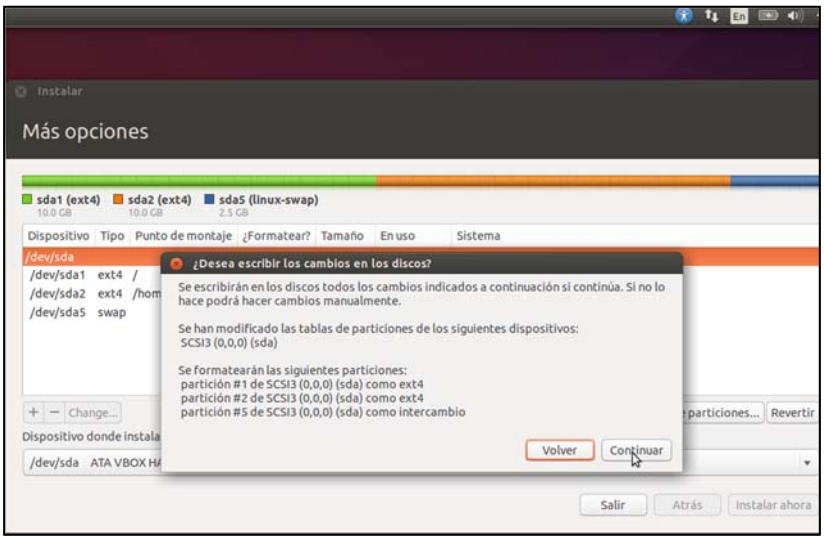
Fuente: elaboración propia.

Figura 17. **Paso 13: particiones creadas**



Fuente: elaboración propia.

Figura 18. **Paso 14: aceptar la creación de particiones**

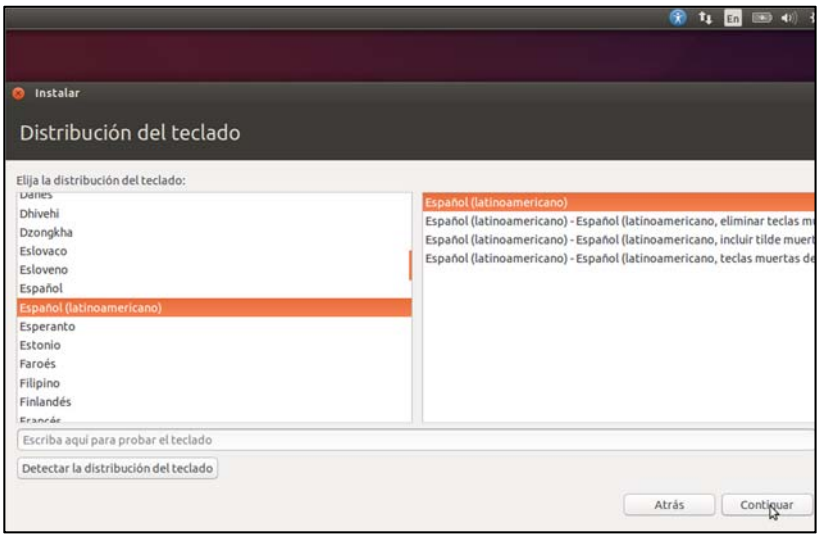


Fuente: elaboración propia.

Figura 19. **Paso 15: selección de ubicación**

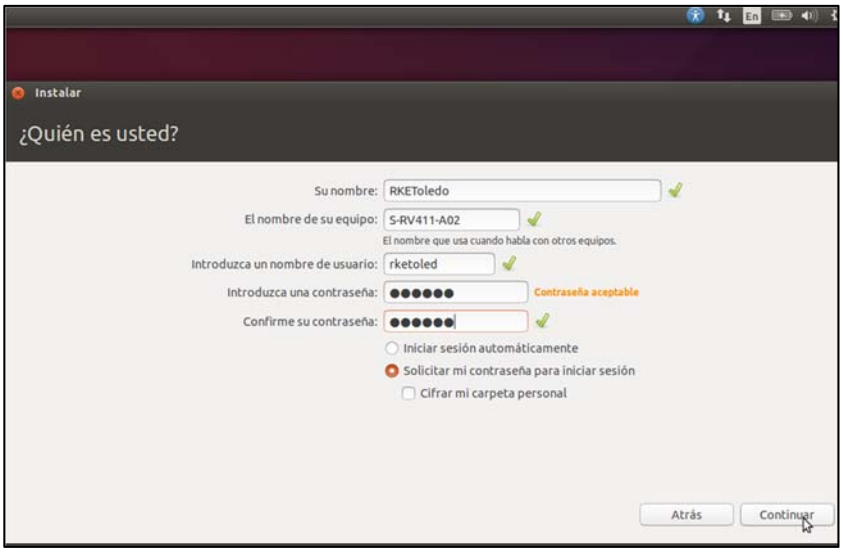
Fuente: elaboración propia.

Figura 20. **Paso 16: selección de distribución del teclado**



Fuente: elaboración propia.

Figura 21. **Paso 17: datos del usuario**

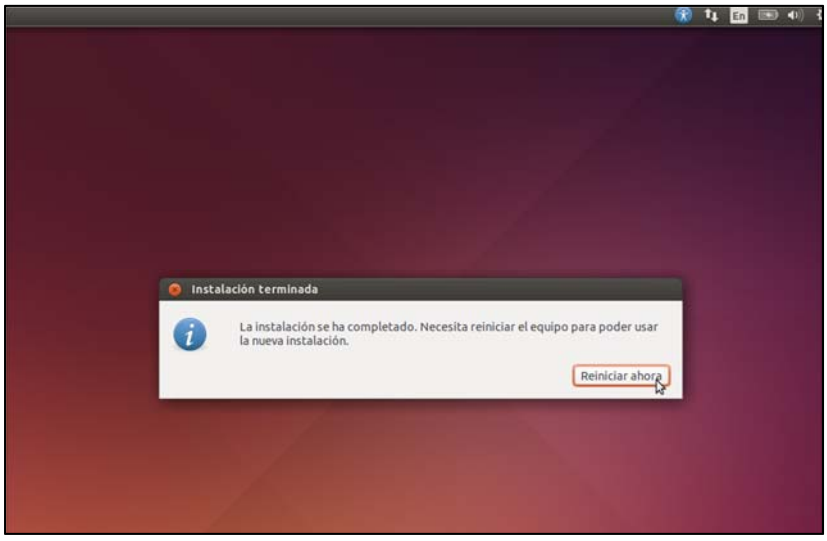


Fuente: elaboración propia.

Figura 22. **Paso 18: comienza la instalación**

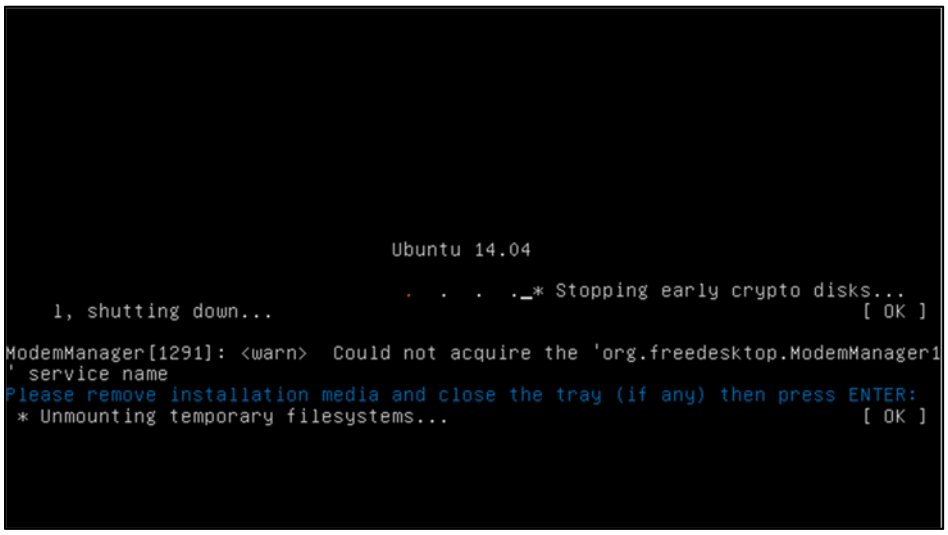
Fuente: elaboración propia.

Figura 23. **Paso 19: fin de la instalación**



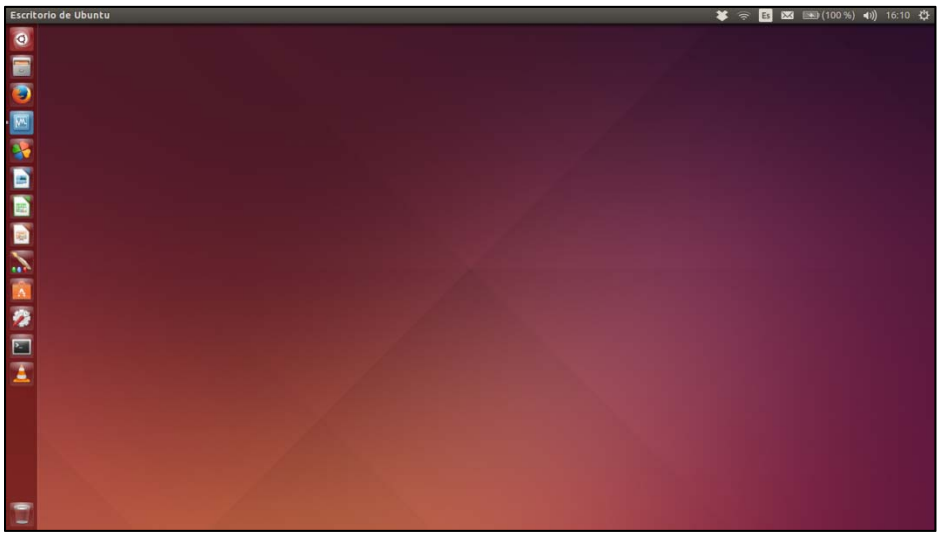
Fuente: elaboración propia.

Figura 24. **Paso 20: reinicio del sistema**



Fuente: elaboración propia.

Figura 25. **Paso 21: sistema operativo listo para usar**



Fuente: elaboración propia.

Con este paso concluye la instalación del sistema operativo Linux Ubuntu 14.04 LTS.

2.3. Administrador de archivos (*nautilus*)

GNOME Files era conocido como nautilus, el cual es el administrador de archivos del entorno de escritorio. Su anterior nombre era un juego de palabras, que evocaban el caparazón de la línea de comandos del sistema operativo.

2.3.1. Instalación

En la terminal de comandos debe escribirse el siguiente código para la instalación de nautilus:

- sudo apt-get install nautilus-open-terminal

Cuanto la instalación finalice o dé aviso de que se ha instalado debe escribirse el comando en la terminal para reiniciar nautilus o pueden reiniciar el ordenador:

- killall nautilus && nautilu

2.4. La librería de interacción natural (*OpenNI*) y el módulo externo (*NITE*)

Este es el enlace de descarga de la librería:

- https://code.google.com/p/simple-openni/downloads/list?can=1&q=OpenNI_NITE_Installer&colspec=Filename+Summary+Uploaded+ReleaseDate+Size+DownloadCount

2.4.1. Instalación de *OpenNI y NITE*

Debe crearse una nueva carpeta en la carpeta personal de Ubuntu y renombrarla como Kinect, descargar y descomprimir OpenNI_NITE_Installer-Linux64-0.27 en:

- Home/'your username'/Kinect

Las carpetas que se encuentran en su interior son las siguientes:

- OpenNI_NITE_Installer-Linux64-0.27/Kinect
- OpenNI_NITE_Installer-Linux64-0.27/Sensor-Bin-Linux-x86-v5.1.2.1
- OpenNI_NITE_Installer-Linux64-0.27/NITE-Bin-Dev-Linux-x86-v1.5.2.21
- OpenNI_NITE_Installer-Linux64-0.27/OpenNI-Bin-Dev-Linux-x86-1.5.4.0

2.5. El controlador del Kinect (*driver sensorKinect*)

Este es el *link* de descarga del *driver*:

- https://github.com/avin2/SensorKinect

2.5.1. Instalación

Descargar, descomprimir sensorKinect-unstable y renombralo como sensorKinect en:

- Home/'your username'/Kinect/OpenNI_NITE_Installer-Linux64-0.27.

Como resultado, las carpetas que se encuentran en OpenNI_NITE_Installer-Linux64-0.27 son las siguientes:

- OpenNI_NITE_Installer-Linux64-0.27/Kinect
- OpenNI_NITE_Installer-Linux64-0.27/Sensor-Bin-Linux-x86-v5.1.2.1
- OpenNI_NITE_Installer-Linux64-0.27/NITE-Bin-Dev-Linux-x86-v1.5.2.21
- OpenNI_NITE_Installer-Linux64-0.27/OpenNI-Bin-Dev-Linux-x86-v1.5.4.0
- OpenNI_NITE_Installer-Linux64-0.27/SensorKinect

2.6. Instalación con terminal parte I

Usando la terminal de comandos de Linux junto con el comando sudo apt-get install se procede a instalar los siguientes paquetes de software.

2.6.1. La librería de acceso (*libusb)*

Es una librería de C, cuya función es brindarle acceso a las aplicaciones a los dispositivos USB en muchos sistemas operativo, es un proyecto de código abierto, el cual está licenciado bajo la licencia pública general GNU versión 2.1 o posterior.

2.6.1.1. Instalación de *libusb*

En la terminal de comandos debe escribirse el siguiente código para la instalación de *libusb*:

- sudo apt-get install libusb-1.0-0-dev

2.6.2. La librería de creación (*freeglut*)

Es una alternativa de software libre o de código abierto a la librería OpenGL, se utiliza en una amplia variedad de aplicaciones prácticas porque es simple, ampliamente disponible y altamente portátil, *freeglut* se encarga de todas las tareas específicas que el sistema requiere para la creación de ventanas, la inicialización de contextos OpenGL y de los eventos de entrada.

2.6.2.1. Instalación de *freeglut*

En la terminal de comandos debe escribirse el siguiente código para la instalación de f*reeglut*:

- sudo apt-get install freeglut3-dev

2.6.3. El compilador (G++)

Es un alias tradicional de *GNU C++*, el cual es un conjunto gratuito de compiladores de C++.

2.6.3.1. Instalación de G++

En la terminal de comandos debe escribirse el siguiente código para la instalación de G++:

- sudo apt-get install g++

2.6.4. El lenguaje de programación (*Python*)

Es un lenguaje de programación que permite trabajar con mayor rapidez para integrar con más eficiencia los programas OpenGL que son verdaderamente portátiles. *Python* se desarrolló bajo una licencia de código abierto, por lo que es de libre uso y distribuible, incluso para uso comercial, la licencia de *Python* es administrada por la *Python* Software Foundation.

2.6.4.1. Instalación de *Python*

En la terminal de comandos debe escribirse el siguiente código para la instalación de P*ython*:

- sudo apt-get install P*ython*

2.7. Instalación en terminal parte II

Usando la terminal de comandos se procede a compilar para finalizar la instalación los paquetes de software que se descargarán y descomprimimos previamente:

2.7.1. Compilar e instalar la librería de interacción natural (*OpenNI*)

Abrir la carpeta OpenNI_NITE_Installer-Linux64-0.27 desde la terminal por lo que se utiliza el comando cd para ello:

- cd Kinect/OpenNI_NITE_Installer-Linux64-0.27

Posteriormente se accede a la carpeta OpenNI-Bin-Dev-Linux-x86-v1.5.4.0 y luego se procede a la instalación de la librería:

- cd OpenNI-Bin-Dev-Linux-x64-v1.5.4.0
- chmod +x ./install.sh
- sudo ./install.sh

Cuando finalice la instalación no se debe cerrar la terminal, simplemente se procede con la instalación de sensorKinect en el siguiente paso.

2.7.2. Compilar e instalar controlador del Kinect (s*ensorKinect*)

- Primer paso regresar a la carpeta OpenNI_NITE_Installer-Linux64-0.27 desde la terminal por lo que se usará el comando cd para ello:
  - cd ..
- Segundo paso: desde la terminal se accede a la carpeta CreateRedist y se instala:
  - cd SensorKinect/Platform/Linux/CreateRedist
  - ./RedistMaker
- Tercero paso: acceder a la carpeta redist y preceder con la instalación:
  - cd ../Redist/*
  - chmod +x ./install.sh
  - sudo ./install.sh

Cuando finalice la instalación no se cierra la terminal; simplemente se procede a la instalación de NITE en el siguiente paso.

2.7.3. Compilar e instalar el módulo externo (*NITE*)

Primero: regresar a la carpeta OpenNI_NITE_Installer-Linux64-0.27 desde la terminal por lo que se utilizará el comando cd cinco veces:

- cd ..
- cd ..
- cd ..
- cd ..
- cd ..

Segundo: acceder a la carpeta NITE-Bin-Dev-Linux-x64-v1.5.2.21 desde la terminal usando el comando cd y se procede a la instalación:

- cd NITE-Bin-Dev-Linux-x64-v1.5.2.21
- chmod +x ./install.sh
- sudo ./install.sh

Cuando finaliza la instalación no se debe cerrar la terminal, simplemente ejecutar algunos de los ejemplos que vienen cargados por defecto de manera que se verifique que la instalación fue satisfactoria.

2.8. Ejecución de ejemplos cargados por defecto

Regresar a la carpeta OpenNI_NITE_Installer-Linux64-0.27 desde la terminal, por lo que se usará el comando cd:

- cd ..

Segundo: acceder a la carpeta x64-Release, por lo que se usará el comando cd:

- cd Kinect/OpenNI_NITE_Installer-Linux64-0.27/OpenNIBin-Dev-Linux-x64-v1.5.4.0/Samples/Bin/x64-Release

Tercero: se utilizará el comando ls para conocer el nombre de los ejemplos y se escribirá el comando ***./*** seguido del nombre del ejemplo:

- ./nombre_del_ejemplo

Como paso final, para salir y saber si la instalación fue exitosa se usará el comando exit:

- exit

En caso de que no funcionara ninguno de los ejemplos o exista un error durante o después de finalizada la instalación, se recomienda borrar todas las carpetas que se crearon para la instalación y realizar de nuevo la instalación.

2.9. El software de programación (*processing*)

Es un software flexible con un lenguaje pensado para aprender a codificar en el contexto de las artes visuales. Desde 2001 *processing* ha promovido la lectura y escritura de software dentro de las artes visuales y la tecnología visual.

Processing es de código abierto y su descarga es gratuita permite la creación de programas interactivos en 2D y 3D, tiene integración con OpenGL

para la aceleración, puede instalarse en GNU/Linux, Mac OS X y Windows, cuenta con más de 100 bibliotecas que le proporcionan una ampliación al software principal y cuenta con documentación y muchos libros disponibles para el aprendizaje del software.

2.9.1. Instalación

Este es el *link* de descarga del software: https://processing.org/download/

Como primer paso se deben descargar, descomprimir y guardar los archivos en la carpeta Kinect:

- Home/'your username'/Kinect

Como segundo paso, utilizando la terminal de comandos de Linux se accederá a la carpeta de processing para su instalación empleando el comando cd:

- cd ~/Kinect/processing-2.2.1

Como tercer paso se inicia desde la terminal el software de *processing*:

- ./processing

Como cuarto paso se accede a las preferencias de *processing* para realizar unos cambios:

- Acceder a: File/Preferences.

- Seleccionar (increase maximun availabre memory to) y cambiar los 256 MB de memoria a 2000 (MB).
- Presionar OK para guardar los cambios.
- Sin cerrar *processing* y la terminal de comandos de Linux, proceder con la instalación de OpenNI Device Rules.

2.10. Controlador de acceso del Kinect (OpenNI *device rules*)

Los OpenNI device rules son controladores del Kinect para que se pueda trabajar con la librería SimpleOpenNI 1.96 y acceder a las cámaras de dicho dispositivo.

En la página que se indica a continuación se encuentran las instrucciones para la instalación. Se debe proceder a la instalación de los mismos:

- http://code.google.com/p/simple-openni/wiki/Installation#Linux

2.10.1. Instalación

Primero: para la instalación se siguen los pasos que se indican a continuación. Primero: desde la terminal se regresa a la carpeta Kinect con el comando cd:

- cd ~/Kinect

Segundo: escribir en la terminal el siguiente código para la instalación Linux Rules:

- wget https://simple-openni.googlecode.com/svn/trunk/SimpleOpenNI-2.0/platform/Linux/installLinuxRules.sh

Tercero: escribir en la terminal el siguiente código para la instalación Prime Sense USB:

- wget https://simple-openni.googlecode.com/svn/trunk/SimpleOpenNI-2.0/platform/Linux/primesense-usb.rules

Cuarto: escribir el comando en la terminal para poder compilar y proceder con la instalación de los *drivers*:

- chmod +x installLinuxRules.sh

Quinto: finalmente se instalan los *drivers* escribiendo el siguiente comando en la terminal:

- sudo ./installLinuxRules.sh

Sin cerrar *processing* y la terminal de comandos de Linux, proceder con la instalación de la librería SimpleOpenNI-1.96.

2.11. Librería de funciones codificadas (*SimpleOpenNI-1.96*)

Es un conjunto de funciones codificadas en un lenguaje de programación, que ofrece una interfaz funcional para el funcionamiento del Kinect en *processing*.

2.11.1. Instalación

En la página que se indica posteriormente, se descarga la librería y cuando finalice la descarga se procede con su instalación:

- http://code.google.com/p/simple-openni/downloads/list?can=1&q=SimpleOpenNI

Primero: después de finalizada la descarga descomprimir el archivo en la carpeta libraries de *processing*:

- /Home/"User Name"/sketchbook/libraries

Segundo: cerrar *processing* seguido de la terminal de Linux

Simplemente presionar la “x” para cerrar o finalizar *processing* y acto seguido escribir en la terminal el comando:

- *exit*

A continuación verificar la instalación.

2.12. Verificación de instalación y ejecución del software de programación (*processing*)

Primer paso: iniciar *processing* con los siguientes comandos en la terminal de Linux:

- cd ~/Kinect/processing-2.2.1
- ./processing

En segundo término se verifica la instalación de la librería SimpleOpenNI. Para ello, dirigirse al menú de *processing* y dar clic con el botón izquierdo del *mouse*, en el *sketch* y posicionar el puntero del *mouse* en Import Library y aparecerá en el menú que se desplegó la librería SimpleOpenNI. Con lo anterior se finaliza con todos los requerimientos necesarios para poder acceder a los datos que proporciona el Kinect.

3. IMAGEN PROFUNDA

Se usa la cámara de profundidad del Kinect porque esta permite capturar la distancia de los objetos y personas frente a él. Debido a que la cámara de profundidad proporciona una imagen en escala de grises, *processing* utiliza para calcular las distancias por medio del análisis de los píxeles de dicha imagen y así obtener las distancias.

Antes de ver las propiedades de la cámara de profundidad es importante comprender que es un píxel, por lo que a continuación se ofrece una breve descripción del mismo.

3.1. La unidad de una imagen (píxeles)

Un píxel es la unidad más pequeña de una imagen. Cada píxel constituye una unidad de color sólido y mediante la organización de muchas unidades pueden crearse imágenes digitales para realizar ilusiones como degradados suaves y sutiles matices. Con suficientes píxeles se pueden hacer imágenes lo suficientemente suaves, por lo que se obtienen imágenes realistas.

3.1.1. Componentes de la unidad de la imagen

Las componentes de un píxel son rojo, verde y azul, (por sus siglas en inglés: RGB), que combinados forman los demás colores.

3.1.2. Valor de la unidad de una imagen

Las imágenes pueden llegar a tener millones de píxeles, pero esto implica un incremento en el uso del espacio de memoria de la computadora, por lo que se usa un valor estándar, en el número de píxeles, el cual permite crear imágenes con una buena resolución y este valor es de 255. A este valor se le conoce como *image's bit depth* o profundidad de bits de la imagen, por lo tanto con 8 bits se pueden almacenar valores de 0 a 255.

Y cada valor del píxel que va desde 0 hasta 255 representa la intensidad de color del mismo, en inglés: *red, green y blue* o RGB que permite la formación de colores de la imagen, los degradados y matices.

3.1.3. El ojo humano *versus* la cámara digital

Para utilizar la cámara de profundidad del Kinect se deben comprender las diferencias entre la visión del ojo humano y una cámara digital.

La diferencia entre el ojo humano y una cámara digital reside en el procesamiento de datos, en vista de que dicho dispositivo al igual que el ordenador no distingue colores como lo hace el ojo humano, solamente puede reconocer valores numéricos, que se calculan mediante el procesamiento de datos para obtener los valores RGB de los píxeles.

3.2. Propiedades y limitaciones

A continuación se explican las propiedades y limitaciones de la cámara de profundidad del Kinect, debido a que es la que proporciona las distancias de las personas y objetos con respecto al dispositivo.

3.2.1. Resolución

La imagen que captura la cámara de profundidad tiene una resolución de 640 píxeles de ancho por 480 píxeles de alto por lo que cada cuadro de la cámara de profundidad tendrá un total de 307,200 píxeles y cada píxel realiza una doble función:

- El número de cada píxel entre 0 y 255 representará un color de gris.

- El número de cada píxel entre 0 y 255 representará una distancia en el espacio.

3.2.2. Rango mínimo

La cámara de profundidad del Kinect tiene un rango mínimo de aproximadamente 20 pulgadas u 508 milímetros. Más cerca de eso, el Kinect no puede calcular con precisión las distancias, debido a que no puede entender una profundidad exacta y lo tomará como si estuviera simplemente en el infinito por lo que la imagen tendrá un color negro y el valor del píxel tendrá un valor de profundidad de cero (0). Esto ocurre también para objetos que están a más de 8000 milímetros de la cámara de profundidad.

3.2.3. Ruido de los bordes

Cuando se observan las fotografías de los ejemplos son notorias manchas negras alrededor de los bordes de los objetos que aparecen y desaparecen en lugar de mostrar un color gris sólido. Lo anterior se debe a que el Kinect solo puede calcular la profundidad donde los puntos del proyector IR se reflejan de nuevo a él, y en los bordes de los objetos se reflejan algunos de puntos de luz

del proyector en ángulos extraños, por lo que no retornan a la cámara de profundidad y aparecen agujeros en los datos y por ende no se puede calcular la profundidad del objeto, al igual que en los objetos más cercanos de 20 pulgadas.

3.2.3.1. Suprimir las manchas o ruido en los bordes de los objetos

Para evitar este problema pueden utilizarse los datos de muchas imágenes de profundidad en el tiempo para suavizar las brechas de estos bordes. Pero este método solo funciona si el objeto no está en movimiento. Por lo que no se usará este método, debido a que el cuerpo humano es un objeto en movimiento.

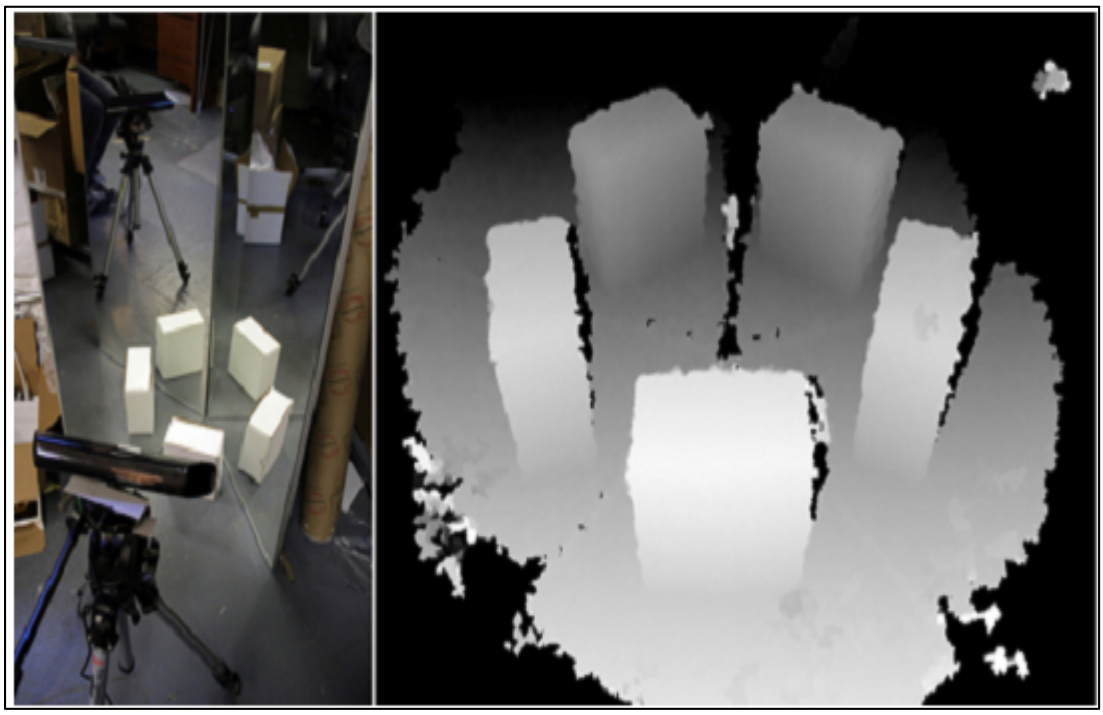
3.2.4. Reflexión

Al observar los ejemplos se nota que los espejos, ventanas o cristales a pesar de estar en el rango adecuado para que el Kinect detecte su distancia, aparecen con un color completamente negro a diferencia de los demás objetos. Esto se debe a que son objetos reflectantes y por ello los rayos de luz del proyector IR son reflejados por el espacio hasta que llegan a otro objeto no reflectante y rebotan de regreso a la cámara de IR, por lo que aparecen de color negro como si estuvieran infinitamente lejos del Kinect.

3.2.4.1. Visión de 360 grados

El fenómeno de la reflexión puede ser aprovechado por el Kinect para convertir a este en un escáner 3D capaz de realizar exploraciones de 360 grados sin necesidad de mover de posición al Kinect.

Figura 26. **Visión de 360 grados**



Fuente: GREG Borenstein. *Making Things See*. p. 56.

3.2.5. Oclusión

Se observan las imágenes de los ejemplos de la cámara de profundidad se puede ver una sólida área de negro a detrás de la cabeza. Pero al observar la imagen de color, no se aprecia dicha sombra. Esto se debe a que el proyector de IR dispara un patrón de puntos IR y cada punto se desplaza hasta alcanzar un objeto y entonces regresa al Kinect para ser leído por la cámara. Pero con los objetos que están detrás del primero, no tienen puntos de IR ya que están atrapados en la sombra del objeto más cercano. Y por ello no hay puntos de IR, que los alcancen y el Kinect no recibirá información detallada

acerca de ellos y se convertirá en otro espacio negro en los datos de la imagen de profundidad. Este problema se llama oclusión.

El Kinect no puede ver a través o alrededor de los objetos, por lo que siempre habrá partes en la imagen que estén ocluidos o bloqueados a la vista y de los cuales no se tendran datos de profundidad aproximadamente.

3.2.6. Desalineación entre la cámara de color y de profundidad

El Kinect capta imágenes en dos cámaras diferentes, una de profundidad y otra de color, estas dos están separadas una de la otra en la parte frontal del Kinect por unos centímetros y debido a esta diferencia de posición de las cámaras, estas captarán imágenes y ángulos ligeramente diferentes de la escena. Esta diferencia es como la diferencia entre los ojos humanos. Al cerrar cada uno de los ojos haciendo algunas observaciones cuidadosas, se observaran diferencias de ángulo y encuadre de lo que se está observando, asimismo esto sucede con ambas cámaras.

3.2.6.1. Alineación de la imagen de color y de profundidad

Si se alinean las imágenes de las cámaras de profundidad (IR) y de color (RGB) se obtendrán exploraciones realistas en 3D. Pero al hacer esto el ordenador tendrá que procesar una gran cantidad de datos, por lo que en el proceso se omite intencionalmente la información de varios píxeles para que las aplicaciones puedan funcionar con normalidad.

3.2.7. Distancias con la cámara de color

Las imágenes de color convencionales están fundamentalmente mal adaptadas para su procesamiento por medio de los programas de ordenador para calcular distancias. Debido a que la imagen de profundidad nos proporciona los valores de los componentes de los píxeles (RGB) y estos valores varían, dependiendo de la intensidad de la luz, no pueden ser usadas para medir.

3.3. Datos de la imagen de profundidad

La distancia real que abarca el Kinect oscila entre 20 pulgadas y 25 pies, y los píxeles de la imagen de profundidad tienen valores de brillo que oscilan entre 255 y 0. Por lo tanto, los píxeles de la imagen de profundidad tienen un brillo de 255 y dicho valor corresponde a los objetos que se encuentran a una distancia de 20 pulgadas del Kinect y los píxeles con un valor de 0 se encuentran a una distancia de por lo menos 25 pies del Kinect. Por lo que los valores intermedios deben representar el rango de distancias de los objetos.

El Kinect tiene una precisión milimétrica, por lo que puede detectar distancias de 0 a 8,000 milímetros aproximadamente. Y como los píxeles de profundidad tienen un rango de brillo de 0 a 255 habrá agujeros en esos 256 valores de brillo, debido a que solo se cubrirá una pequeña minoría de los 8,000 posibles valores de distancia. Para cubrir todos los valores de distancia, sin agujeros, se necesita una resolución más alta. Como el Kinect captura la información de profundidad con una resolución de 11 bits por píxel, las lecturas de profundidad tienen un rango de 0 a 2,047 y que es mejor que el de 0 a 255.

3.3.1. Escala de grises

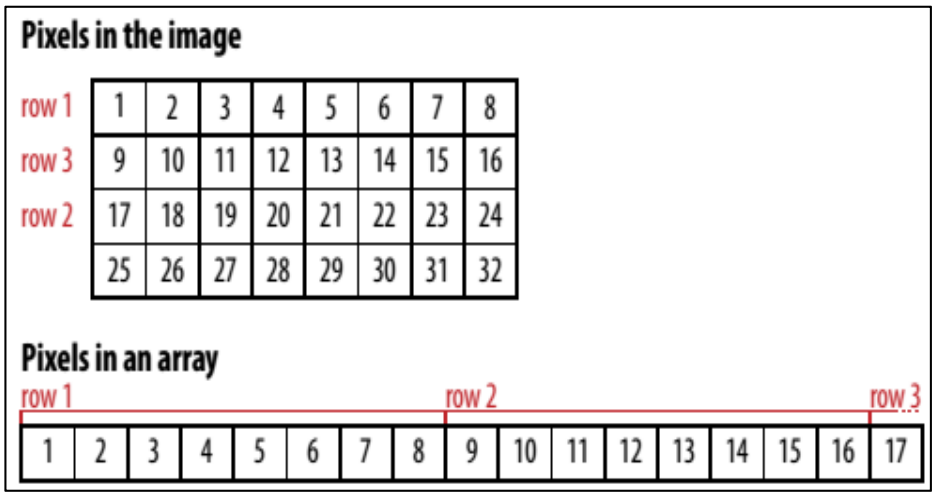
El color de un píxel se determina por la diferencia entre sus componentes rojo, azul y verde. Pero en la imagen de profundidad es una escala de grises, por lo que sus píxeles no tienen color, por lo tanto todos sus componentes son iguales. Pero un píxel de escala de grises tiene brillo, por lo que el brillo representa la distancia entre el blanco y negro.

3.3.2. Matriz de datos de dos dimensiones (2D)

Como el Kinect es una cámara, los datos que provienen de él son datos en dos dimensiones y lo que representan estos valores de profundidad en su campo rectangular de vista es una matriz unidimensional, que slo almacena una pila de números.

Se inicia con el píxel de la esquina superior izquierda de la imagen. Colocándolo en una caja y a continuación se moviliza hacia la derecha a lo largo de la fila superior de píxeles, poniendo a cada píxel en un cuadro en la parte superior. Al llegar al final de la fila se salta a la parte izquierda de la imagen, movilizando una fila hacia abajo y repitiendo el procedimiento anterior sin dejar de meter los píxeles en una caja desde la segunda fila en la parte superior de la creciente pila que comenzó en la primera. Se continúa con este procedimiento para cada fila de píxeles de la imagen hasta llegar al último píxel en la parte inferior derecha. Se obtendrá una imagen rectangular que tendrá una sola pila de píxeles o una matriz unidimensional.

Figura 27. **Matriz unidimensional de píxeles**



Fuente: GREG Borenstein. *Making Things See*. p. 56.

La imagen de la cámara de profundidad es de 640 por 480 píxeles, por lo que se tendrán 480 filas de píxeles y cada una es de 640 píxeles de ancho y por ello se obtendrá una matriz unidimensional de 307,200 (640 x 480) píxeles.

3.4. Alcance de las variables

Cuando se trata del código, el alcance de una variable describe cuánto tiempo una variable existe:

- En el interior de un determinado ciclo
- En una sola función
- Durante todo el *sketch* o programa (variable global)

3.4.1. Ventajas

Estas permiten rastrear a un usuario en movimiento por la cantidad de píxeles que cambian en sus valores, por lo que puede permitir al usuario dibujar líneas rectas en una aplicación e ignorar a un usuario que no realiza ningún movimiento.

3.5. Ejemplos de la cámara de color (RGB) e infrarroja (IR)

A continuación se proporcionarán ejemplos para entender cómo obtener y manipular los datos del Kinect, asimismo, los ejemplos estarán comentados para entender de mejor manera su funcionamiento.

3.5.1. Ejemplo 1, cámara de color (RGB) y cámara de profundidad (*depth*)

En este ejemplo se accede a la cámara de color y de profundidad del Kinect para empezar la introducción en la librería SimpleOpenNI, como se observa en la figura 28.

Figura 28. **Código del ejemplo 1, parte 1**

sketch_1Camara_RGB_y_Depth

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2014
Versión 2S 2015

Ejemplo 1
Acceder al Kinect:
Leer imagen de la cámara de profundidad.
Leer imagen de la cámara de color
Desplegar ambas imágenes en la pantalla
*/

//Importación de la librería de SimpleOpenNI
import SimpleOpenNI.*;

SimpleOpenNI EToledo;
/*
Declaración de objeto(variable)
para acceder a todos los datos del Kinect
*/

/*
La función de configuración,
declarar el tamaño de nuestra aplicación
```

Fuente: elaboración propia.

Figura 29. **Código del ejemplo 1, parte 2**

sketch_1Camara_RGB_y_Depth

```
/*
La función de configuración,
declarar el tamaño de nuestra aplicación
*/
void setup()
{
  /*
  Las imágenes que vienen del Kinect son
  de 640 píxeles de ancho por 480 de alto*
  por lo que necesitamos el doble en una dimensión
  para desplegar ambas imágenes
  */
  size(640*2, 480);
  //size(640, 480*2);
  //Declaración de la instancia
  EToledo = new SimpleOpenNI(this);
  /*
  Llamamos a dos métodos con nuestra instancia
  y habilitamos las cámaras de color y profundidad
  */
  EToledo.enableDepth();
  EToledo.enableRGB();
}

//La función desplegar información
void draw()
{
  //La función para actualizar nuestro objeto (variable)
  EToledo.update();
  /*
  El vector (0,0) indica donde debe mostrar
```

Fuente: elaboración propia.

Figura 30. **Código del ejemplo 1, parte 3**

```
  El vector (0,0) indica donde debe mostrar
  la imagen de profundidad
  */
  image(EToledo.depthImage(), 0, 0);
  /*
  El vector (0,480) indica donde debe mostrar
  la imagen de color
  recuerden que en la línea se indica la dimensión para desplegar
  ambas imágenes:
  size(640, 480*2);
  */
  image(EToledo.rgbImage(), 640, 0);
  //image(EToledo.rgbImage(), 0, 480);
}
```

Fuente: elaboración propia.

Figura 31. **Resultado del código del ejemplo 1, parte 1**

Fuente: elaboración propia.

A continuación, comentarios de las siguientes líneas de código del programa (//):

- size(640*2, 480);
- image(EToledo.rgbImage(), 640, 0);

A continuación se eliminan los comentarios (//) a las siguientes líneas de código:

- //size(640, 480*2);
- //image(EToledo.rgbImage(), 0, 480);

En la figura 32 se observa el cambio en el resultado de la imagen del Kinect.

Figura 32. **Resultado del código del ejemplo 1, parte 2**

Fuente: elaboración propia.

Como se aprecia, no se visualiza el cuadro completo de la imagen de color o RGB, ya que se encuentra fuera de la pantalla porque cuando se cambian las

líneas de código se modifican las dimensiones de la ventana de visualización y la poción donde se despliega la información de la cámara de profundidad.

3.5.2. Ejemplo 2, cámara de color (RGB) y cámara de profundidad (*depth*)

En este ejemplo se accede a los valores de los píxeles de la cámara de color y de profundidad del Kinect para poder visualizar las componentes de los píxeles.

Figura 33. **Código del ejemplo 2, parte 1**

sketch_2Buscando_un_Pixel

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2014
Versión 2S 2014

Ejemplo 2
Buscando el valor de un Píxel
*/

import SimpleOpenNI.*;
SimpleOpenNI EToledo;

void setup(){
  size(640*2, 480);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  EToledo.enableRGB();
```

Fuente: elaboración propia.

Figura 34. **Código del ejemplo 2, parte 2**

```
  EToledo.enableRGB();
}

void draw(){
  EToledo.update();
  //PImage: Variable para almacenar imágenes
  PImage depthImage = EToledo.depthImage();
  PImage rgbImage = EToledo.rgbImage();
  image(depthImage, 0, 0);
  image(rgbImage, 640, 0);
}

/*
Función que se activa cuando se
presiona el mouse en la ventana que despliega las
imágenes de color y profundidad
*/
void mousePressed(){
  /*
  Obtiene el valor del píxel usando las variables
  Especiales mouseX y mouseY las cuales son las
  coordenadas del mouse, y almacena estos datos en
  la variable tipo color "ET"
```

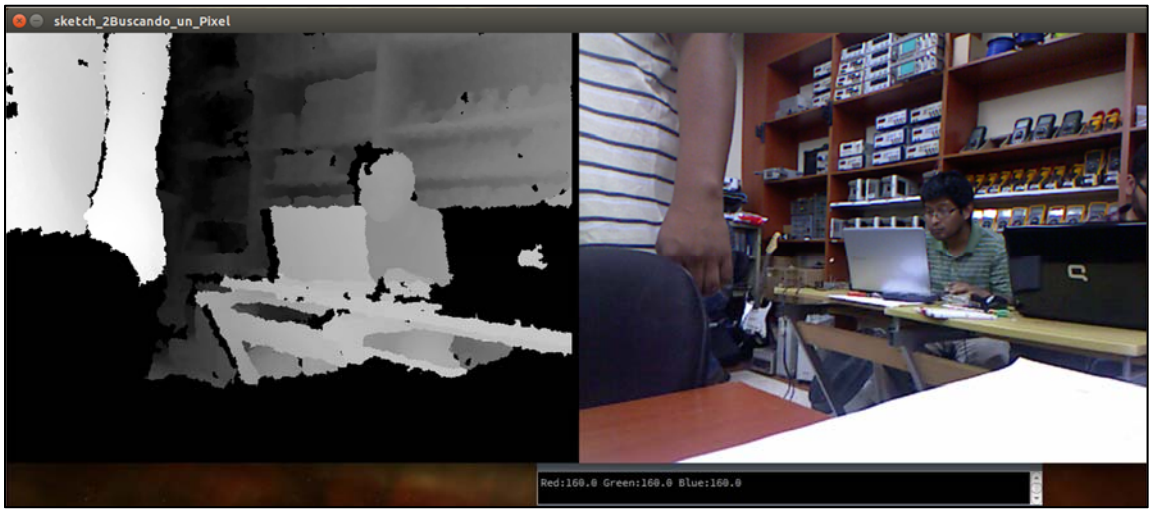
Fuente: elaboración propia.

Figura 35. **Código del ejemplo 2, parte 3**

```
  la variable tipo color "ET"
  */
  color ET = get(mouseX, mouseY);
  /*
  Imagen de color:
  Imprimir valores de píxeles de 0 a 255 (RGB).
  Imagen de profundidad (escala de grises):
  los píxeles no tienen color y las
  componentes del objeto son iguales
  el brillo significa la distancia entre el
  blanco y negro
  */
  println("Red:" + red(ET) + " Green:" + green(ET) + " Blue:" + blue(ET));
}
```

Fuente: elaboración propia.

Figura 36. **Resultado del código del ejemplo 2, parte 1**

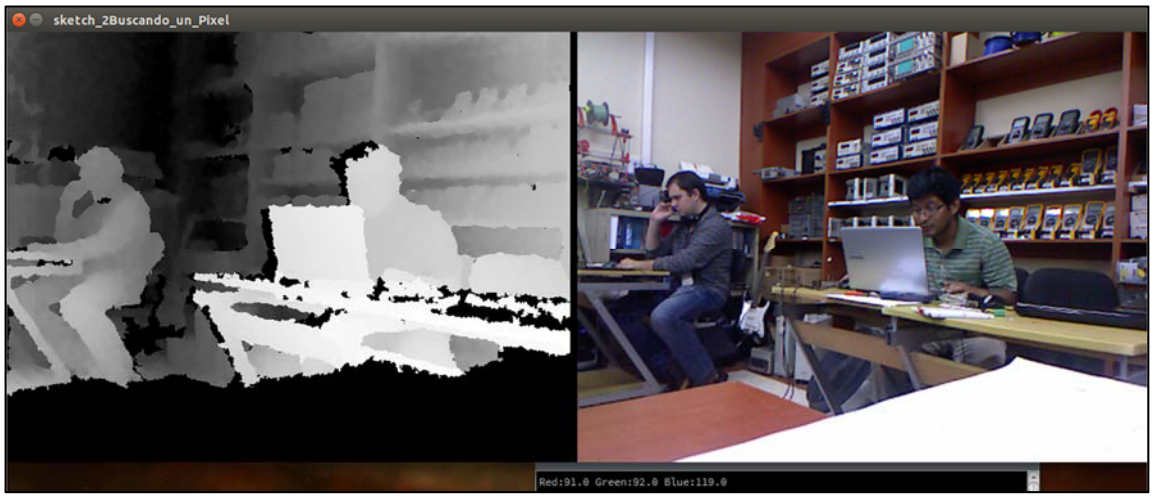


Fuente: elaboración propia.

Como se aprecia el píxel situado en la parte superior izquierda de la *laptop* de la imagen de profundidad tiene valores:

- Rojo: 160
- Verde: 160
- Azul: 160

Figura 37. **Resultado del código del ejemplo 2, parte 2**



Fuente: elaboración propia.

Como se aprecia el píxel situado en la parte superior izquierda de la *laptop* de la imagen de color tiene valores:

- Rojo: 91
- Verde: 92
- Azul: 119

Con lo cual se concluye que las imágenes de color no son útiles para obtener las distancias de los objetos frente al Kinect, debido a que estas no proporcionan un valor constante en las magnitudes de los píxeles, a diferencia de las imágenes de profundidad.

3.5.3. Ejemplo 3, Kinect como metro

En este ejemplo se accede a los valores de los píxeles de la cámara de profundidad del Kinect para calcular la distancia de los objetos que se encuentran frente al dispositivo.

Figura 38. **Código del ejemplo 3, parte 1**

sketch_3Metro_con_Kinect

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2014
Versión 2S 2014

Ejemplo 3
El Kinect como un Metro

El Kinect puede detectar objetos a una distancia de cerca
de 25 pies o 63.5 centímetros.

Y los píxeles de nuestra imagen de profundidad tienen
valores de brillo que oscilan entre los 255 y 0, por lo
que los píxeles de la imagen de profundidad que tiene un
brillo de 255 corresponde a una distancia de los objetos
de 25 pies o 63.5 centímetros de distancia.
```

Fuente: elaboración propia.

Figura 39. **Código del ejemplo 3, parte 2**

```
de 25 pies o 63.5 centímetros de distancia.

En el mapeo de pixeles de profundidad de nuestros los rangos
de brillo de 0 a 255 equivalen para el rango físico de
0 a 800 centímetros y habrán un montón de agujeros debido a
que esos 256 valores de brillo sólo cubrirán una pequeña minoría
de los 800 posibles valores de distancia.

Solución:
Para cubrir todos esos valores de distancia y sin agujeros,
necesitamos tener acceso a los datos de profundidad del Kinect
con una resolución más alta. El Kinect realmente captura la
información de profundidad con una resolución de 11 bits por pixel.
Esto significa que las lecturas de profundidad primarias tienen
un rango de 0 a 2047 que es mucho mejor que 0 a 255
disponible en los pixeles de profundidad
2^8 = 256 o 0-255
2^11 = 2048 o 0-2047

¿Porque por defecto esta a 8bits por pixel la imagen de
profundidad (PImage)? Mas memoria y mas lenta la ejecución del código.
*/

import SimpleOpenNI.*;
SimpleOpenNI EToledo;

void setup()
{
  /*
  640 columnas x 480 filas de una matriz de 2D (Pixelss).
  Los datos se llenar de derecha a izquierda y de
  arriba hacia abajo.
  */
  size(640, 480);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
}

void draw()
{
  EToledo.update();
  PImage depthImage = EToledo.depthImage();
  image(depthImage, 0, 0);
}
```

Fuente: elaboración propia.

Figura 40. **Código del ejemplo 3, parte 3**

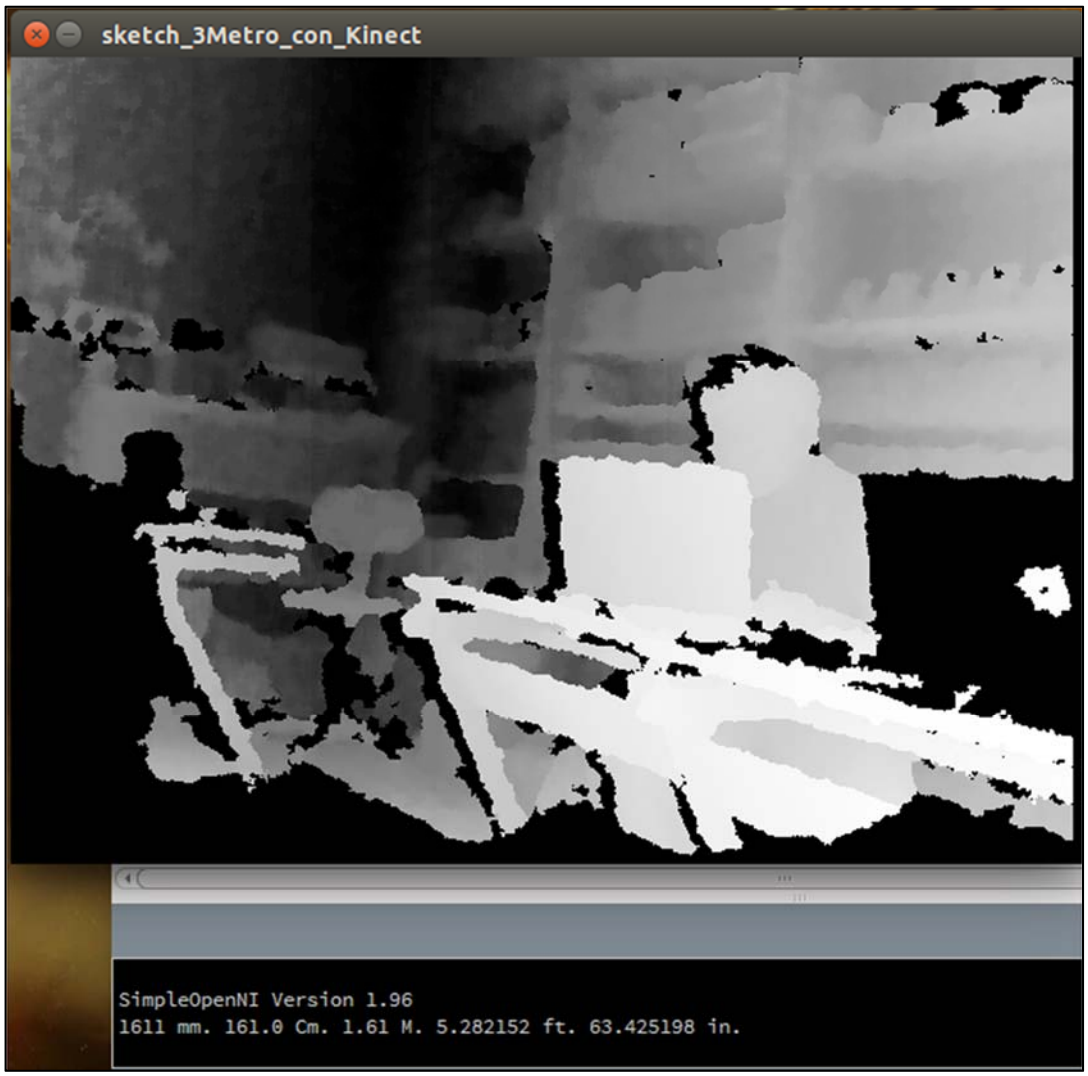
```
void mousePressed(){
  /*
  Acceder a los datos de profundidad con mayor resolución
  y guardarlos en la variable EToledo y esta variable
  sera una matriz de datos enteros de profundidad
  */
  int[] depthValues = EToledo.depthMap();
  /*
  mouseX: Esta variable nos dice exactamente que tan lejos
  estamos del lado izquierdo.

  mouseY: Esta variable nos dice exactamente que tan abajo estamos.
  Y debemos obtener la posición de este pixel la cual debe ser
  la posición del primer pixel en la fila más el número de pixeles
  entre el comienzo de la fila y este pixel *640 Columnas.
  X=No de pixeles a la derecha
  Y=No de pixeles hacia abajo
  X + Y(por los anteriores) = No. de pixel elegido.
  */
  int clickPosition = mouseX + (mouseY * 640);
  // Valores del pixel elegido
```


```
  mouseY: Esta variable nos dice exactamente que tan abajo estamos.
  Y debemos obtener la posición de este pixel la cual debe ser
  la posición del primer pixel en la fila más el número de pixeles
  entre el comienzo de la fila y este pixel *640 Columnas.
  X=No de pixeles a la derecha
  Y=No de pixeles hacia abajo
  X + Y(por los anteriores) = No. de pixel elegido.
  */
  int clickPosition = mouseX + (mouseY * 640);
  // Valores del pixel elegido
```


```
  int millimeters = depthValues[clickPosition];
  float centimeter = millimeters / 10;
  float meters = centimeter / 100;
  float feet = centimeter / 30.48;
  float inches = millimeters / 25.4;
  println(millimeters + " mm. " + centimeter + " Cm. " + meters + " M. " + feet + " ft. " + inches + " in. ");
}

/*
Ya sabemos como acceder a los datos del Kinect como una matriz
de valores, y cómo convertir esa matriz a una posición con un
valor en particular de la imagen.
*/
```

Fuente: elaboración propia.

Figura 41. **Resultado del código del ejemplo 3**



Fuente: elaboración propia.

Como se pude apreciar en la imagen, se muestra la distancia aproximada a la que se encuentra la parte superior izquierda de la *laptop*. Siempre debe tenerse en cuenta el rango mínimo del Kinect.

3.5.4. Ejemplo 4, rastreo del objeto más cercano e interfaz en tiempo real

En este ejemplo se comparan los valores de los píxeles de la cámara de profundidad del Kinect para mostrar en la pantalla, por medio de un punto de color verde, en dónde está localizado el píxel más cercano y luego se procede a unir el píxel más cercano con el píxel que anteriormente era el más cercano por medio de una línea de color azul. Presionando el botón izquierdo del *mouse* se puede limpiar la pantalla, esto solamente funcionará cuando la variable "x" ya no tenga un valor menor de 500.

Figura 42. **Código del ejemplo 4, parte 1**

sketch_4Rastreo_de_Objeto_e_Interfaz_en_Tiempo_Real

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2014
Versión 2S 2014

Ejemplo 4
4.1)
Rastreo del objeto mas cercano, Interfaz Intuitiva.
Buscar dentro de la matriz de datos de la cámara de profundidad
el punto o el píxel mas cercano.
4.2)
Interfaz en Tiempo Real
Dibujar lineas conectando los puntos de coordenadas X y Y
pasados y presentes.
*/

import SimpleOpenNI.*;
```

Fuente: elaboración propia.

Figura 43. **Código del ejemplo 4, parte 2**

```
SimpleOpenNI EToledo;

//4.1) Variables con la magnitud y posición del píxel mas pequeño
int PixelMC;
int PosPresPixelMX;
int PosPresPixelMY;
//4.2) Variables pasado
int PosPasPixelMX;
int PosPasPixelMY;
int x = 0;

void setup()
{
  size(640, 480);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
}

void draw()
{
  /*
  4.1)
  Valor inicial de la variable para recorrer toda la imagen
  ya que es el valor mas alto de la magnitud de pixeles que
  puede aparecer
  */
  PixelMC = 8000;
  EToledo.update();
  //4.1) Obtener la matriz de profundidad del Kinect
  int[] PosicionPixel = EToledo.depthMap();
  //4.1) 640 columnas x 480 filas matriz 2D de Pixelss
  //4.1) Buscar Filas
  for(int y = 0; y < 480; y++){
    //4.1) Buscar Columnas
    for(int x = 0; x < 640; x++){
      //4.1) Obtener el pixel de la matriz
      int i = x + y * 640;
      //4.1) Obtener la magnitud del pixel de la matriz
      int ValorPixel = PosicionPixel[i];
      /*
      4.1)
      Si es el pixel mas cercano guardamos su magnitud y pocicionn
      En alta resolución los puntos más cercanos en la imagen
      (Rango mínimo) tienen lecturas de profundidad de alrededor
```

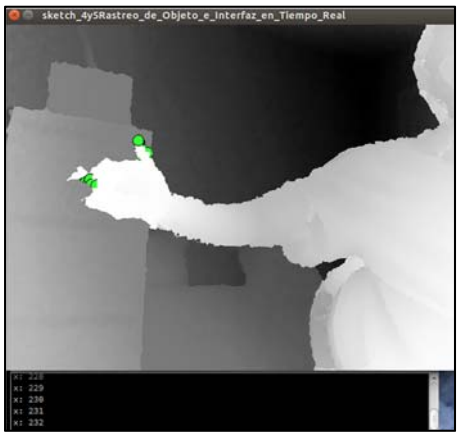
Fuente: elaboración propia.

Figura 44. **Código del ejemplo 4, parte 3**

```
      de 450, pero hay otros puntos que tienen lecturas de 0.
      */
      if(ValorPixel > 0 && ValorPixel < PixelMC){
        PixelMC = ValorPixel;
        PosPresPixelMX = x;
        PosPresPixelMY = y;
      }
    }
  }
  /*
  4.1)
  Dibuja la imagen de profundidad en la pantalla solamente si la variable
  es menor de 500
  */
  x = x + 1;
  if(x < 500){
    image(EToledo.depthImage(),0,0);
    println("x:", x);
  }
  /*
  4.1)
  Dibujar una elipse en la posición del pixel que guardamos
  (R,G,B)
  */
  fill(55,255,55);
  ellipse(PosPresPixelMX, PosPresPixelMY, 15, 15);
  /*
  5.1) Dibujar linea azul iniciando
  en los valores pasados y terminando en los valores
  presentes
  */
  /*
  stroke(55,55,255);
  line(PosPasPixelMX, PosPasPixelMY, PosPresPixelMX, PosPresPixelMY);

  PosPasPixelMX = PosPresPixelMX;
  PosPasPixelMY = PosPresPixelMY;
  */
}

void mousePressed(){
  background(0);
}
```

Fuente: elaboración propia.

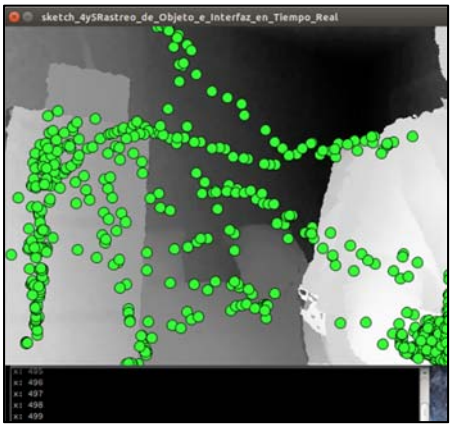
Figura 45. **Resultado del código del ejemplo 4, parte 1**



Fuente: elaboración propia.

Como se aprecia la variable “x” tiene un valor menor de 500 y no puede limpiarse la pantalla, ni visualizarse en donde están localizados los píxeles más cercanos anteriores.

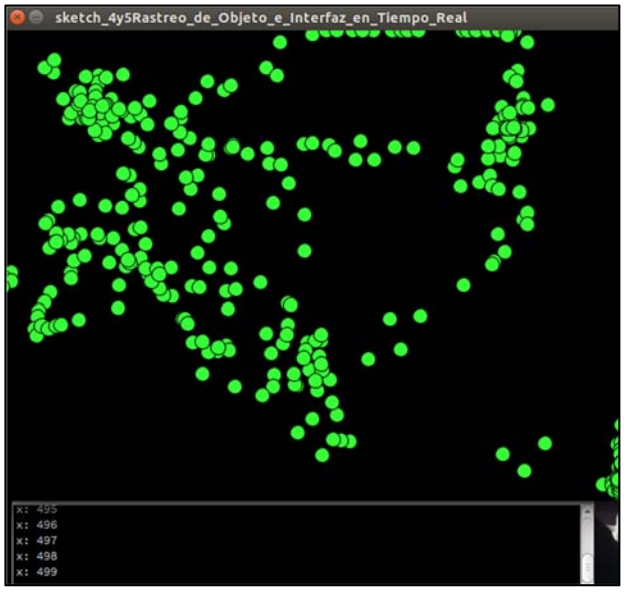
Figura 46. **Resultado del código del ejemplo 4, parte 2**



Fuente: elaboración propia.

Como se aprecia en la variable “x” esta tiene un valor menor de 500 y puede limpiarse la pantalla y visualizar en dónde se encuentran localizados los píxeles más cercanos anteriores.

Figura 47. **Resultado del código del ejemplo 4, parte 3**



Fuente: elaboración propia.

Se limpia la pantalla al presionar el botón izquierdo del *mouse* y la ventana de visualización se torna de color negro y se puede visualizar más claramente donde está el píxel más cercano y los píxeles más cercanos anteriores.

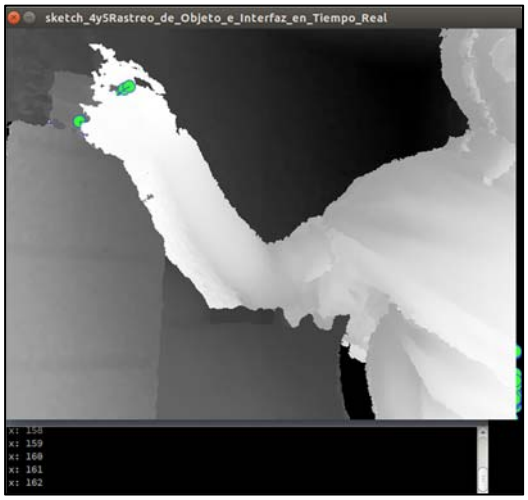
Figura 48. **Código del ejemplo 4, parte 4**

```
  5.1) Dibujar linea azul iniciando
  en los valores pasados y terminando en los valores
  presentes
  */

  stroke(55,55,255);
  line(PosPasPixelMX, PosPasPixelMY, PosPresPixelMX, PosPresPixelMY);

  PosPasPixelMX = PosPresPixelMX;
  PosPasPixelMY = PosPresPixelMY;

}
```

Fuente: elaboración propia.

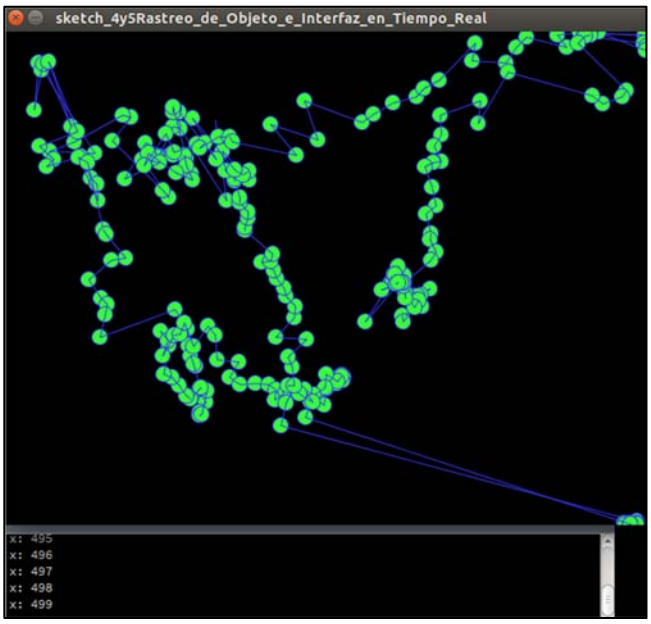
Al eliminar el comentario en el bloque de código se procede a unir con una línea de color azulada el píxel más cercano presente con el píxel más cercano pasado.

Figura 49. **Resultado del código del ejemplo 4, parte 4**



Fuente: elaboración propia.

Figura 50. **Resultado del código del ejemplo 4, parte 5**



Fuente: elaboración propia.

3.5.5. Ejemplo 5, interfaz mejorada y guardado de datos

En este ejemplo se mejora la interfaz gráfica reflejando el eje “x” para que los movimientos que realiza el usuario sean acordes a lo que se muestra en la pantalla y se guarde lo que el usuario dibujó en la pantalla.

Figura 51. **Código del ejemplo 5, parte 1**

sketch_5Interfaz_Mejorada_y_Guardar_Datos

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2014
Versión 2S 2014

Ejemplo 5
Interfaz Mejorada y Guardar Datos
*/

import SimpleOpenNI.*;
SimpleOpenNI EToledo;

//4) Variables con la magnitud y posición del pixel mas pequeño
int PixelMC;
int PosPresPixelMX;
int PosPresPixelMY;
//5) Variables pasado
float PosPasPixelMX;
float PosPasPixelMY;

void setup()
{
  size(640, 480);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  background(255);
}

void draw()
{
  PixelMC = 8000;
  EToledo.update();
  //Obtener la matriz de profundidad del Kinect
  int[] PosicionPixel = EToledo.depthMap();
  //640 columnas x 480 filas matriz 2D de Pixeles
  for(int y = 0; y < 480; y++){
    for(int x = 0; x < 640; x++){
      //Revertir imagen
      int espejo = 640 - x - 1;
```

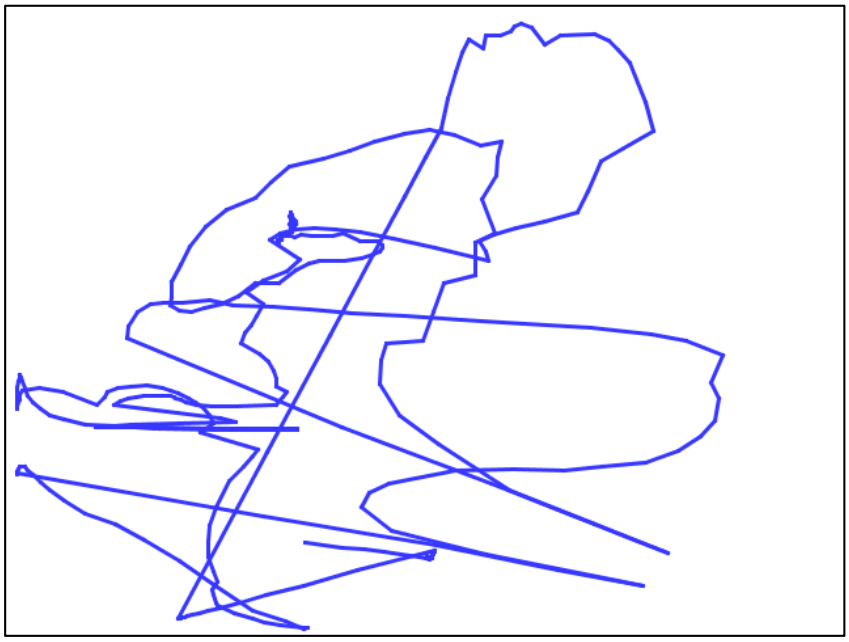
Fuente: elaboración propia.

Figura 52. **Código del ejemplo 5, parte 2**

```
      //Obtener el pixel de la matriz
      int i = espejo + y * 640;
      //Obtener la magnitud del pixel de la matriz
      int ValorPixel = PosicionPixel[i];
      /*
      Limitar a la distancia real del usuario
      533.4mm = 21in
      1016mm = 40in
      */
      if(ValorPixel > 533 && ValorPixel < 1016 && ValorPixel <PixelMC){
        PixelMC = ValorPixel;
        PosPresPixelMX = x;
        PosPresPixelMY = y;
      }
    }
  }
  /*
  Interpolación para la transición entre el punto mas cercano
  presente y el punto mas cercano pasado
  */
  float interpolarX = lerp(PosPasPixelMX, PosPresPixelMX, 0.3f);
  float interpolarY = lerp(PosPasPixelMY, PosPresPixelMY, 0.3f);
  stroke(55,55,255);//Color de la linea
  strokeWeight(3); // linea mas gruesa
  line(PosPasPixelMX, PosPasPixelMY, interpolarX, interpolarY);
  PosPasPixelMX = interpolarX;
  PosPasPixelMY = interpolarY;
}

void mousePressed(){
  //Salvar imagen y limpiar la pantalla
  save("drawing.png");
  background(255);
}
```

Fuente: elaboración propia.

Figura 53. **Resultado del código del ejemplo 5**



Fuente: elaboración propia.

Puede utilizarse el Kinect para dibujar o escribir en la pantalla como si fuera una pantalla táctil y guardar lo que se ha realizado.

3.5.6. Ejemplo 6, pantalla táctil (GLCD *touchscreen*) 640x480

En este ejemplo se utilizará el Kinect como una pantalla GLCD para controlar la posición y la escala de una imagen en la pantalla.

Figura 54. **Código del ejemplo 6, parte 1**

sketch_6Glcd_TouchScreen_640x480

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2014
Versión 2S 2014

Ejemplo 6
¿Glcd TouchScreen 640x480?
*/

import SimpleOpenNI.*;
SimpleOpenNI EToledo;

//Variables con la magnitud y posición del píxel mas pequeño
int PixelMC;
int PosPresPixelMX;
int PosPresPixelMY;
//Variables pasado
float PosPasPixelMX;
float PosPasPixelMY;

//Coordenadas de la imagen.
float Imagen1X;
float Imagen1Y;
//Declaración lógica para saber si la imagen se mueve o no.
```

Fuente: elaboración propia.

Figura 55. **Código del ejemplo 6, parte 2**

```
//Declaración lógica para saber si la imagen se mueve o no.
boolean ImagenMover;
//Variable guardar imagen.
PImage Imagen1;
//Mover o detener imagen.
int ISS = 0;

float Imagen1Escala;
int Imagen1Ancho = 100;
int Imagen1Largo = 100;

void setup()
{
  size(640, 480);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  //Si la imagen en movimiento clic del mouse para detenerla
  ImagenMover = true;
  //Cargar la imagen
  Imagen1 = loadImage("z5.jpg");
  background(0);
}

void draw()
{
  /*
  Obtener la matriz de profundidad del Kinect
  640 columnas x 480 filas matriz 2D de Píxeles
  Revertir imagen
  Obtener el píxel de la matriz
```

Fuente: elaboración propia.

Figura 56. **Código del ejemplo 6, parte 3**

```
  Obtener el píxel de la matriz
  Obtener la magnitud del píxel de la matriz
  Limitar a la distancia real del usuario
  */
  PixelMC = 8000;
  EToledo.update();
  int[] PosicionPixel = EToledo.depthMap();
  for(int y = 0; y < 480; y++){
    for(int x = 0; x < 640; x++){
      int espejo = 640 - x - 1;
      int k = espejo + y * 640;
      int ValorPixel = PosicionPixel[k];
      if(ValorPixel > 610 && ValorPixel < 1000 && ValorPixel <PixelMC){
        PixelMC = ValorPixel;
        PosPresPixelMX = x;
        PosPresPixelMY = y;
      }
    }
  }
  /*
  Interpolación para la transición entre el punto mas cercano
  presente y el punto mas cercano pasado.
  */
  float interpolarX = lerp(PosPasPixelMX, PosPresPixelMX, 0.3f);
  float interpolarY = lerp(PosPasPixelMY, PosPresPixelMY, 0.3f);
  //Limpiar pantalla
  background(0);
  /*
  Solo actualizar la posición de la imagen
  si esta esta en movimiento
```

Fuente: elaboración propia.

Figura 57. **Código del ejemplo 6, parte 4**

```
si esta esta en movimiento
*/
if(ImagenMover){
  Imagen1X = interpolarX;
  Imagen1Y = interpolarY;
  //Escala de la imagen.
  Imagen1Escala = map(PixelMC, 610,1000, 0,4);
}
//Mostrar la imagen en la pantalla.
image(Imagen1,Imagen1X,Imagen1Y,Imagen1Ancho * Imagen1Escala, Imagen1Largo * Imagen1Escala);
PosPasPixelMX = interpolarX;
PosPasPixelMY = interpolarY;
delay(70);

println(PixelMC + "mm");
if(PixelMC > 508 && PixelMC < 610){
//Mover y detener, Lógica de Boole: NOT
ImagenMover = !ImagenMover;
  println("Retardo 1s.");
  delay(1000);
  switch(ISS)
  {
    case 0:
      println("Imagen Fija.");
      ISS = 1;
    break;
    case 1:
      println("Imagen En Movimiento.");
      ISS = 0;
    break;
```

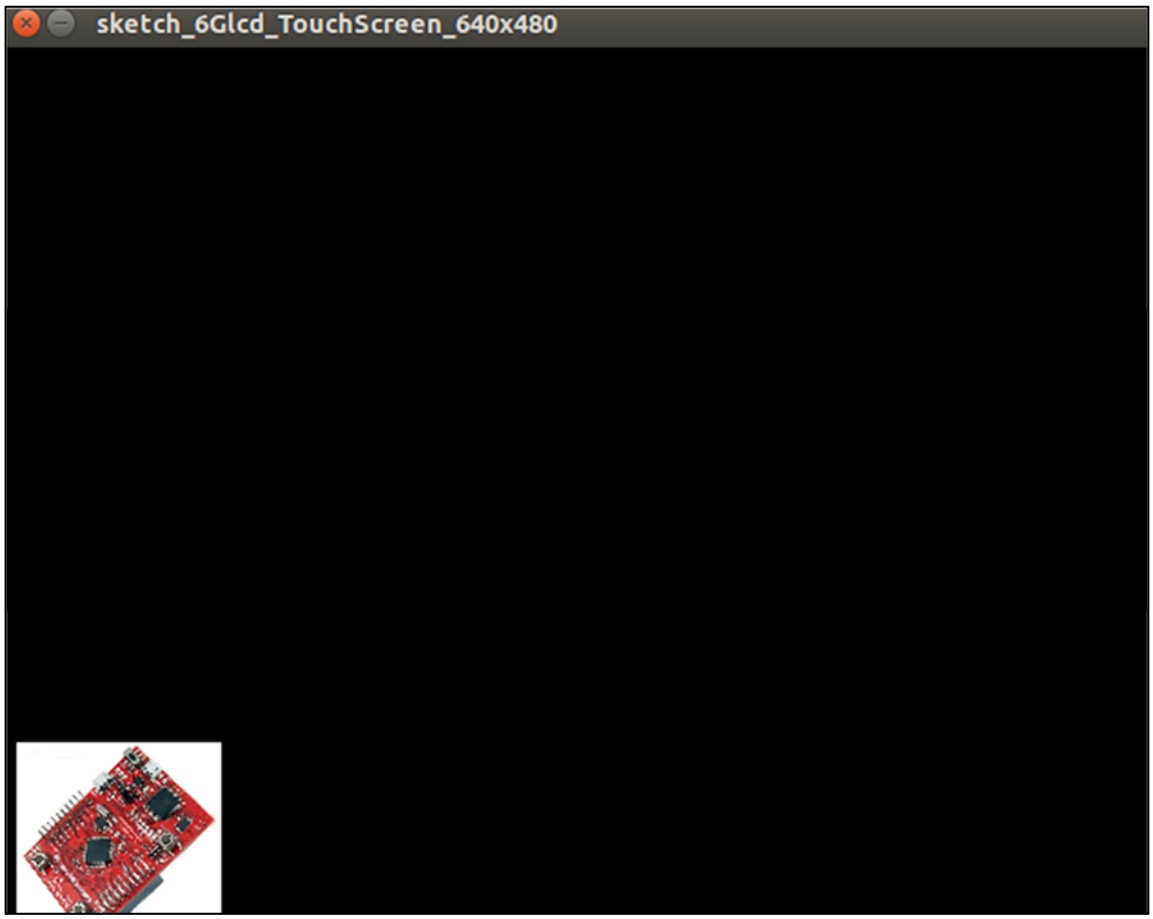
Fuente: elaboración propia.

Figura 58. **Código del ejemplo 6, parte 5**

```
    }
  }
  if(PixelMC > 1100){
  //Mover y detener, Lógica de Boole: NOT
  ImagenMover = !ImagenMover;
    println("Retardo 1s.");
    delay(1000);
    switch(ISS)
    {
      case 0:
        println("Imagen Fija.");
        ISS = 1;
      break;
      case 1:
        println("Imagen En Movimiento.");
        ISS = 0;
      break;
    }
  }
}

void mousePressed(){
   //Mover y detener, Lógica de Boole: NOT
   ImagenMover = !ImagenMover;
}

void delay(int delay)
{
  int time = millis();
  while(millis() - time <= delay);
}
```

Fuente: elaboración propia.

Figura 59. **Resultado del código del ejemplo 6**



Fuente: elaboración propia.

Como resultado de la programación realizada es posible manipular la posición y escala de una imagen por medio de la posición de los brazos.

4. NUBE DE PUNTOS

Hasta este capítulo se ha usado la información de la cámara de profundidad como una especie de metadatos sobre una imagen de dos dimensiones, por lo que se ha procesado la información de dicha cámara con el fin de sacar conclusiones acerca de la imagen. Sin embargo, existe otra manera de utilizar los datos que provienen del Kinect. Ya que se puede pensar en él como un conjunto de puntos en el espacio tridimensional y al tratar los datos de profundidad como puntos tridimensionales, es factible iniciar a mostrarlos en 3D, con lo que se crea un modelo en 3D para ser observado desde diferentes perspectivas interactivas.

Cuando se utiliza *processing* para la orientación de objetos en 3D, debe aprenderse a manipular los ejes respecto a la pantalla, dibujar formas en 3D o cargar objetos en 3D; manipular dichas formas en el espacio, entender cómo SimpleOpenNI representa los datos de las imágenes como vectores, para manipular objetos visualizándolos desde diferentes ángulos y distancias.

A continuación se hace referencia a los conceptos de iluminación y texturizado en 3D para la manipulación de los objetos en 3D.

4.1. El eje Z

En los ejemplos con los que se concluye el presente capítulo se manipulará la tercera dimensión, que constituye el eje “z”. Si el valor numérico del eje “z” es elevado significa que los objetos están más cerca de la pantalla.

Si los valores del eje “z” son menores indicará que el objeto se aleja de la pantalla, lo que se traduce como la escala del objeto.

4.2. Conjunto de puntos en tres dimensiones (3D)

En el capítulo anterior se utilizó SimpleOpenNI para acceder a los valores de profundidad y se usó como un simple arreglo de enteros. Después se tradujo esa matriz unidimensional de valores de profundidad en las posiciones “X”, “Y” en la pantalla. Pero ahora se utilizarán vectores para trabajar en tres dimensiones porque son necesarios tres números para representar cualquier punto en el espacio con una coordenada “x”, una coordenada “y” y una coordenada “z”, utilizando números enteros, por lo que se necesitan tres variables independientes para representar un punto.

4.2.1. La clase vector (PVector)

Processing proporciona una clase vector llamada PVector que permite declarar, inicializar y acceder a un vector.

Figura 60. **PVector**

```
PVector p = new PVector(1,2,3);
println("x: " + p.x + " y: " + p.y + " z: " + p.z);
```

Fuente: GREG, Borenstein. *Making Things See*. p. 114.

Con lo que puede llenarse la matriz con vectores en lugar de números enteros. Por lo tanto, cada elemento de la matriz contendrá los tres datos de las coordenadas de cada punto. Se puede simplemente recorrer la matriz de una vez en cada punto para obtener las coordenadas “x”, “y” y “z”.

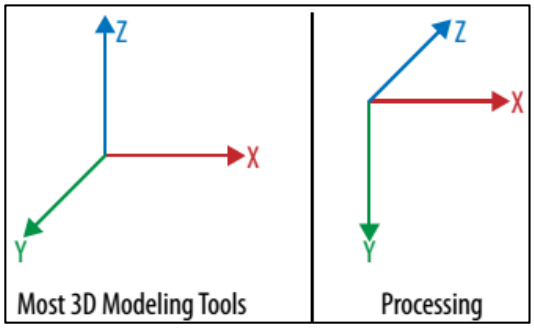
La densidad de la nube de puntos se ve limitada por la resolución de la cámara de profundidad del Kinect. Porque con los vectores se obtiene una gran cantidad de puntos de profundidad, pero no un número infinito de ellos y al igual que en la nube de puntos puede visualizarse a través de los espacios entre los puntos individuales.

4.3. Los ejes del software de programación (*processing*)

En *processing* el origen se encuentra en la esquina superior izquierda de la pantalla y valores positivos del eje “y” se mueven hacia la parte inferior. Por lo tanto, la nube de puntos aparecerá al contrario. Para resolver este problema debe girarse el boceto alrededor del eje “x”. Los valores positivos del eje “x” van hacia la derecha y-valores positivos del eje “z” van hacia el fondo.

Cuando se carga un modelo en el boceto de *processing* con una orientación particular este se orienta de tal forma, que su parte superior apunta hacia arriba del eje z. En otros programas de modelado 3D, el eje z apunta hacia arriba lejos del plano de tierra, que se define por la “x” y “y” ejes.

Figura 61. **Límites del cubo**



Fuente: GREG, Borenstein. *Making Things See*. p. 134.

Processing orienta sus ejes de forma diferente a otras herramientas de modelado, por lo que debe rotarse modelos 3D importados para conseguir que tengan la orientación correcta.

4.4. Nube de puntos a color

Para proporcionarle a la nube de puntos el beneficio de los datos a color deben alinearse las imágenes de la cámara de color y de profundidad para que ambas imágenes coincidan, por lo que es necesario acceder a los píxeles de la imagen de color e investigar qué color tienen para los puntos de profundidad. SimpleOpenNI proporciona un método para alinear la cámara de color con la de profundidad y dicho método se llama:

- alternativeViewPointDepthToImage

Únicamente debe llamarse a este método al inicio del código en *processing* y se alinearán los datos de profundidad y color.

4.5. Nube de puntos interactiva

Para que los usuarios obtengan una interfaz con la que puedan interactuar, controlar y manipular algunos objetos o imágenes, debe contar con retroalimentación en la pantalla, acerca de lo que pueden manipular. Para ello deben dibujarse formas, tales como, cubos o esferas en ubicaciones de la pantalla donde sea conveniente.Dichas formas deben coexistir con los puntos de la nube de puntos y actuar como áreas de espacio donde los usuarios pueden dirigirse para interactuar.

También debe investigarse si el usuario está interactuando con estas áreas del espacio que se han resaltado. Por ello debe comprobarse cuántos de los puntos de la nube de puntos se encuentran dentro del área definida, por lo que debe verse su posición en el espacio y aplicar un poco de lógica de Boole para considerar si cumplen con los criterios que se establezcan.

A los elementos interactivos en 3D se les llama "*hot points*".

4.5.1. Elementos interactivos (*hot points*)

Estos elementos interactivos son tal como los botones tradicionales que se encuentran comúnmente en las interfaces 2D, para activar un botón 2D se hace clic mientras que el ratón está dentro de los límites del botón (área). Pero para activar un *hot point*, debe moverse un número de puntos de la nube de puntos en los límites del *hot point* (volumen).

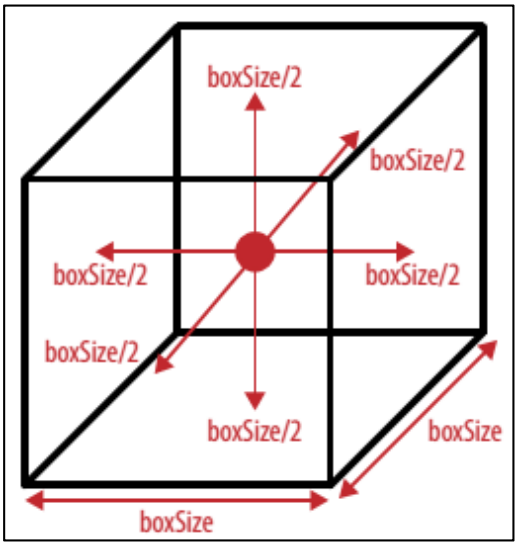
A diferencia de los botones 2D, cuyo estado es completamente binario (es decir, tienen solo dos estados: encendido y apagado), los botones 3D son más continuos, ya que puede medirse, no solo si alguna parte de la nube de puntos está dentro del botón, sino que se puede saber cuántos de sus puntos se encuentran dentro de sus límites.

Por lo que puede utilizarse este atributo a favor, con puntos calientes para dar al usuario el control de los valores continuos, así como el tono de un sonido, el brillo de una imagen, entre otros.

Cuando se obtiene el cubo o figura en la pantalla, hay que comprobar si un punto de la nube de puntos está dentro de la figura. Al crear la figura se conoce su punto inicial y sus dimensiones. Por ello, estos valores se traducen

en una serie de límites en el espacio. Si el punto satisface todos estos límites, a lo largo de cada una de sus dimensiones, entonces se sabe que está en el interior del cubo.

Figura 62. **Límites del cubo**

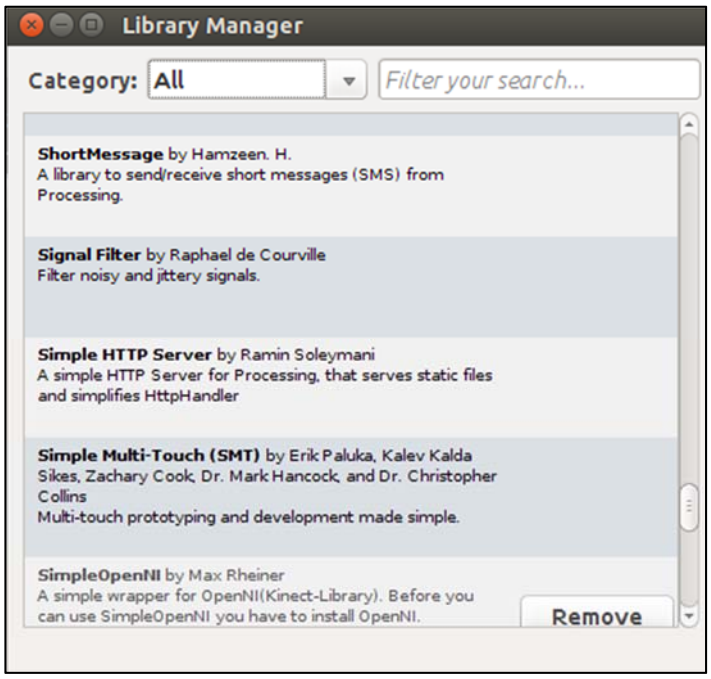


Fuente: GREG Borenstein. *Making Things See*. p. 134.

4.6. La librería de modelado (OBJLoader)

OBJLoader es una librería de *processing*, que se puede descargar de la lista de librerías de *processing*.

Figura 63. **Librerías de *processing***



Fuente: elaboración propia.

Pueden descargarse los modelos de tres dimensiones que se encuentran disponibles en varias páginas de internet como:

- http://tf3dm.com/
- http://nasa3d.arc.nasa.gov/

Al momento de descargar un modelo debe asegurarse que el archivo tenga la extensión .obj. Al descargar dicho archivo, también se obtendrá un archivo con extensión .mtl. El cual contiene los datos de los colores y texturas del objeto.

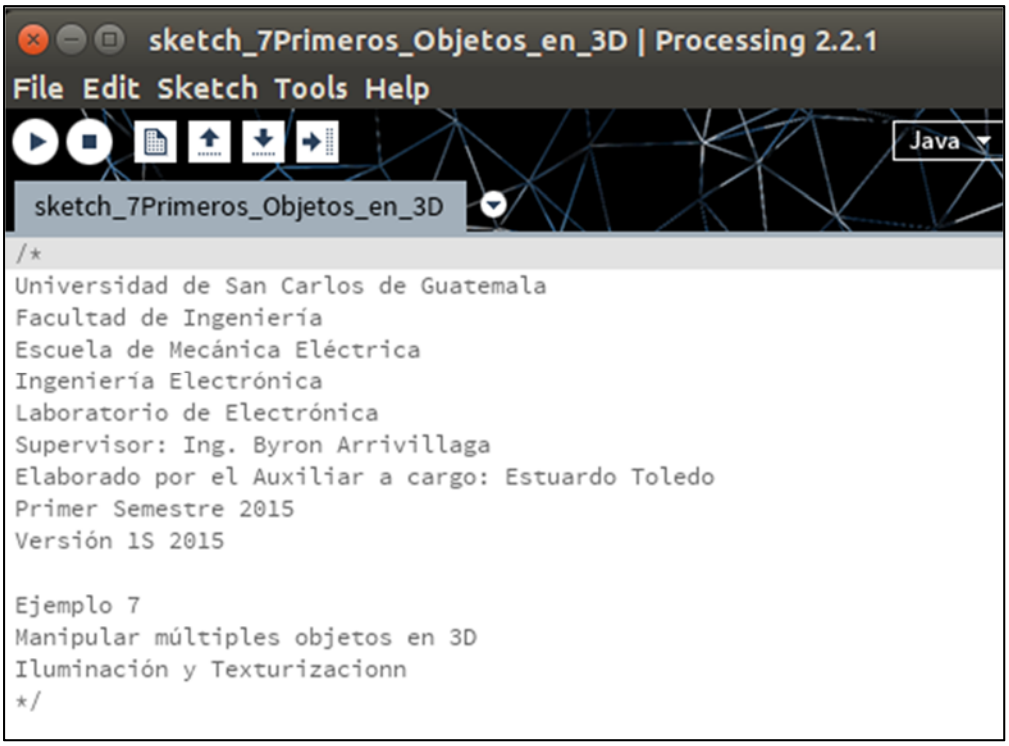
4.7. Ejemplos de la nube de puntos

A continuación se proporcionarán ejemplos para entender la nube de puntos, así como la manipulación de objetos en 3D. De igual manera, los ejemplos estarán comentados para entender su funcionamiento.

4.7.1. Ejemplo 7, primer objeto en 3D

En este ejemplo se inicia manipulando el eje z del Kinect para mover objetos 3D en la pantalla y adicionalmente se usa poca la iluminación y la texturizarían para hacer más realistas los objetos.

Figura 64. **Código del ejemplo 7, parte 1**



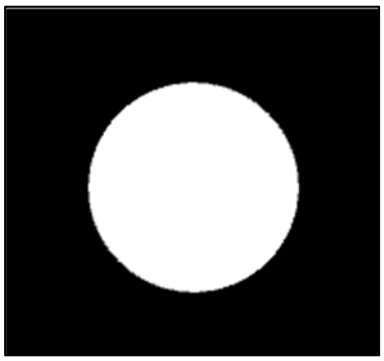
Fuente: elaboración propia.

Figura 65. **Código del ejemplo 7, parte 2**

```
/*
-------------------------
(X=0,Y=0)                -
                         -
                         -
                         -
                         -
                         -
                         -
                         -
-------------------------
Valor del eje z se acerca el objeto y viceversa
*/
int z = 100;
int x = 0;
void setup(){
  size(640, 480, P3D);
  //P3D = dibujar en 3D
}

void draw(){
  background(0);
  //traslada el circulo en el eje z
  translate(0,0,z); //x,y,z
  ellipse(width/2, height/2, 100, 100);
  if (z > 0 & x == 0) {
    z = z -1;
    if (z == 0){
      x = 1;
    }
  }
  if (z <= 100 & x == 1) {
    z++;
    if (z == 100){
      x = 0;
    }
   }
}
```

Fuente: elaboración propia.

Figura 66. **Resultado del código del ejemplo 7**



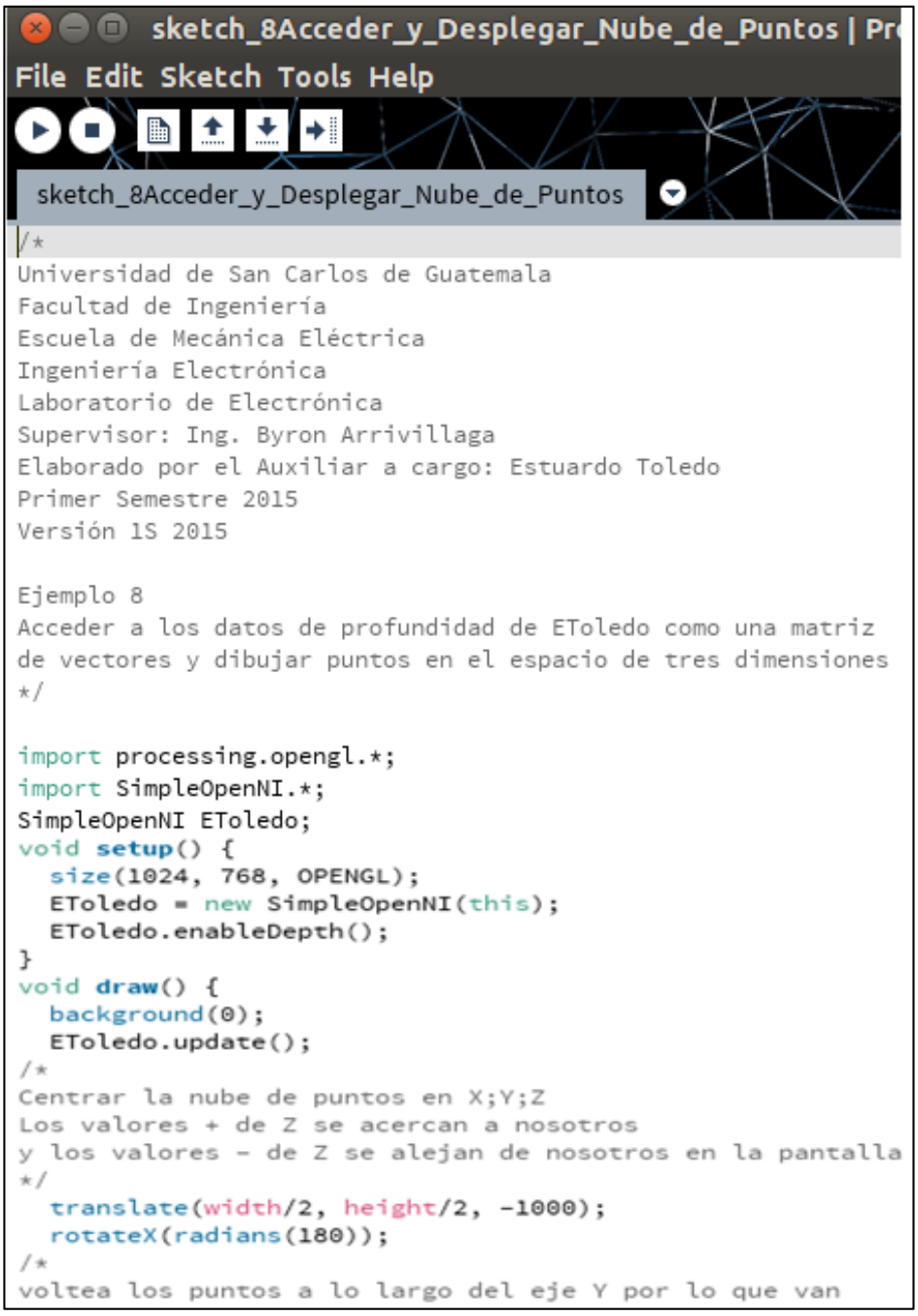
Fuente: elaboración propia.

Como puede apreciarse, el círculo se mueve sobre el eje “z”, por lo que se visualiza como el mismo se aleja y acerca a la pantalla.

4.7.2. Ejemplo 8, acceder y desplegar la nube de puntos

En este ejemplo se utiliza el Kinect para acceder los datos de la imagen de profundidad como vectores para dibujar los puntos en el espacio tridimensional.

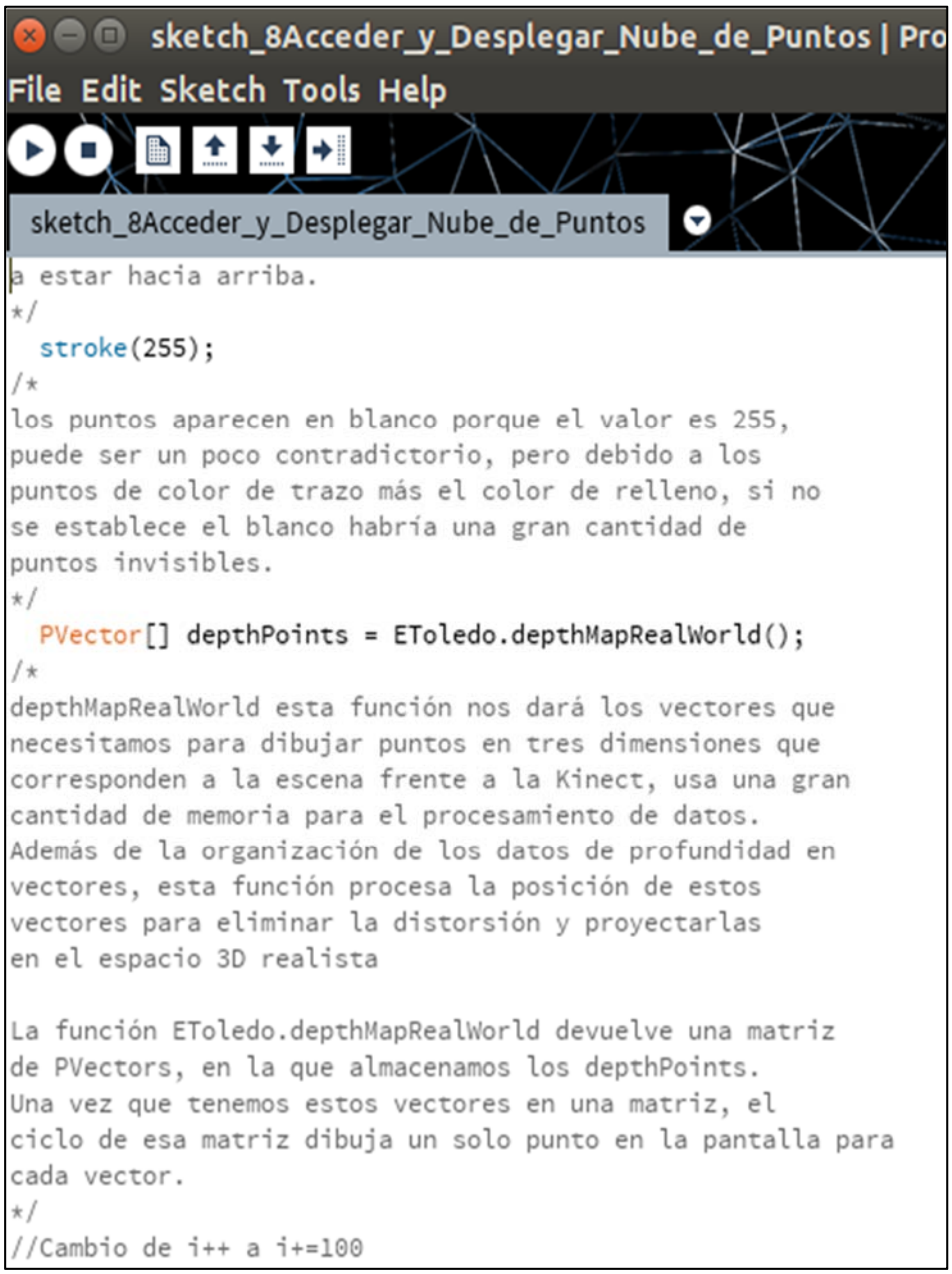
Figura 67. **Código del ejemplo 8, parte 1**




```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Primer Semestre 2015
Versión 1S 2015

Ejemplo 8
Acceder a los datos de profundidad de EToledo como una matriz
de vectores y dibujar puntos en el espacio de tres dimensiones
*/

import processing.opengl.*;
import SimpleOpenNI.*;
SimpleOpenNI EToledo;
void setup() {
  size(1024, 768, OPENGL);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
}
void draw() {
  background(0);
  EToledo.update();
/*
Centrar la nube de puntos en X;Y;Z
Los valores + de Z se acercan a nosotros
y los valores - de Z se alejan de nosotros en la pantalla
*/
  translate(width/2, height/2, -1000);
  rotateX(radians(180));
/*
voltea los puntos a lo largo del eje Y por lo que van
```

Fuente: elaboración propia.

Figura 68. **Código del ejemplo 8, parte 2**




```
a estar hacia arriba.
*/
  stroke(255);
/*
los puntos aparecen en blanco porque el valor es 255,
puede ser un poco contradictorio, pero debido a los
puntos de color de trazo más el color de relleno, si no
se establece el blanco habría una gran cantidad de
puntos invisibles.
*/
  PVector[] depthPoints = EToledo.depthMapRealWorld();
/*
depthMapRealWorld esta función nos dará los vectores que
necesitamos para dibujar puntos en tres dimensiones que
corresponden a la escena frente a la Kinect, usa una gran
cantidad de memoria para el procesamiento de datos.
Además de la organización de los datos de profundidad en
vectores, esta función procesa la posición de estos
vectores para eliminar la distorsión y proyectarlas
en el espacio 3D realista

La función EToledo.depthMapRealWorld devuelve una matriz
de PVectors, en la que almacenamos los depthPoints.
Una vez que tenemos estos vectores en una matriz, el
ciclo de esa matriz dibuja un solo punto en la pantalla para
cada vector.
*/
//Cambio de i++ a i+=100
```

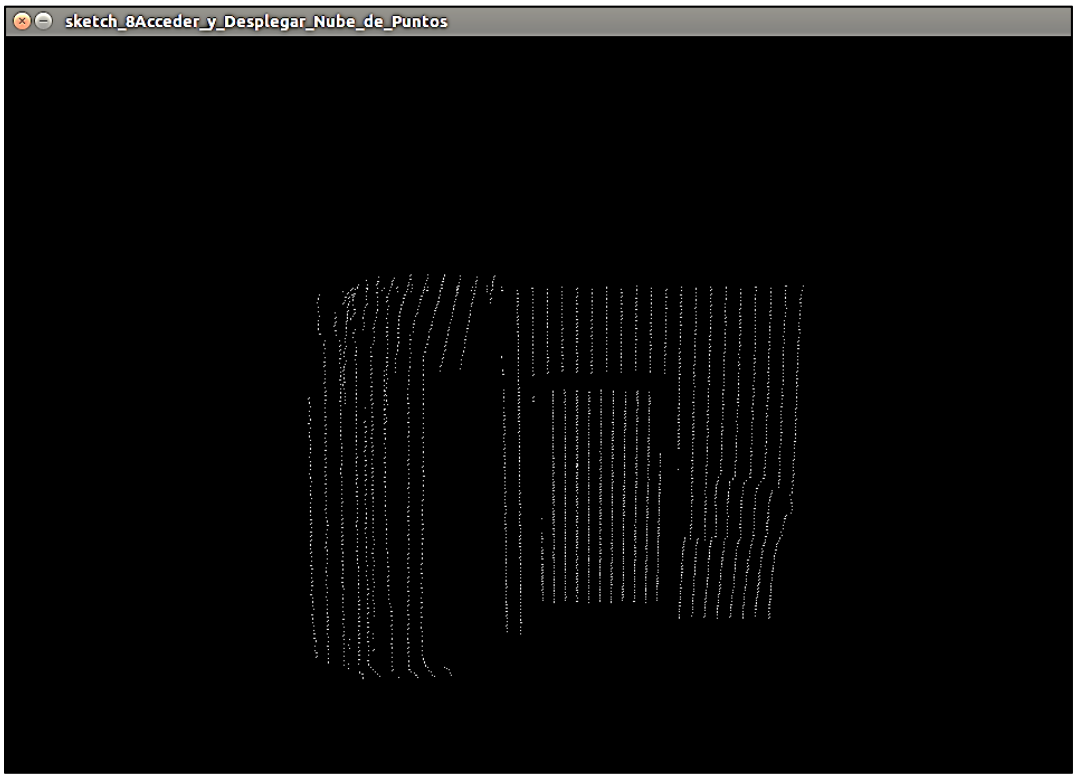
Fuente: elaboración propia.

Figura 69. **Código del ejemplo 8, parte 3**

```
  for(int i = 0; i < depthPoints.length; i+=100){
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
/*
PVector es la función de procesamiento de los tres puntos (x,y,z) que
extrae las coordenadas para dibujar los puntos con las coordenadas (x,y,z)
*/
  }
}
```

Fuente: elaboración propia.

Figura 70. **Resultado del código del ejemplo 8**



Fuente: elaboración propia.

En este se inicia la visualización de los objetos en 3D, visualizándolos como una serie de puntos.

4.7.3. Ejemplo 9, traslación y rotación de la nube de puntos

En este ejemplo se utiliza el Kinect con la posición del mouse para trasladar y rotar la nube de puntos en el espacio tridimensional.

Figura 71. **Código del ejemplo 9, parte 1**

Fuente: elaboración propia.

Figura 72. **Código del ejemplo 9, parte 2**

```
/*
Variable para contener nuestra rotación actual
representada en grados
*/
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
}
void draw() {
  background(0);
  EToledo.update();
  /*
  Trasladar la nube de puntos del origen (0,0,0) al centro
  de la ventana y el eje Z 1000 píxeles mas cerca
  */
  translate(width/2, height/2, -1000);

  //Rotación vertical de la nube de puntos
  rotateX(radians(180));

  /*
  Trasladar el centro de rotación al centro de la
  nube de puntos
  */
```

Fuente: elaboración propia.

Figura 73. **Código del ejemplo 9, parte 3**

```
translate(0, 0, 1000);

//Rotar al rededor del eje Y, con un cambio de rotación (++)
//rotateY(radians(rotation));
//rotation++;
float mouseRotation = map(mouseX, 0, width, -180, 180);
rotateY(radians(mouseRotation));
/*
Probar reemplazar:
rotateY(radians(rotation));
rotation++;
float mouseRotation = map(mouseX, 0, width, -180, 180);
rotateY(radians(mouseRotation));
*/

/*
los puntos aparecen en blanco porque el valor es 255,
puede ser un poco contradictorio, pero debido a los
puntos de color de trazo más el color de relleno, si no
se establece el blanco habría una gran cantidad de
puntos invisibles.
*/
stroke(255);

/*
depthMapRealWorld esta función nos dará los vectores que
necesitamos para dibujar puntos en tres dimensiones que
```

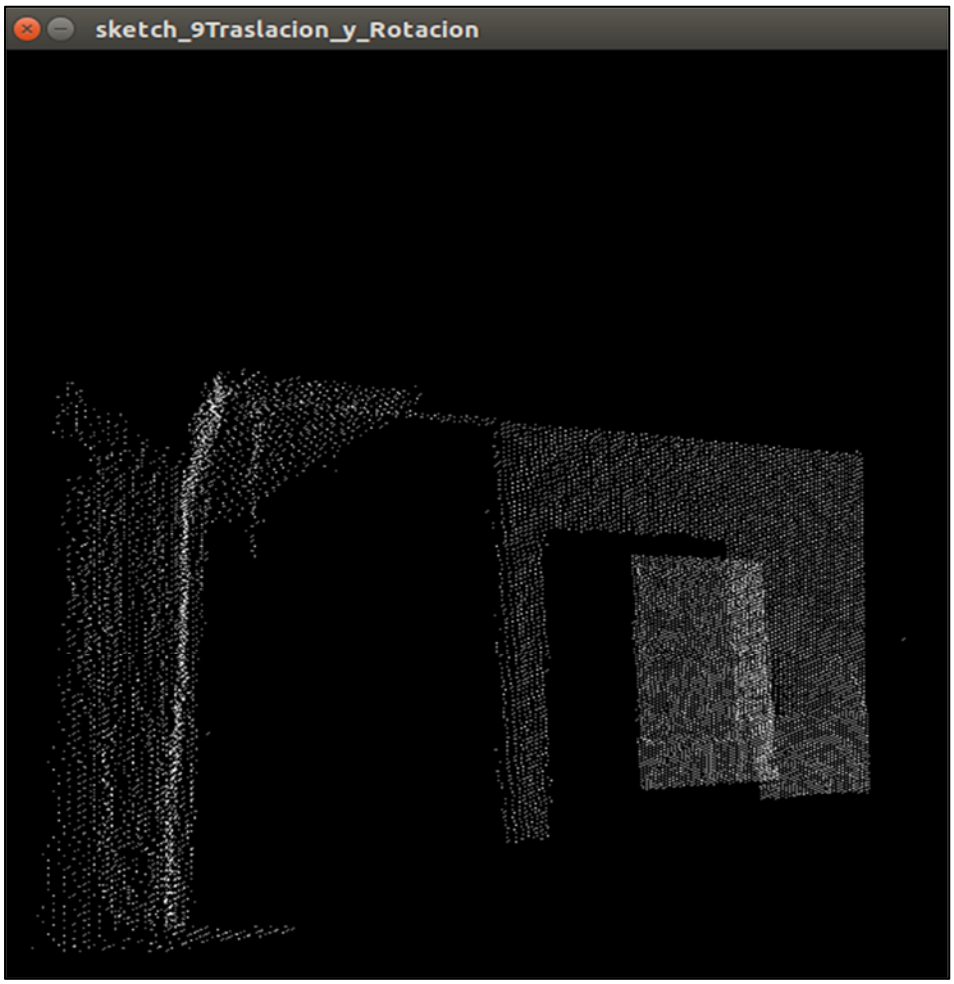
Fuente: elaboración propia.

Figura 74. **Código del ejemplo 9, parte 4**

```
  corresponden a la escena frente a la Kinect, usa una gran
  cantidad de memoria para el procesamiento de datos.
  Además de la organización de los datos de profundidad en
  vectores, esta función procesa la posición de estos
  vectores para eliminar la distorsión y proyectarlas
  en el espacio 3D realista.

  La función EToledo.depthMapRealWorld devuelve una matriz
  de PVectors, en la que almacenamos los depthPoints.
  Una vez que tenemos estos vectores en una matriz, el
  ciclo de esa matriz dibuja un solo punto en la pantalla para
  cada vector.
  */
  PVector[] depthPoints = EToledo.depthMapRealWorld();
  /*
  El cambio de i++ a i+=10
  permite un procesamiento más rápido ya que
  solo dibuja cada un cada 10 puntos.
  */
  for (int i = 0; i < depthPoints.length; i+=25) {

    /*
    PVector es la función de procesamiento de los tres puntos (x,y,z) que
    extrae las coordenadas para dibujar los puntos con las coordenadas (x,y,z)
    */
    PVector currentPoint = depthPoints[i];
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}
```

Fuente: elaboración propia.

Figura 75. **Resultado del código del ejemplo 9**



Fuente: elaboración propia.

Cambiando el valor de la variable “y” se obtiene una mejor visualización de los objetos que se encuentran frente al Kinect, pero entre menor sea el valor de la variable, más procesamiento de datos significará, y por ende, el programa será más lento. Adicionalmente se obtendrá una mejor perspectiva 3D de los objetos al poder rotar la nube de puntos, casi como se pudieran a preciar de lado.

4.7.4. Ejemplo 10, alineación imagen de color y de profundidad

En este ejemplo se usará el Kinect con la posición del *mouse* para trasladar y rotar la nube de puntos en el espacio tridimensional, o dejar este movimiento en función de una variable que se incrementa en cada ciclo del programa.

Adicionalmente usará la función alternativeViewPointDepthToImage para alinear los datos de la imagen de profundidad y de color para proporcionarle color a la nube de puntos.

Figura 76. **Código del ejemplo 10, parte 1**

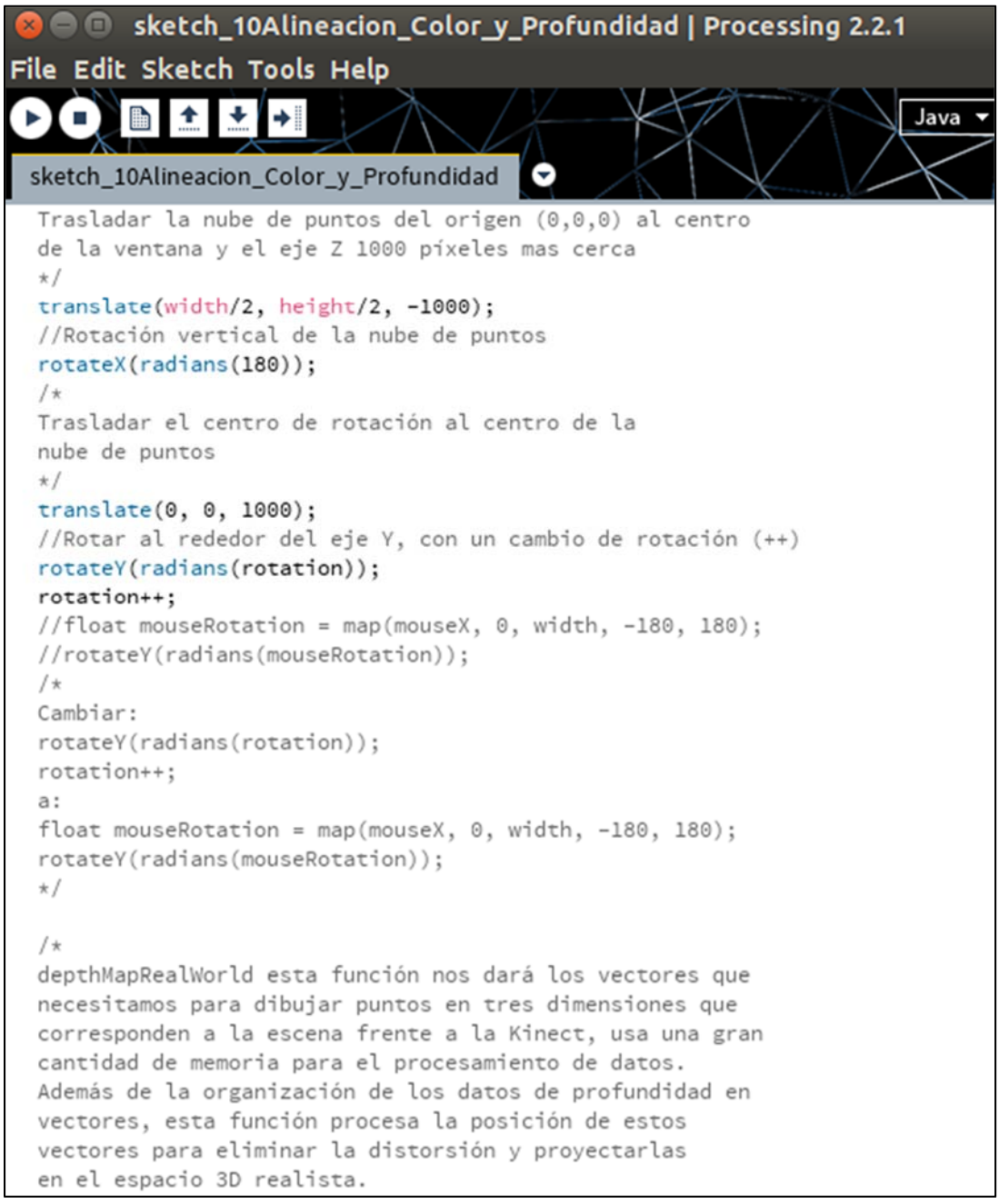
Fuente: elaboración propia.

Figura 77. **Código del ejemplo 10, parte 2**

```
import processing.opengl.*;
import SimpleOpenNI.*;
SimpleOpenNI EToledo;

/*
Variable para contener nuestra rotación actual
representada en grados
*/
float rotation = 0;

void setup() {
  size(1024, 768, OPENGL);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  EToledo.enableRGB();
  EToledo.alternativeViewPointDepthToImage();
}

void draw() {
  background(0);
  EToledo.update();
  //Carga la imagen de color del Kinect como un vector
  PImage rgbImage = EToledo.rgbImage();
  /*
```

Fuente: elaboración propia.

Figura 78. **Código del ejemplo 10, parte 3**



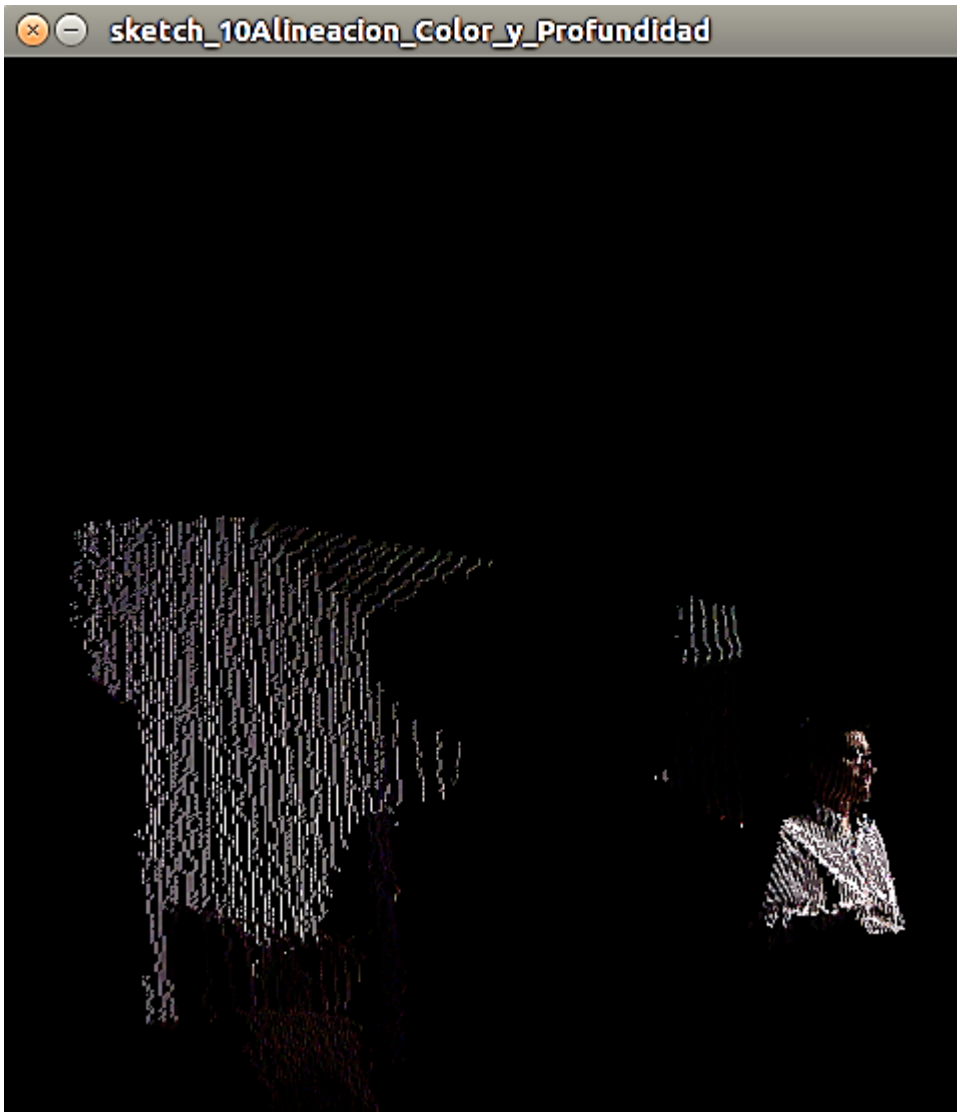
Fuente: elaboración propia.

Figura 79. **Código del ejemplo 10, parte 4**

```
  La funcion EToledo.depthMapRealWorld devuelve una matriz
  de PVectors, en la que almacenamos los depthPoints.
  Una vez que tenemos estos vectores en una matriz, el
  ciclo de esa matriz dibuja un solo punto en la pantalla para
  cada vector.
  */
  PVector[] depthPoints = EToledo.depthMapRealWorld();
  /*
  El cambio de i++ a i+=10
  permite un procesamiento más rápido ya que
  solo dibuja cada 10 puntos.
  */
  for (int i = 0; i < depthPoints.length; i+=20) {
    /*
    PVector es la función de procesamiento de los tres puntos (x,y,z) que
    extrae las coordenadas para dibujar los puntos con las coordenadas (x,y,z)
    */
    PVector currentPoint = depthPoints[i];
    //Accede al color del píxel RGB
    stroke(rgbImage.pixels[i]);
    point(currentPoint.x, currentPoint.y, currentPoint.z);
  }
}
```

Fuente: elaboración propia.

Figura 80. **Resultado del código del ejemplo 10**



Fuente: elaboración propia.

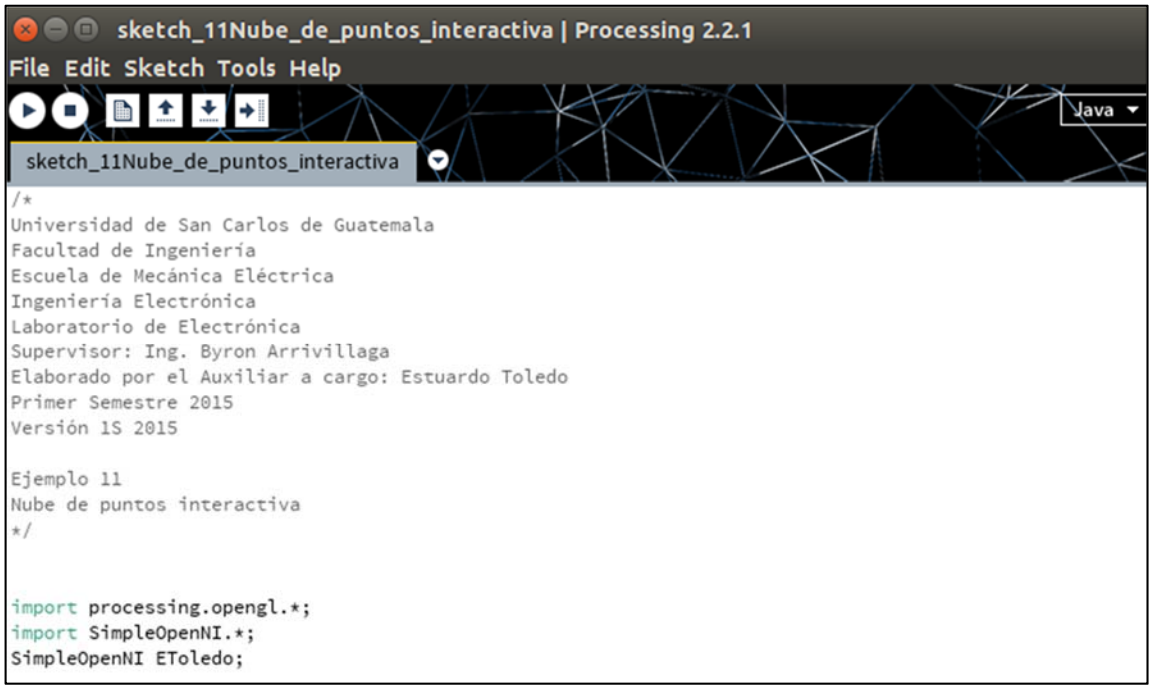
Al proporcionarle color a la nube de puntos mediante la alineación de la cámara de color y de profundidad, se logra obtener una imagen más sólida del boceto 3D.

4.7.5. Ejemplo 11, traslación y rotación de la nube de puntos

En este ejemplo se usará la posición del mouse para trasladar y rotar la nube de puntos en el espacio tridimensional. Acercar o alejar la nube de puntos mediante las flechas arriba y abajo del teclado y realizar una captura de pantalla al presionar el botón izquierdo del *mouse* sobre la pantalla y guardar la imagen.

Se podrán dibujar formas geométricas sobre la pantalla para empezar a interactuar con el usuario.

Figura 81. **Código del ejemplo 11, parte 1**



Fuente: elaboración propia.

Figura 82. **Código del ejemplo 11, parte 2**

```
/*
Variable para contener nuestra rotación actual
representada en grados
*/
float rotation = 0;
//Tamaño del cubo
int boxSize = 150;
//Vector con las coordenadas iniciales del cubo
PVector boxCenter = new PVector(0, 0, 600);
//Se usara como zoom, valor inicial
float s = 1;

void setup() {
  size(1024, 768, OPENGL);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
}
void draw() {
  background(0);
  EToledo.update();
  /*
  Trasladar la nube de puntos del origen (0,0,0) a
  (width/2, height/2, -1000)
  */
  translate(width/2, height/2, -1000);
  //Rotación vertical de la nube de puntos
  rotateX(radians(180));
  //Mantener la traslación, para que mejore la escala del centrado
  translate(0, 0, 1400);
  //Rotar al rededor del eje Y
  rotateY(radians(map(mouseX, 0, width, -180, 180)));
  //Zoom
  translate(0,0,s*-1000);
  scale(s);
  println(s);
  stroke(255);
```

Fuente: elaboración propia.

Figura 83. **Código del ejemplo 11, parte 3**

```
/*
depthMapRealWorld esta función nos dará los vectores que
necesitamos para dibujar puntos en tres dimensiones que
corresponden a la escena frente a la Kinect, usa una gran
cantidad de memoria para el procesamiento de datos.
Además de la organización de los datos de profundidad en
vectores, esta función procesa la posición de estos
vectores para eliminar la distorsión y proyectarlas
en el espacio 3D realista.

La función EToledo.depthMapRealWorld devuelve una matriz
de PVectors, en la que almacenamos los depthPoints.
Una vez que tenemos estos vectores en una matriz, el
ciclo de esa matriz dibuja un solo punto en la pantalla para
cada vector.
*/
PVector[] depthPoints = EToledo.depthMapRealWorld();
/*
Inicializa la variable para almacenar todos los puntos
que esten dentro de la caja.
*/
int depthPointsInBox = 0;
//solo dibuja cada un cada 50 puntos.
for (int i = 0; i < depthPoints.length; i+=50) {
  /*
  PVector es la función de procesamiento de los tres puntos (x,y,z) que
  extrae las coordenadas para dibujar los puntos con las coordenadas (x,y,z)
  */
  PVector currentPoint = depthPoints[i];
  //Limites del cubo
  if (currentPoint.x > boxCenter.x - boxSize/2 && currentPoint.x < boxCenter.x + boxSize/2) {
    if (currentPoint.y > boxCenter.y - boxSize/2 && currentPoint.y < boxCenter.y + boxSize/2) {
      if (currentPoint.z > boxCenter.z - boxSize/2 && currentPoint.z < boxCenter.z + boxSize/2) {
        depthPointsInBox++;
      }
    }
  }
  point(currentPoint.x, currentPoint.y, currentPoint.z);
}
```

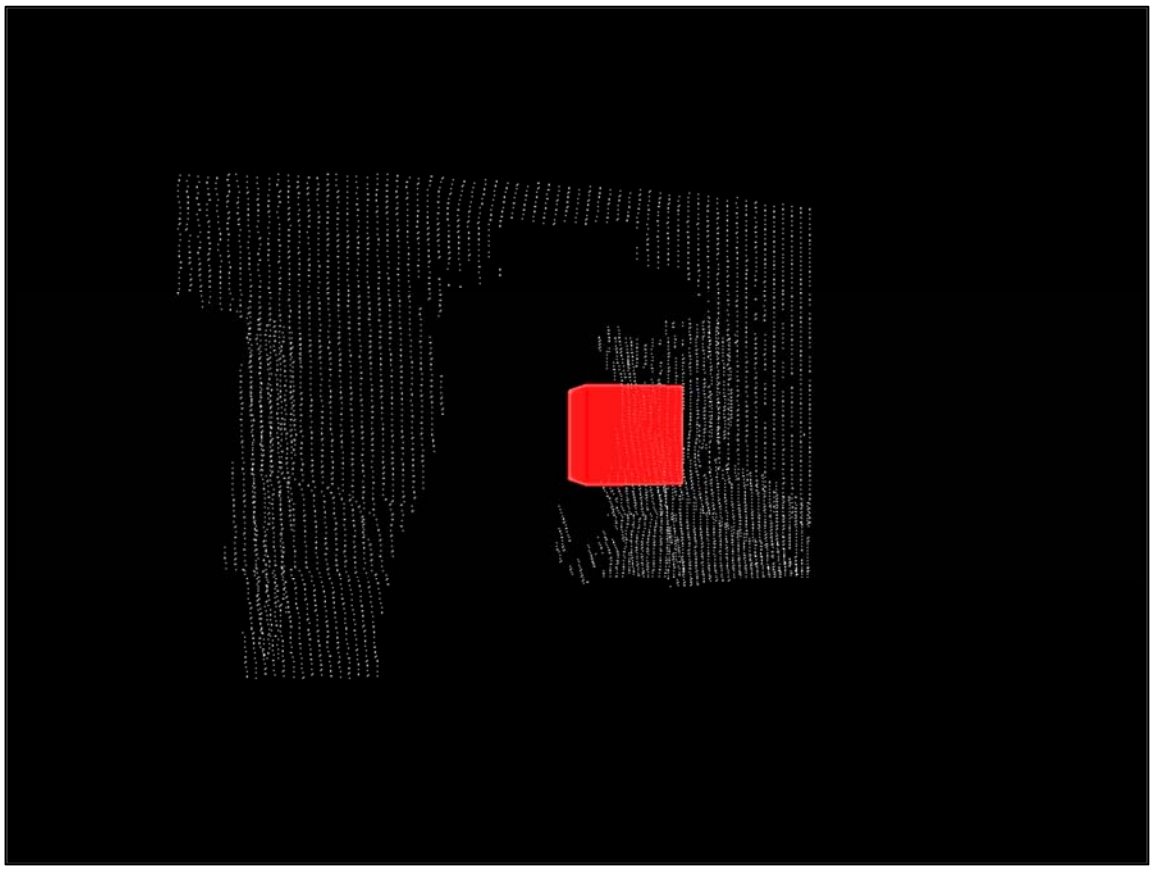
Fuente: elaboración propia.

Figura 84. **Código del ejemplo 11, parte 4**

```
  println(depthPointsInBox);
  //Configurar cubo: (0,Opaco,transparencia,255)
  float boxAlpha = map(depthPointsInBox,0,255,50,255);
  //Mover el cubo a donde lo necesitemos
  translate(boxCenter.x, boxCenter.y, boxCenter.z);
  //El cuarto elemento es para llenar alfa y determina la opacidad del color.
  fill(255, 0, 0, boxAlpha);
  //Linea de color rojo
  stroke(255, 0, 0);
  //Dibuja el cubo
  box(boxSize);
}

//Usar botones para controlar el zoom flecha arriba acerca y viceversa
void keyPressed(){
  if(keyCode == 38){
    s = s + 0.01;
  }
  if(keyCode == 40){
    s = s - 0.01;
  }
}
void mousePressed(){
  save("touchedPoint.png");
}
```

Fuente: elaboración propia.

Figura 85. **Resultado del código del ejemplo 11**



Fuente: elaboración propia.

Como se aprecia, cuántos más puntos de la nube de puntos se toquen con la mano, más opaco se tornará el cubo, con lo cual pueden realizarse diversas funciones con un solo objeto en el espacio tridimensional.

4.7.6. Ejemplo 12, objeto en 3D y librería saito.objloader

En este ejemplo se usará la librería saito.objloader para manipular el modelo 3D con extensión “.obj” y un fondo de pantalla para darle un mejor efecto y controlar el modelo por medio del teclado y el mouse. A continuación algunas de las geometrías con las que se forman dichos modelos.

Los modelos pueden realizarse directamente por los usuarios con programas de modelado 3D, como Blender o pueden ser descargados de diversas páginas de internet que ofrecen modelos e imágenes gratuitas.

Figura 86. **Código del ejemplo 12, parte 1**

Fuente: elaboración propia.

Figura 87. **Código del ejemplo 12, parte 2**

```
//Variables de rotación
float rotateX;
float rotateY;
//Variable para cargar el fondo
PImage fondo;
//Variable Zoom inicial
float zoom = 0.5;
//Declaración de un objeto
OBJModel model;
//Variable de dibujo de lineas
boolean estructura = false;

void setup() {
  size(1440, 720, OPENGL);
  //Cargar fondo
  fondo = loadImage("ven0aaa2.jpg");
  //Usa triángulos como la geometría básica para el modelo OBJ
  model = new OBJModel(this, "Arc-170ship/Arc_obj_Sh3d adapted/Arc170.obj", "relative", TRIANGLES);
  /*
  Opciones de geometría: POINTS, LINES, TRIANGLES, QUADS, POLYGON, TRIANGLE_STRIP, y QUAD_STRIP
  La opción de geometría mas usada en el modelado de objetos son los triángulos, por que tienen
  superficie y a partir de ellos se pueden formar polígonos
  */
  //Deshabilita las texturas del modelo
  model.disableTexture();
  //Escala inicial del modelo
  model.scale(zoom);
  //Deshabilita la verificación de errores del modelo
  model.disableDebug();
  //Traslada el modelo al centro
  model.translateToCenter();
  //Constructor del modelo
  noStroke();
}
```

Fuente: elaboración propia.

Figura 88. **Código del ejemplo 12, parte 3**

```
void draw() {
  background(fondo);
  //Proporciona el efecto de luces y sombras al modelo OBJ para hacerlo más real
  lights();
  //Trasladar al origen (0,0,0)
  translate(width/2, height/2, -200);
  //Rotación del modelo en función del mouse
  rotateX(rotateY);
  rotateY(rotateX);
  //Cambia la escala del modelo en función de la rueda del mouse
  scale(zoom);
  //Dibuja el modelo
  model.draw();
  println(zoom);
}

void mouseDragged(){
  /*
  mouseX y mouseX representan la posición actual del mouse
  pmouseX y pmouseY representan la posición de la cual se
  movió el mouse
  */
  rotateX += (mouseX - pmouseX) * 0.01;
  rotateY -= (mouseY - pmouseY) * 0.01;
}

//Usa la rueda del mouse para controlar la escala del modelo.
void mouseWheel(MouseEvent event){
  float e = event.getAmount();
  if(e > 0){
    zoom -= 0.1;
  }
  if(e < 0) {
    zoom += 0.1;
  }
}
```

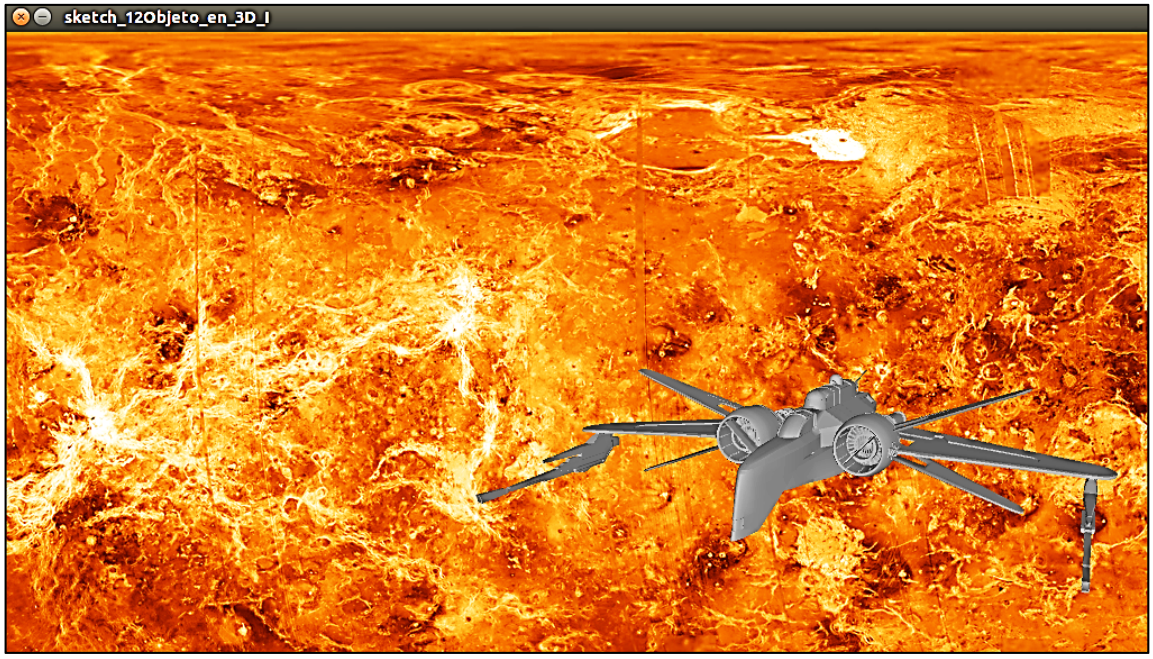
Fuente: elaboración propia.

Figura 89. **Código del ejemplo 12, parte 4**

```
void keyPressed(){ //tecla enter
  if(estructura){
    //shapeMode: cambia la geometría del modelo
    model.shapeMode(TRIANGLE_STRIP);
    stroke(0);
  }
  else {
    model.shapeMode(TRIANGLES);
    noStroke();
  }
  estructura = !estructura;
}
```

Fuente: elaboración propia.

Figura 90. **Resultado del código del ejemplo 12, parte 1**



Fuente: elaboración propia.

Como se observa, se controla la rotación y la escala de la nave con el botón y la rueda del *mouse*.

Figura 91. **Resultado del código del ejemplo 12, parte 2**

Fuente: elaboración propia.

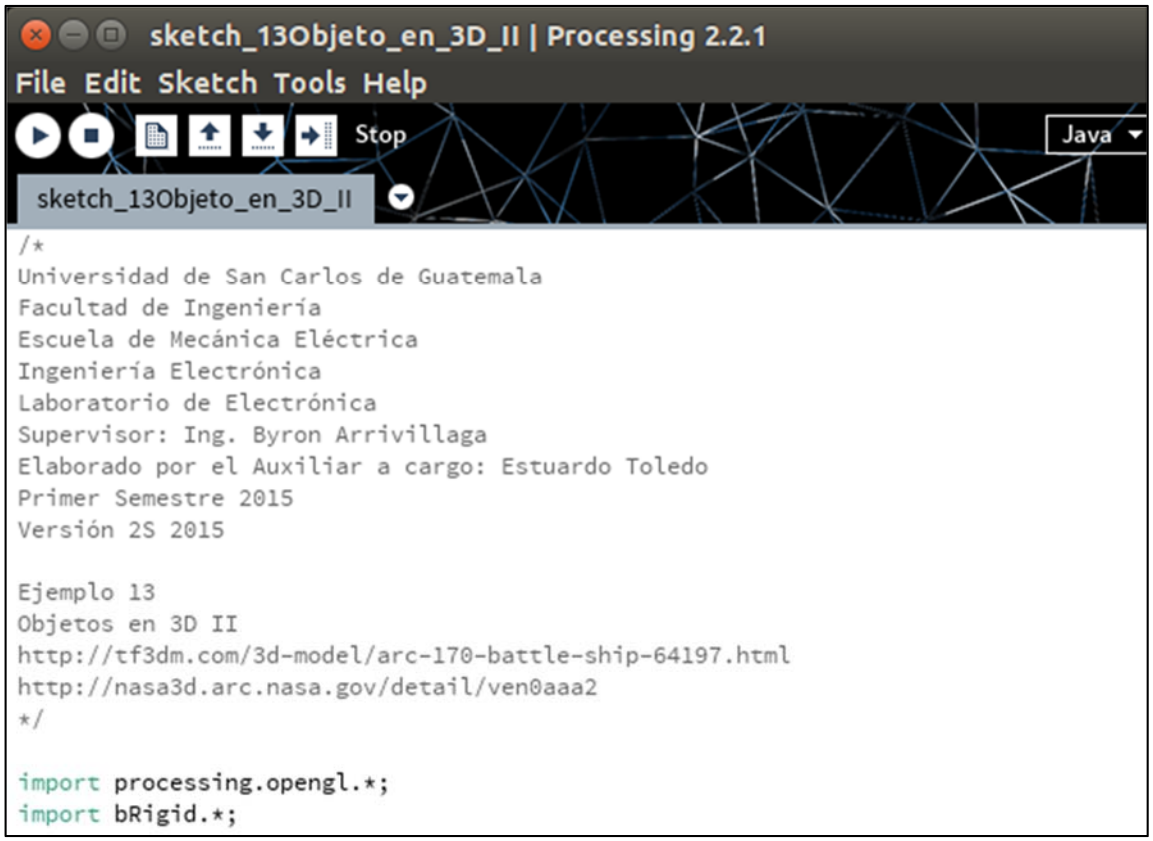
Obsérvese que al presionar la tecla enter del teclado se aprecia la geometría del modelo. La geometría que más se usa en los modelos 3D son los triángulos, porque tienen superficie y a partir de ellos se pueden formar polígonos.

El fondo de pantalla fue tomado de la página de la NASA 3D Resources y el modelo se obtuvo de la página TF3DM.

4.7.7. Ejemplo 13, objeto en 3D y librería bRigid

En este ejemplo se usa la librería bRigid para manipular el modelo 3D, con la diferencia de que, con esta librería puede cargarse el modelo con sus respectivos colores y controlarlo por medio del teclado y el *mouse*.

Figura 92. **Código del ejemplo 13, parte 1**



Fuente: elaboración propia.

Figura 93. **Código del ejemplo 13, parte 2**

```
//Variables de rotación
float rotateX;
float rotateY;
//Variable para cargar el fondo
PImage fondo;
//Variable Zoom inicial
float zoom = 0.5;
//Declaración del modelo
PShape model;

void setup() {
  //Dibujar en 3D
  size(1440, 720, P3D);
  //Cargar fondo
  fondo = loadImage("ven0aaa2.jpg");
  //Carga el modelo
  model = loadShape("Arc-170ship/Arc_obj_Sh3d adapted/Arc170.obj");
  //Escala inicial del modelo
  model.scale(zoom);
  //Rota el modelo para corregir su posición.
  model.rotateZ(22);
  //Constructor del modelo
  noStroke();
}
```

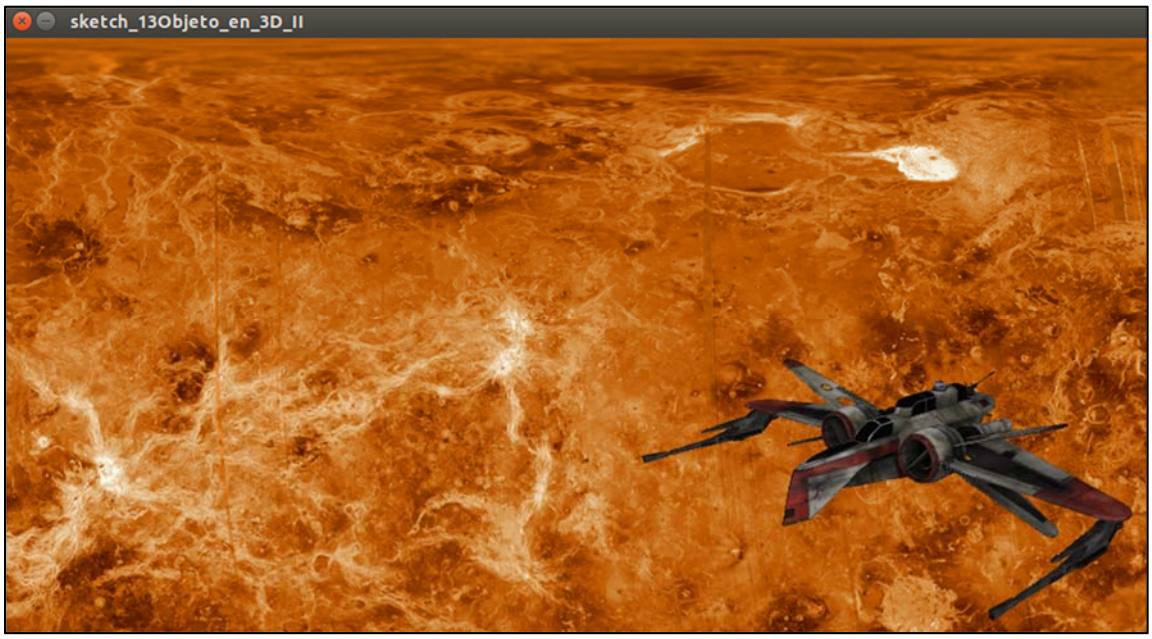
Fuente: elaboración propia.

Figura 94. **Código del ejemplo 13, parte 3**

```
void draw() {
  background(fondo);
  //Proporciona el efecto de luces y sombras al modelo OBJ para hacerlo más real
  lights();
  //Trasladar al origen (0,0,0)
  translate(width/2, height/2, -200);
  //Rotación del modelo en función del mouse
  rotateX(rotateY);
  rotateY(rotateX);
  //Cambia la escala del modelo en función de la rueda del mouse
  scale(zoom);
  //Dibuja el modelo
  shape(model);
  println("Zoom: ", zoom);
}

void mouseDragged(){
  /*
  mouseX y mouseX representan la posición actual del mouse
  pmouseX y pmouseY representan la posición de la cual se
  movió el mouse
  */
  rotateX += (mouseX - pmouseX) * 0.01;
  rotateY -= (mouseY - pmouseY) * 0.01;
}

//Usa la rueda del mouse para controlar la escala del modelo.
void mouseWheel(MouseEvent event){
  float e = event.getAmount();
  if(e > 0){
    zoom -= 0.1;
  }
  if(e < 0) {
    zoom += 0.1;
  }
}
```

Fuente: elaboración propia.

Figura 95. **Resultado del código del ejemplo 13**



Fuente: elaboración propia.

Como se aprecia en la imagen, se manipula el objeto 3D con su respectivo color.

El fondo de pantalla fue tomado de la página de la NASA 3D Resources y el modelo se obtuvo de la página TF3DM.

4.7.8. Ejemplo 14, objeto en 3D y Kinect

En este ejemplo se usa la librería *processing*.serial para enviar datos por el puerto serial a la launchpad MSP430 de Texas Instruments para escuchar una melodía o encender y apagar unos *leds* bajo el control de los *hot point* que se seleccionan con las manos.

Para poner en funcionamiento este ejemplo se hace una serie de configuraciones.

Primero: instalar el gestor de *bluetooth* de Ubuntu.

Figura 96. **Gestor de *bluetooth* de Ubuntu**

Fuente: elaboración propia.

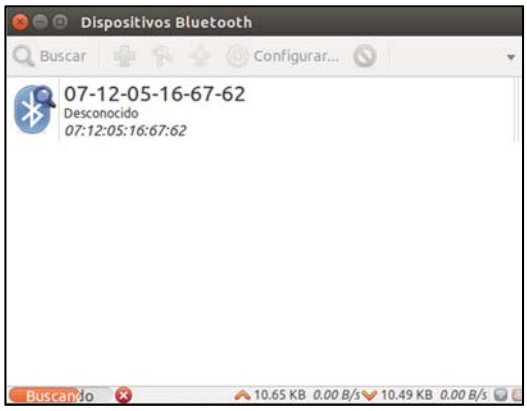
Segundo: cuando se encuentre instalado el gestor de *bluetooth* se guarda el código del microcontrolador MSP430 en el dispositivo y se conecta toda la circuitería incluyendo el módulo de *bluetooth* HC-06, que se debe desconectar y conectar cada vez que se programa el microcontrolador.

Figura 97. **Módulo *bluetooth* HC-06**

Fuente: NEOTEO. www.neoteo.com/módulo-bluetooth-hc-06-android. Consulta: 1 de diciembre de 2015.

Tercero: emparejar el módulo de *bluetooth* del microcontrolador con el ordenador, para lo cual se busca un nuevo dispositivo en el gestor de *bluetooth*. Se ingresa la clave de acceso al dispositivo, en la mayoría viene por defecto la clave “1234” y con esto, el dispositivo está igualado. Debe recordarse a que puerto COM”X” se conectó el módulo. Si el dispositivo está igualado se notará que el led de color rojo dejará de parpadear.

Figura 98. **Emparejar el módulo *bluetooth*, parte 1**



Fuente: elaboración propia.

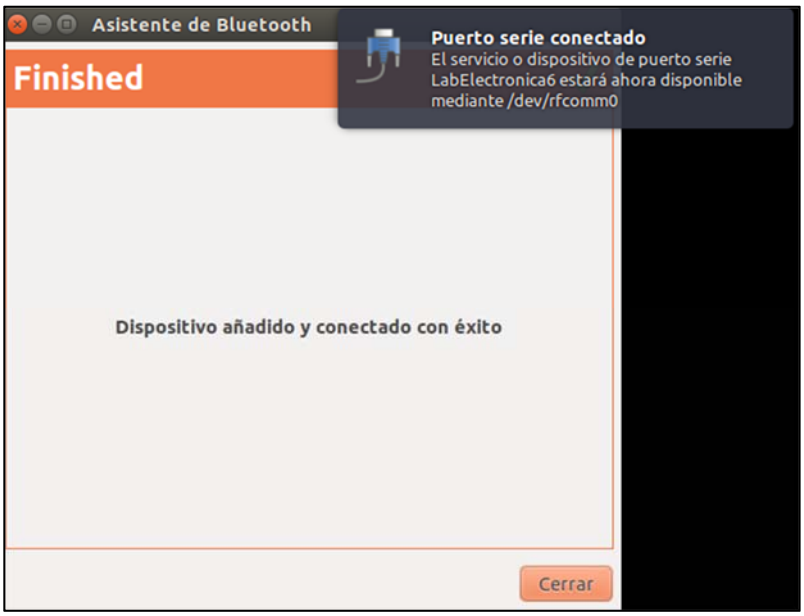
Figura 99. **Emparejar el módulo *bluetooth*, parte 2**

Fuente: elaboración propia.

Figura 100. **Emparejar el módulo *bluetooth*, parte 3**

Fuente: elaboración propia.

Figura 101. **Emparejar el módulo *bluetooth*, parte 4**



Fuente: elaboración propia.

Cuarto: abrir el puerto serial, ya que por defecto Linux los tiene cerrados, por lo que se escriben unos comandos en la terminal de Linux:

Verificar que exista el dispositivo rfcomm0.

- ls /dev/rfcomm0

Abrir el puerto para el dispositivo rfcomm0

- sudo chown rketoledo /dev/rfcomm0

Por último iniciar el boceto para controlar el microcontrolador por medio del Kinect.

Figura 102. **Código del ejemplo 14, parte 1 *processing***

Fuente: elaboración propia.

Figura 103. **Código del ejemplo 14, parte 2 *processing***

```
PImage fondo;
//Variable escala inicial del modelo
float escala = 0.5;
//Declaración del modelo
PShape model;
//-------------------------------------------------Pantalla
//Variable de rotación de la pantalla
float rotation = 0;
//Se usara como zoom de la pantalla
float s = 1;
//-------------------------------------------------Cubo1 Giro +X
//Tamaño del cubo 1
int boxSize1 = 150;
//Vector con las coordenadas iniciales del cubo
PVector boxCenter1 = new PVector(0, 0, 800);
//-------------------------------------------------Cubo2 Giro -X
//Tamaño del cubo 2
int boxSize2 = 150;
//Vector con las coordenadas iniciales del cubo
PVector boxCenter2 = new PVector(300, 0, 800);
//-------------------------------------------------Cubo3 Giro +Y
//Tamaño del cubo 3
int boxSize3 = 150;
//Vector con las coordenadas iniciales del cubo
PVector boxCenter3 = new PVector(150, 200, 800);
//-------------------------------------------------Cubo4 Giro -Y
//Tamaño del cubo 4
int boxSize4 = 150;
//Vector con las coordenadas iniciales del cubo
PVector boxCenter4 = new PVector(150, -200, 800);
//-------------------------------------------------CuboS Escala +
//Tamaño del cubo 5
int boxSize5 = 150;
//Vector con las coordenadas iniciales del cubo
PVector boxCenter5 = new PVector(-300, 200, 800);
//-------------------------------------------------Cubo6 Escala -
//Tamaño del cubo 6
int boxSize6 = 150;
//Vector con las coordenadas iniciales del cubo
```

Fuente: elaboración propia.

Figura 104. **Código del ejemplo 14, parte 3 *processing***


```
PVector boxCenter6 = new PVector(-300, -200, 800);
//-------------------------------------------------Cubo7 Escala = 0.5 y Stop
//Tamaño del cubo 7
int boxSize7 = 150;
//Vector con las coordenadas iniciales del cubo
PVector boxCenter7 = new PVector(-300, 0, 800);
//-------------------------------------------------Delay
int Ti = 0;
int To = 0;
int Modulo = 0;
int Time = 200;

void setup() {
  //Dibujar en 3D
  size(1440, 720, P3D);
  //Cargar fondo
  fondo = loadImage("ven0aaa2.jpg");
  //Carga el modelo
  //-------------------------------------------------Modelo
  model = loadShape("Arc-170ship/Arc_obj_Sh3d adapted/Arc170.obj");
  //escala inicial del modelo
  model.scale(escala);
  //Rota el modelo para corregir su posición.
  model.rotateZ(22);
  //Constructor del modelo
  noStroke();
  //-------------------------------------------------Kinect
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  //-------------------------------------------------Puerto Serial
  println(Serial.list());
  String portName = Serial.list()[0];
  port = new Serial(this, portName, 9600);
  port.buffer(100);
  //-------------------------------------------------Tiempo inicial
  Ti = millis();
}
```

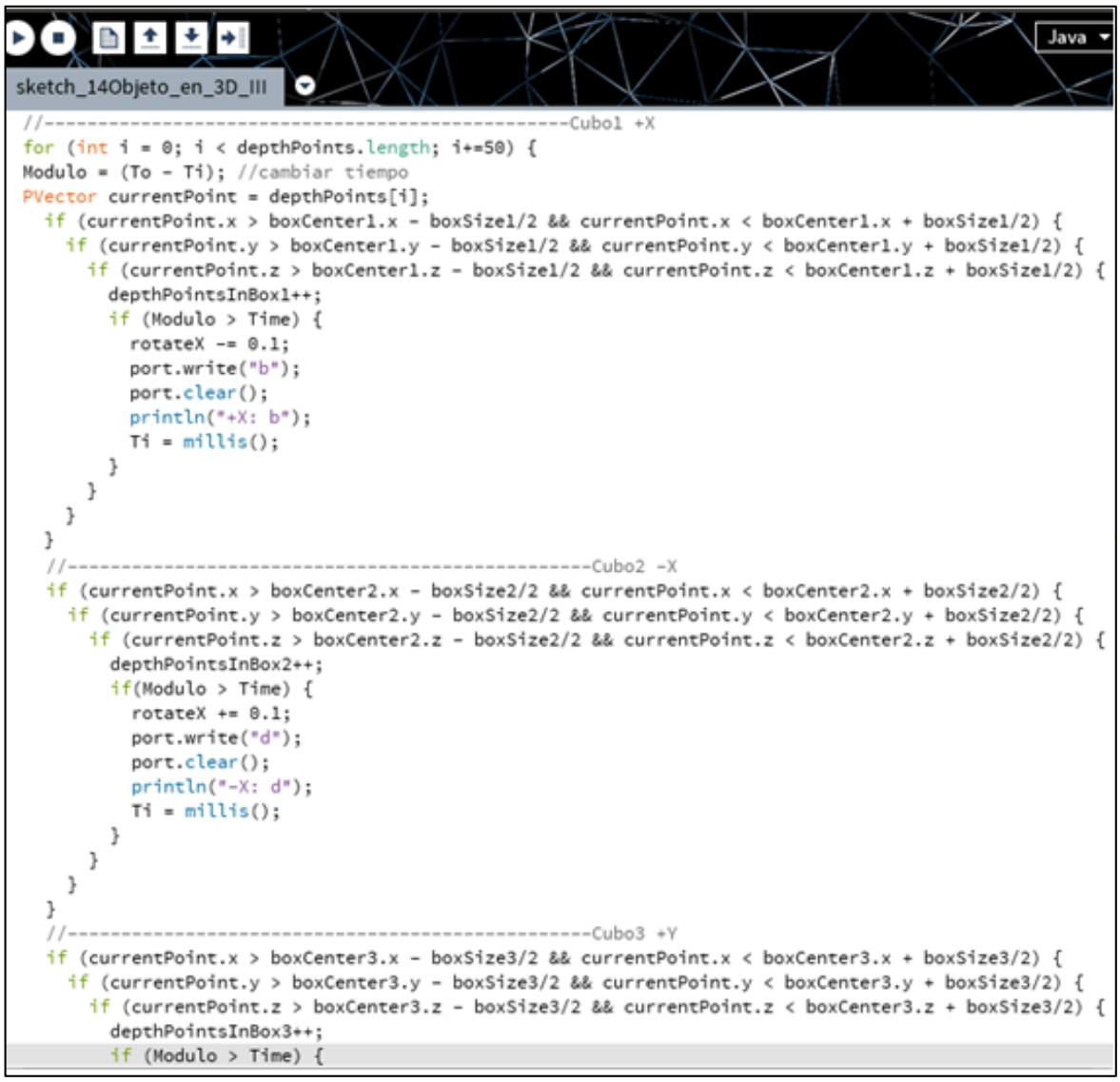
Fuente: elaboración propia.

Figura 105. **Código del ejemplo 14, parte 4 *processing***


```
void draw() {
  pushMatrix();
  background(0);
  //-------------------------------------------------Modelo
  //Proporciona el efecto de luces y sombras al modelo OBJ para hacerlo más real
  lights();
  //Traslada el modelo (X,Y,Z)
  translate(width/2, height/2, -100);
  //Rotación del modelo en función del mouse
  rotateX(rotateY);
  rotateY(rotateX);
  //Cambia la escala del modelo en función de la rueda del mouse
  scale(escala);
  println("Escala del modelo: ", escala);
  //Dibuja el modelo
  shape(model);
  popMatrix();
  //-------------------------------------------------Kinect
  EToledo.update();
  //Trasladar la nube de puntos del origen (X,Y,Z) a (width/2, height/2, -1000)
  translate(3000, 1000, -1000);
  //Rotación vertical de la nube de puntos
  rotateX(radians(180));
  //Mantener la traslación, para que mejore la escala del centrado
  translate(0, 0, 1400);
  //Rotar al rededor del eje Y
  rotateY(radians(map(mouseX, 0, width, -180, 180)));
  //Zoom de la pantalla
  translate(0,0,s*-1000);
  scale(s);
  println("Zoom de la pantalla: ", s);
  PVector[] depthPoints = EToledo.depthMapRealWorld();
  int depthPointsInBox1 = 0;
  int depthPointsInBox2 = 0;
  int depthPointsInBox3 = 0;
  int depthPointsInBox4 = 0;
  int depthPointsInBox5 = 0;
  int depthPointsInBox6 = 0;
  int depthPointsInBox7 = 0;
```

Fuente: elaboración propia.

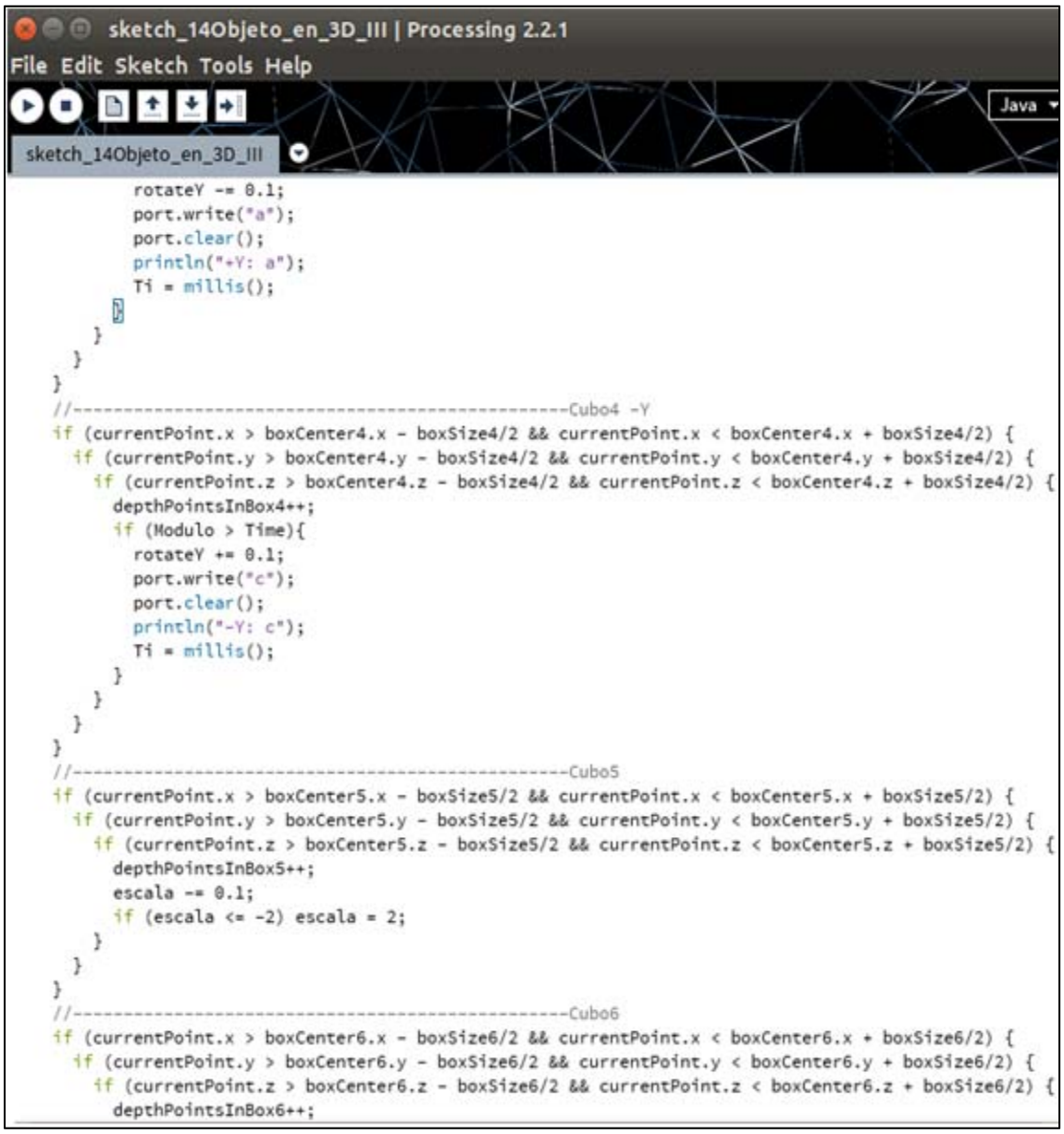
Figura 106. **Código del ejemplo 14, parte 5 *processing***




```
//-----------------------------------------------Cubo1 +X
for (int i = 0; i < depthPoints.length; i+=50) {
Modulo = (To - Ti); //cambiar tiempo
PVector currentPoint = depthPoints[i];
  if (currentPoint.x > boxCenter1.x - boxSize1/2 && currentPoint.x < boxCenter1.x + boxSize1/2) {
    if (currentPoint.y > boxCenter1.y - boxSize1/2 && currentPoint.y < boxCenter1.y + boxSize1/2) {
      if (currentPoint.z > boxCenter1.z - boxSize1/2 && currentPoint.z < boxCenter1.z + boxSize1/2) {
        depthPointsInBox1++;
        if (Modulo > Time) {
          rotateX -= 0.1;
          port.write("b");
          port.clear();
          println("+X: b");
          Ti = millis();
        }
      }
    }
  }
  //-----------------------------------------------Cubo2 -X
  if (currentPoint.x > boxCenter2.x - boxSize2/2 && currentPoint.x < boxCenter2.x + boxSize2/2) {
    if (currentPoint.y > boxCenter2.y - boxSize2/2 && currentPoint.y < boxCenter2.y + boxSize2/2) {
      if (currentPoint.z > boxCenter2.z - boxSize2/2 && currentPoint.z < boxCenter2.z + boxSize2/2) {
        depthPointsInBox2++;
        if(Modulo > Time) {
          rotateX += 0.1;
          port.write("d");
          port.clear();
          println("-X: d");
          Ti = millis();
        }
      }
    }
  }
  //-----------------------------------------------Cubo3 +Y
  if (currentPoint.x > boxCenter3.x - boxSize3/2 && currentPoint.x < boxCenter3.x + boxSize3/2) {
    if (currentPoint.y > boxCenter3.y - boxSize3/2 && currentPoint.y < boxCenter3.y + boxSize3/2) {
      if (currentPoint.z > boxCenter3.z - boxSize3/2 && currentPoint.z < boxCenter3.z + boxSize3/2) {
        depthPointsInBox3++;
        if (Modulo > Time) {
```

Fuente: elaboración propia.

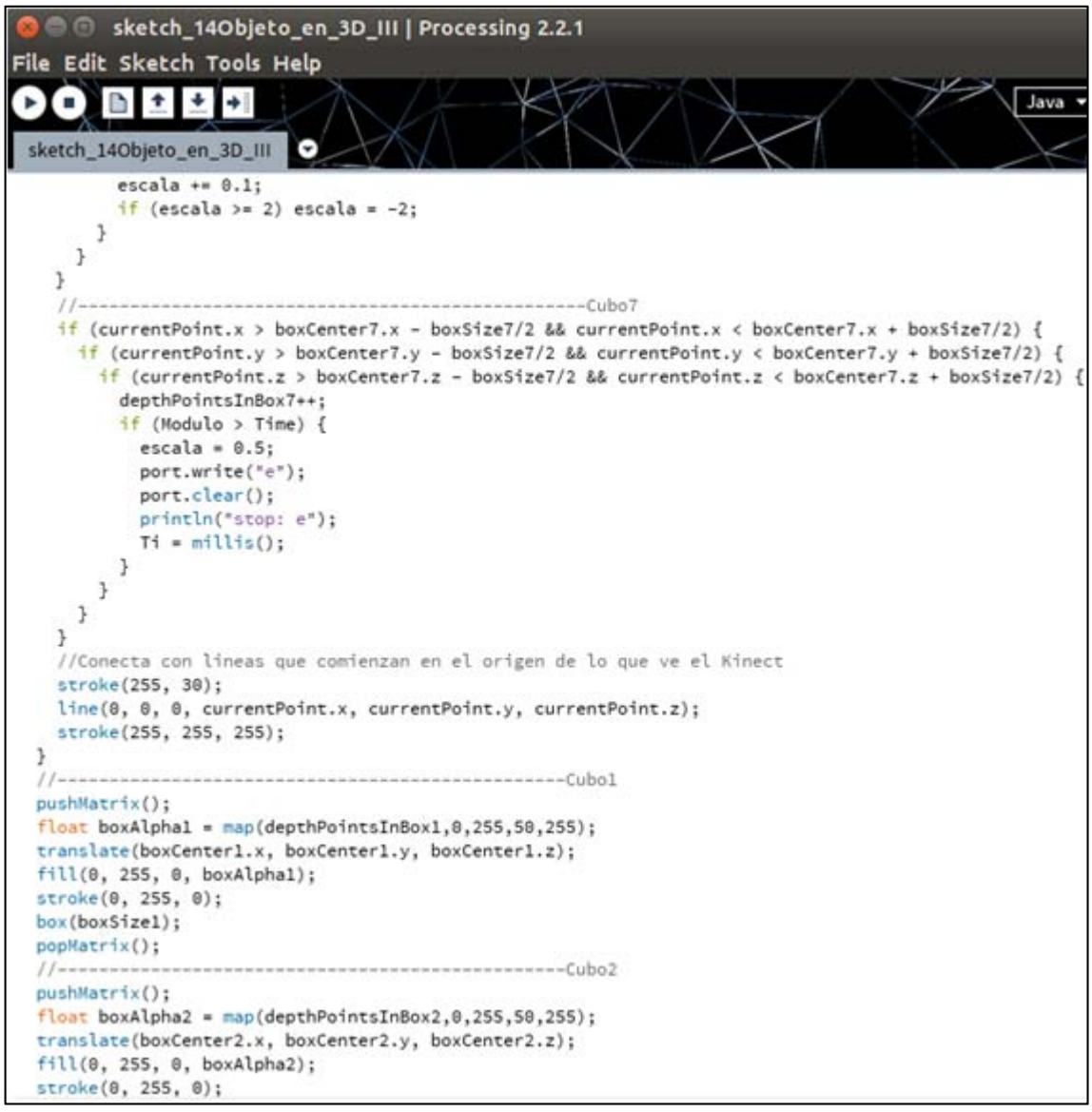
Figura 107. **Código del ejemplo 14, parte 6 *processing***




```
          rotateY -= 0.1;
          port.write("a");
          port.clear();
          println("+Y: a");
          Ti = millis();
        }
      }
    }
  }
  //-------------------------------------------------Cubo4 -Y
  if (currentPoint.x > boxCenter4.x - boxSize4/2 && currentPoint.x < boxCenter4.x + boxSize4/2) {
    if (currentPoint.y > boxCenter4.y - boxSize4/2 && currentPoint.y < boxCenter4.y + boxSize4/2) {
      if (currentPoint.z > boxCenter4.z - boxSize4/2 && currentPoint.z < boxCenter4.z + boxSize4/2) {
        depthPointsInBox4++;
        if (Modulo > Time){
          rotateY += 0.1;
          port.write("c");
          port.clear();
          println("-Y: c");
          Ti = millis();
        }
      }
    }
  }
  //-------------------------------------------------Cubo5
  if (currentPoint.x > boxCenter5.x - boxSize5/2 && currentPoint.x < boxCenter5.x + boxSize5/2) {
    if (currentPoint.y > boxCenter5.y - boxSize5/2 && currentPoint.y < boxCenter5.y + boxSize5/2) {
      if (currentPoint.z > boxCenter5.z - boxSize5/2 && currentPoint.z < boxCenter5.z + boxSize5/2) {
        depthPointsInBox5++;
        escala -= 0.1;
        if (escala <= -2) escala = 2;
      }
    }
  }
  //-------------------------------------------------Cubo6
  if (currentPoint.x > boxCenter6.x - boxSize6/2 && currentPoint.x < boxCenter6.x + boxSize6/2) {
    if (currentPoint.y > boxCenter6.y - boxSize6/2 && currentPoint.y < boxCenter6.y + boxSize6/2) {
      if (currentPoint.z > boxCenter6.z - boxSize6/2 && currentPoint.z < boxCenter6.z + boxSize6/2) {
        depthPointsInBox6++;
```

Fuente: elaboración propia.

Figura 108. **Código del ejemplo 14, parte 7 *processing***




```
        escala += 0.1;
        if (escala >= 2) escala = -2;
      }
    }
  }
  //-------------------------------------------------Cubo7
  if (currentPoint.x > boxCenter7.x - boxSize7/2 && currentPoint.x < boxCenter7.x + boxSize7/2) {
    if (currentPoint.y > boxCenter7.y - boxSize7/2 && currentPoint.y < boxCenter7.y + boxSize7/2) {
      if (currentPoint.z > boxCenter7.z - boxSize7/2 && currentPoint.z < boxCenter7.z + boxSize7/2) {
        depthPointsInBox7++;
        if (Modulo > Time) {
          escala = 0.5;
          port.write("e");
          port.clear();
          println("stop: e");
          Ti = millis();
        }
      }
    }
  }
  //Conecta con lineas que comienzan en el origen de lo que ve el Kinect
  stroke(255, 30);
  line(0, 0, 0, currentPoint.x, currentPoint.y, currentPoint.z);
  stroke(255, 255, 255);
}
//-------------------------------------------------Cubo1
pushMatrix();
float boxAlpha1 = map(depthPointsInBox1,0,255,50,255);
translate(boxCenter1.x, boxCenter1.y, boxCenter1.z);
fill(0, 255, 0, boxAlpha1);
stroke(0, 255, 0);
box(boxSize1);
popMatrix();
//-------------------------------------------------Cubo2
pushMatrix();
float boxAlpha2 = map(depthPointsInBox2,0,255,50,255);
translate(boxCenter2.x, boxCenter2.y, boxCenter2.z);
fill(0, 255, 0, boxAlpha2);
stroke(0, 255, 0);
```

Fuente: elaboración propia.

Figura 109. **Código del ejemplo 14, parte 8 *processing***

```
  box(boxSize2);
  popMatrix();
  //-------------------------------------------------Cubo3
  pushMatrix();
  float boxAlpha3 = map(depthPointsInBox3,0,255,50,255);
  translate(boxCenter3.x, boxCenter3.y, boxCenter3.z);
  fill(0, 255, 0, boxAlpha3);
  stroke(0, 255, 0);
  box(boxSize3);
  popMatrix();
  //-------------------------------------------------Cubo4
  pushMatrix();
  float boxAlpha4 = map(depthPointsInBox4,0,255,50,255);
  translate(boxCenter4.x, boxCenter4.y, boxCenter4.z);
  fill(0, 255, 0, boxAlpha4);
  stroke(0, 255, 0);
  box(boxSize4);
  popMatrix();
  //-------------------------------------------------Cubo5
  pushMatrix();
  float boxAlpha5 = map(depthPointsInBox5,0,255,50,255);
  translate(boxCenter5.x, boxCenter5.y, boxCenter5.z);
  fill(0, 255, 0, boxAlpha5);
  stroke(0, 255, 0);
  box(boxSize5);
  popMatrix();
  //-------------------------------------------------Cubo6
  pushMatrix();
  float boxAlpha6 = map(depthPointsInBox6,0,255,50,255);
  translate(boxCenter6.x, boxCenter6.y, boxCenter6.z);
  fill(0, 255, 0, boxAlpha6);
  stroke(0, 255, 0);
  box(boxSize6);
  popMatrix();
  //-------------------------------------------------Cubo7
  pushMatrix();
  float boxAlpha7 = map(depthPointsInBox7,0,255,50,255);
  translate(boxCenter7.x, boxCenter7.y, boxCenter7.z);
  fill(0, 255, 0, boxAlpha7);
  stroke(0, 255, 0);
  box(boxSize7);
  popMatrix();
  //Tiempo final
  To = millis();
}
```

Fuente: elaboración propia.

Figura 110. **Código del ejemplo 14, parte 9 *processing***

```
//Usar botones para controlar el zoom de la pantalla flecha arriba acerca y viceversa
void keyPressed(){
  if(keyCode == 38){
    s = s + 0.01;
  }
  if(keyCode == 40){
    s = s - 0.01;
  }
}
```

Fuente: elaboración propia.

Figura 111. **Código del ejemplo 14 parte 1, *code composer studio***

main.c ×

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Laboratorio de Microcontroladores/PIC's
Versión 1S 2014
Elaborado por: Aux. Estuardo Toledo
MSP430G2553
Ejemplo 9
Ejecución de la CPU en el DCO sin cristal
Sonido
Cargar archivos con codigo extra

*/

#include <msp430.h>
#include "melody1.h"
```

Fuente: elaboración propia.

Figura 112. **Código del ejemplo 14 parte 2, *code composer studio***

```
#ifndef TIMER0_A1_VECTOR
#define TIMER0_A1_VECTOR    TIMERA1_VECTOR
#define TIMER0_A0_VECTOR    TIMERA0_VECTOR
#endif

void PuertosES(void);
void Oscilador(void);
void Error(void);
void UART(void);

/*
 * main.c
 */
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
    PuertosES();
    Oscilador();
    UART();
        //return 0;
}
void PuertosES(void) {
        P1SEL &=~BIT0;  //Bit 0 del puerto 1 como S
        P1DIR |= BIT0;  //Bit 0 del puerto 1 como S
        P1OUT &=~BIT0;  //Puerto 1 bit 0 apagado

        P2SEL = 0;              //8 Bits puerto 2 como S
        P2DIR = 255;    //8 Bits puerto 2 como S
        P2OUT = 0;    //Puerto 2 apagado
}
 void Oscilador(void) {
       if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFE)
            Error();
          //Si el dato de CAL es borrado corre la rutina de error

       BCSCTL1 = CALBC1_1MHZ;  //Define el rango
       DCOCTL = CALDCO_1MHZ;   //Defiene el paso del DCO y la modulacion

       BCSCTL3 |= LFXT1S_2;    //Reloj del sistema
       IFG1 &= ~OFIFG;         //Limpia la bandera de OSCFault
       BCSCTL2 &=~ SELS; // Selecciona DCOCLk como la funete de SMCLK
 }
```

Fuente: elaboración propia.

Figura 113. **Código del ejemplo 14 parte 3, *code composer studio***

```
void Error(void) {
   P2OUT = 255; //Enciende leds
   while(1);     //Queda atrapado
}
void UART(void) {
        P1SEL |= (BIT1 + BIT2);         // P1.1 = RXD, P1.2=TXD (Configuracion de puertos)
        P1SEL2 |= (BIT1 + BIT2);        // P1.1 = RXD, P1.2=TXD
        UCA0CTL1 |= UCSSEL_2 + UCSWRST;
        /*
        UCA0CTL1: Registro de control USCI
        UCSSEL: Seleccion de fuente de reloj (UCSSEL_2: SMCLK)
        UCSWRST: Resetea el modulo para configurar.
        */
        /*
        Los siguientes registros configuran la velocidad de transmisión:
        9600 baud con 1 MHZ.
        Las configuraciones posibles están listadas en el userguide, página 435.
        */
        UCA0BR0 = 104; //Registro de control 0 de Baud Rate: 1MHz 9600, UCBRx = 104
        UCA0BR1 = 0; //Registro de control 1 de Baud Rate: 1MHz 9600, UCBRFx = 0
        UCA0MCTL = UCBRS0; // Registro de control de la modulacion: UCBRSx = 1
        UCA0CTL1 &= ~UCSWRST; //Finaliza el reseteo para configurar

        IE2 |= UCA0RXIE;        //Havilita la interrupcion de RX
        _BIS_SR(GIE);           //Interrupciones Globales
        _enable_interrupts();
         /*
          *Habilitación general de interrupciones del micro. Esta macro genera la
          *instrucción EINT, que pone a 1 el bit GIE (global interrupt enable) del
          *registro SR (status register)
         */
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
    //while (!(IFG2&UCA0TXIFG));                // USCI_A0 TX esta listo?
    //UCA0TXBUF = UCA0RXBUF;                    // TX -> RXed character
        if(UCA0RXBUF == 'e'){   //Leer caracter
                play();
        }
        if(UCA0RXBUF == 'b'){   //Leer caracter
                P2OUT = 1;
        }
        if(UCA0RXBUF == 'd'){   //Leer caracter
                P2OUT = 10;
        }
        if(UCA0RXBUF == 'a'){   //Leer caracter
                P2OUT = 2;
        }
        if(UCA0RXBUF == 'c'){   //Leer caracter
                P2OUT = 20;
        }
}
```

Fuente: elaboración propia.

Figura 114. **Código del ejemplo 14 parte 4, *code composer studio***

melody1.h ×

```
/*
 * melody1.h
 */

#ifndef MELODY1_H_
#define MELODY1_H_


//Definicion de notas y frequecies en Hertz.
#define E3 3033
#define Fs3 2703
#define G3 2551
#define Gs3 2408
#define A3 2273
#define As3 2145
#define B3 2025
#define Cb4
#define C4 1911
#define Cs4 1804
#define Db4 1804
#define D4 1703
#define Ds4 1607
#define Eb4 1607
#define E4 1517
#define F4 1432
#define Fs4 1351
#define Gb4 1351
#define G4 1276
#define Gs4 1204
#define Ab4 1204
#define A4 1136
#define As4 1073
#define Bb4 1073
#define B4 1012
#define C5 956
#define Cs5 902
#define Db5 902
#define D5 851
#define Ds5 804
#define Eb5 804
#define E5 758
```

Fuente: elaboración propia.

Figura 115. **Código del ejemplo 14 parte 5, *code composer studio***

melody1.h ×

```
#define F5 716
#define Fs5 676
#define Gb5 676
#define G5 638
#define Gs5 602
#define Ab5 602
#define A5 568

//Esta funcion detiene los tonos en un cierto número de milisegundos
void delay_ms(unsigned int ms )
{
    unsigned int i;
    for (i = 0; i<= ms; i++)
       __delay_cycles(500); //Built-in function that suspends the execution for 500 cicles
}


//Esta funcion detiene los tonos en un cierto número de microsegundos
void delay_us(unsigned int us )
{
    unsigned int i;
    for (i = 0; i<= us/2; i++)
       __delay_cycles(1);
}


//Esta función genera la onda cuadrada para el sonido del altavoz piezoeléctrico
//a una frecuencia determinada
void beep(unsigned int note, unsigned int duration)
{
    int i;
    long delay = (long)(10000/note);  //Semiperiodo para cada nota
    long time = (long)((duration*100)/(delay*2));  //Tiempo de pausa para cada nota
    for (i=0;i<time;i++)
    {
        P1OUT |= BIT0;     //Sonido por P1.1
        delay_us(delay);   //Semiperiodo
        P1OUT &= ~BIT0;    //Apaga el sonido
        delay_us(delay);   //Semiperiodo
    }
    delay_ms(20); //Para separar las notas individuales
}
```

Fuente: elaboración propia.

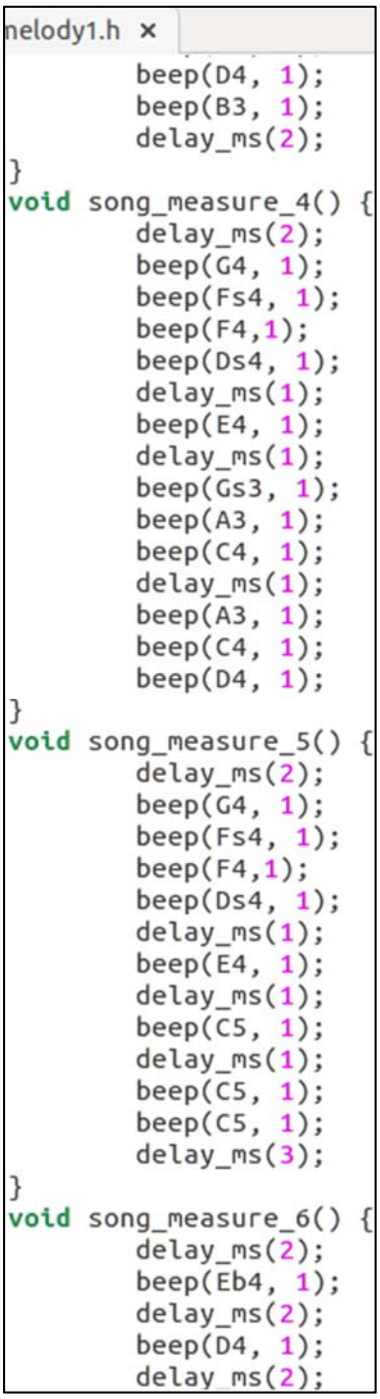
Figura 116. **Código del ejemplo 14 parte 6, *code composer studio***

melody1.h ×

```
void song_measure_1() {
        beep(E4, 1);
        beep(E4, 1);
        delay_ms(1);
        beep(E4, 1);
        delay_ms(1);
        beep(C4, 1);
        beep(E4, 1);
        delay_ms(1);
        beep(G4, 1);
        delay_ms(3);
        beep(G3, 1);
        delay_ms(3);
}
void song_measure_2() {
        beep(C4, 1);
        delay_ms(2);
        beep(G3, 1);
        delay_ms(2);
        beep(E3, 1);
        delay_ms(2);
        beep(A3, 1);
        delay_ms(1);
        beep(B3, 1);
        delay_ms(1);
        beep(As3, 1);
        beep(A3, 1);
        delay_ms(1);
}
void song_measure_3() {
        beep(G3, 1);
        beep(E4, 1);
        delay_ms(1);
        beep(G4, 1);
        beep(A4, 1);
        delay_ms(1);
        beep(F4, 1);
        beep(G4, 1);
        delay_ms(1);
        beep(E4, 1);
        delay_ms(1);
        beep(C4, 1);
```

Fuente: elaboración propia.

Figura 117. **Código del ejemplo 14 parte 7, *code composer studio***



```
nelody1.h ×
        beep(D4, 1);
        beep(B3, 1);
        delay_ms(2);
}
void song_measure_4() {
        delay_ms(2);
        beep(G4, 1);
        beep(Fs4, 1);
        beep(F4,1);
        beep(Ds4, 1);
        delay_ms(1);
        beep(E4, 1);
        delay_ms(1);
        beep(Gs3, 1);
        beep(A3, 1);
        beep(C4, 1);
        delay_ms(1);
        beep(A3, 1);
        beep(C4, 1);
        beep(D4, 1);
}
void song_measure_5() {
        delay_ms(2);
        beep(G4, 1);
        beep(Fs4, 1);
        beep(F4,1);
        beep(Ds4, 1);
        delay_ms(1);
        beep(E4, 1);
        delay_ms(1);
        beep(C5, 1);
        delay_ms(1);
        beep(C5, 1);
        beep(C5, 1);
        delay_ms(3);
}
void song_measure_6() {
        delay_ms(2);
        beep(Eb4, 1);
        delay_ms(2);
        beep(D4, 1);
        delay_ms(2);
```

Fuente: elaboración propia.

Figura 118. **Código del ejemplo 14 parte 8, *code composer studio***

melody1.h ×

```
        beep(C4, 1);
        delay_ms(7);
}
void song_measure_7() {
        beep(C4, 1);
        beep(C4, 1);
        delay_ms(1);
        beep(C4, 1);
        delay_ms(1);
        beep(C4, 1);
        beep(D4, 1);
        delay_ms(1);
        beep(E4, 1);
        beep(C4, 1);
        delay_ms(1);
        beep(A3, 1);
        beep(G3, 1);
        delay_ms(3);
}
void song_measure_8() {
        beep(C4, 1);
        beep(C4, 1);
        delay_ms(1);
        beep(C4, 1);
        delay_ms(1);
        beep(C4, 1);
        beep(D4, 1);
        beep(E4,1);
        delay_ms(7);
}
void play(){
        song_measure_1();
        song_measure_2();
        song_measure_3();
        song_measure_2();
        song_measure_3();
        song_measure_4();
        song_measure_5();
        song_measure_4();
        song_measure_6();
        song_measure_4();
        song_measure_5();
```

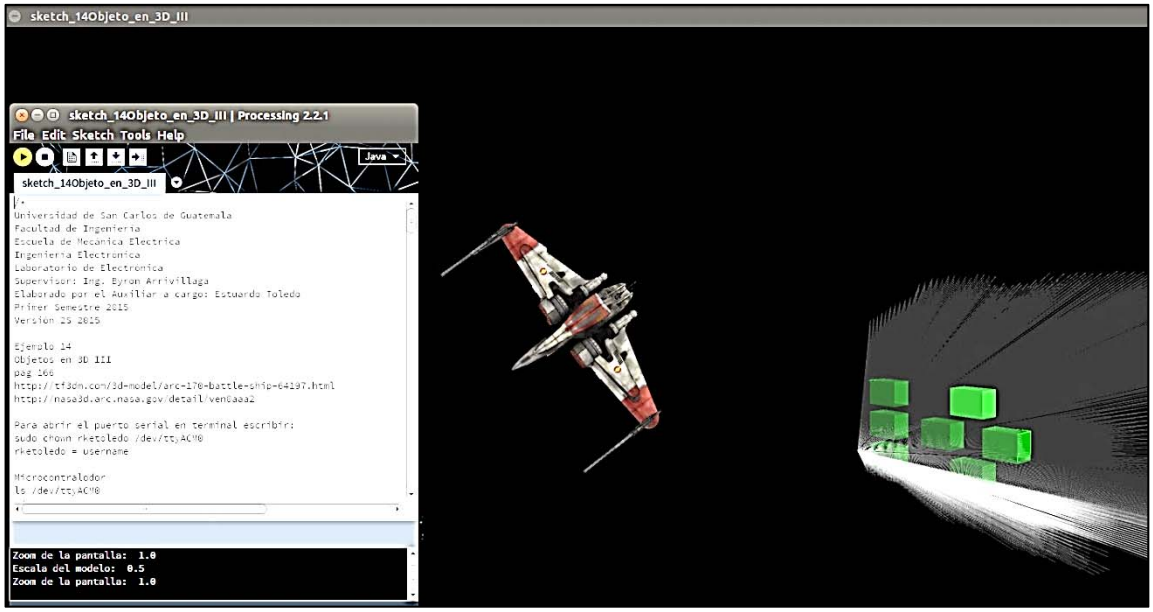
Fuente: elaboración propia.

Figura 119. **Código del ejemplo 14 parte 9, *code composer studio***

```
        song_measure_4();
        song_measure_6();
        song_measure_7();
        song_measure_8();
        song_measure_7();
}


#endif /* MELODY1_H_ */
```

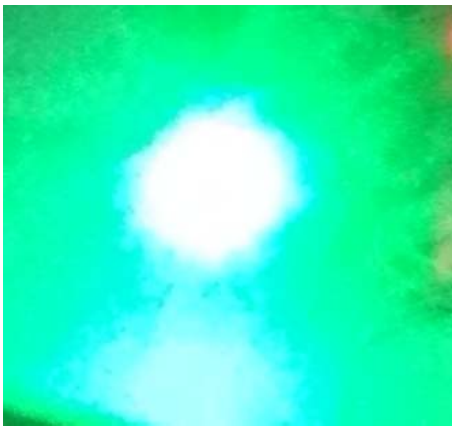
Fuente: elaboración propia.

Figura 120. **Resultado del código del ejemplo 14, parte 1**



Fuente: elaboración propia.

Figura 121. **Resultado del código del ejemplo 14, parte 2**



Fuente: elaboración propia.

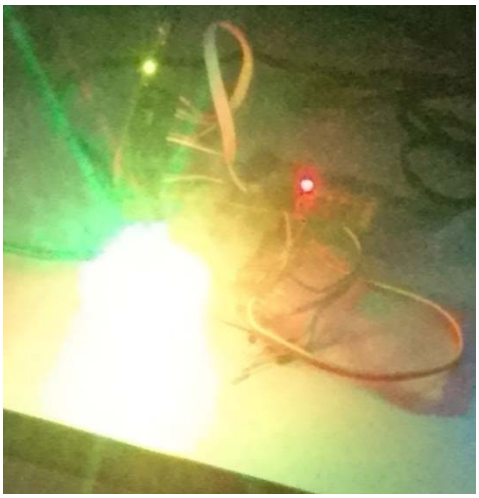
Figura 122. **Resultado del código del ejemplo 14, parte 3**

Fuente: elaboración propia.

Figura 123. **Resultado del código del ejemplo 14, parte 4**

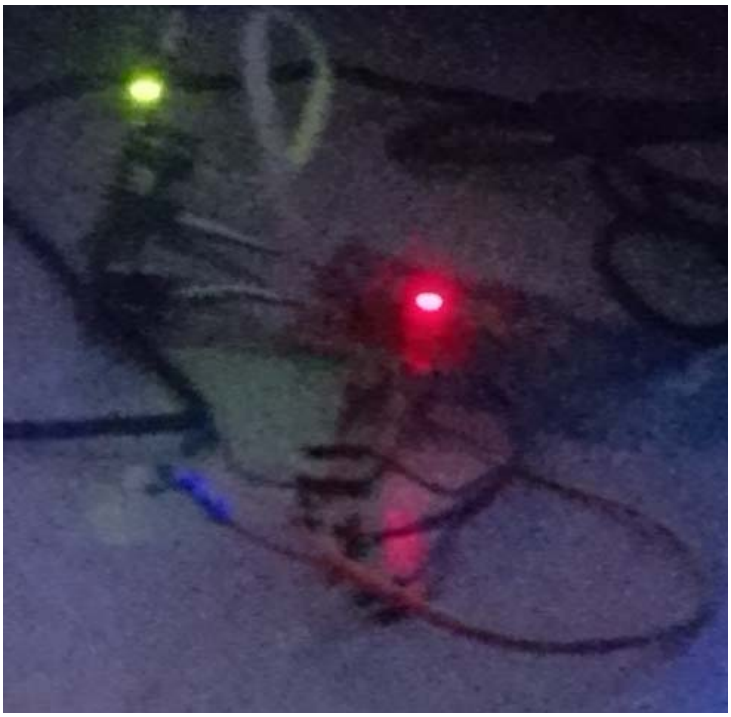
Fuente: elaboración propia.

Figura 124. **Resultado del código del ejemplo 14, parte 5**



Fuente: elaboración propia.

Figura 125. **Resultado del código del ejemplo 14, parte 6**



Fuente: elaboración propia.

Como se aprecia en la figura 125, se inicia la melodía y los *leds* del microcontrolador por medio de la selección de los cubos de la pantalla y usando las manos se controla la rotación y la escala del modelo de la pantalla.

5. OBTENCIÓN DEL ESQUELETO

A continuación se demuestra cómo seguir al usuario con OpenNI. Para realizar proyectos interactivos deben procesarse datos de profundidad para que respondan a las acciones de los usuarios. No se conoce la posición del usuario, por ello, debe obtenerse por medio de la nube de puntos. Con OpenNI se obtiene la capacidad para procesar la imagen de profundidad para detectar y rastrear a las personas. Con esto, no es necesario recorrer los puntos de profundidad para comprobar la posición, por lo que se accede a la posición de cada parte del cuerpo de cada usuario que OpenNI rastrea. Cuando OpenNI ha detectado a un usuario indicará la posición de cada articulación del usuario: cabeza, cuello, hombros, codos, manos, torso, caderas, rodillas, pies.

OpenNI utiliza el término “articulación” para referirse a todos los puntos en el cuerpo de un usuario que la librería es capaz de seguir, para conocer si son o no, articulaciones reales.

Por lo que ahora se tendrá la capacidad de implementar interfaces de usuario mucho más sofisticadas que antes no podían generarse. Pueden utilizarse las interacciones de: gestos, poses globales del cuerpo, realizar un seguimiento de los movimientos del cuerpo a través del tiempo, comparar los movimientos y medir las distancias entre las partes del cuerpo. Por lo que el trabajo es más complejo que en la imagen de profundidad o en la nube de puntos; y sin las capacidades básicas de programación, las matrices y la comprensión 3D se dificultaría al usar la librería de OpenNI, porque deben usarse nuevos conceptos de programación como: *setter methods, callbacks*, nuevas técnicas matemáticas de cálculos con vectores en tres dimensiones

como: vector resta, búsqueda de productos escalares, búsqueda de productos cruzados, entre otros.

5.1. Eje Z

No se utilizará el eje Z, porque ya se proyecta en un plano de dos dimensiones para alinearlo con la imagen de profundidad y si se mueve se perdería su registro con la imagen de profundidad.

En lugar de mover el eje Z, se escalará el tamaño del objeto que se desea mover para aportarle la perspectiva, que cuando más cerca está un objeto más dimensión tendrá y viceversa.

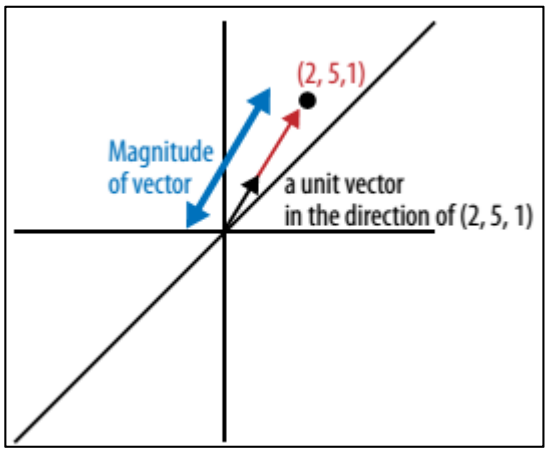
5.2. Vectores en tres dimensiones (3D)

Los vectores normalmente son usados para representar coordenadas de un punto en el espacio de una variable, pero, también pueden describir la distancia y la dirección.

5.2.1. La función normalización de vectores (*normalize*)

Esta función transforma un vector en un vector unitario, lo que significa que la magnitud del vector es uno.

Figura 126. **Vector 3D**



Fuente: GREG Borenstein. *Making Things See*. p. 221.

5.3. La función de almacenamiento de vectores (PMatrix3D)

Las matrices pueden almacenar vectores en una sola cuadrícula de números. En teoría, las matrices pueden contener un número arbitrario de vectores, pero en el presente caso PMatrix3D solo almacena cuatro vectores.

PMatrix3D almacena cuatro vectores que pueden representar el sistema de coordenadas. Se utilizan tres vectores para representar los ejes “X”, “Y”, y “Z” y sus componentes contiene cualquier escala y las operaciones para las rotaciones.

El cuarto componente del vector contiene las operaciones para la traslación. Este vector afecta a todo el sistema de coordenadas, por lo que se puede pensar en este como un sistema de coordenadas inicial.

Por ello, PMatrix3D almacena un conjunto de transformaciones que modifica el sistema de coordenadas inicial para que coincida con la orientación de la articulación del usuario.

Hay que calcular la diferencia en el ángulo entre el vector de la mano y el ángulo original del modelo. Si se realiza esto, entonces se rotan los argumentos correctos y el modelo se coloca en su posición correcta.

5.4. Orientación de modelos en tres dimensiones (3D)

Con OpenNI se puede mover los modelos 3D mediante las orientaciones del cuerpo, porque puede transferirse el ángulo y la orientación del cuerpo.

No obstante, los datos de orientación solo están disponibles para las articulaciones internas en el esqueleto del usuario, de las articulaciones que terminan en extremidades no se puede obtener datos de orientación.

Para orientar el modelo en la dirección correcta hay que calcular la diferencia del ángulo entre el vector de la mano y el ángulo inicial del modelo.

5.5. El método del eje y el ángulo (*Axis-Angle*)

Hay que representar la diferencia entre dos puntos, respecto su distancia y dirección. Por ello se puede calcular un solo vector que represente la posición relativa del usuario.

Por ello se hace coincidir la posición y el ángulo del modelo para que un vector conecte las manos del usuario y a este método se le llama *Axis-Angle*.

Con ello se puede usar la posición y el ángulo relativo de las extremidades del usuario.

El método Axis-Angle proporciona el ángulo y el eje necesario para describir la rotación relativa entre dos vectores. El método usa dos técnicas geométricas: el producto escalar y el producto vectorial.

5.5.1. Producto escalar

El producto escalar indica el ángulo entre vectores

5.5.2. Producto cruz

Proporciona un vector que representa el eje en que está definido el ángulo. El vector representa una línea en el espacio y el ángulo describe una rotación alrededor de esa línea.

5.6. Interacción natural (OpenNI)

OpenNI puede analizar la escena, seguir la mano del usuario, buscar el centro de masa del usuario, averiguar qué píxeles de la imagen de profundidad representan personas y cuáles representan el fondo de la escena. Esta información está disponible tan pronto como OpenNI detecta que una persona ha entrado en escena.

El algoritmo de OpenNI detecta que una persona está presente y realiza un seguimiento continuo mientras este se mueve alrededor de la escena. Por lo que se puede diferenciar usuarios. Esta información se proporciona en forma de una matriz de números que corresponden a cada píxel de la imagen de

profundidad. El valor de esta matriz o mapa para cada píxel será 0 si el píxel es parte de la escena y si el píxel es parte del usuario el valor del mapa será uno, dos, tres, entre otros.

5.7. Calibración

Para que el algoritmo de OpenNI rastree las articulaciones de una persona, esta necesita estar de pie en una pose conocida. De pie, con los pies juntos y los brazos levantados por encima de los hombros a los lados de la cabeza. Esta postura es conocida por varios nombres. En la literatura técnica y en la propia documentación del PrimeSense es llamada el "Psi" pose.

OpenNI tiene la capacidad de rastrear a los usuarios sin necesidad de una calibración, ya que puede grabar los datos de calibración de un solo usuario y luego utilizar esa información para otros, pero esta técnica es muy compleja y no funciona en la mayor parte de las situaciones.

5.7.1. Proceso de calibración

La calibración es un proceso de inicio y retorno entre el boceto y OpenNI. En cada paso OpenNI realizará nuestras funciones de llamado y dentro de esa función se realiza una acción para continuar el proceso. Por lo que esta acción provoca el siguiente llamado sucesivamente.

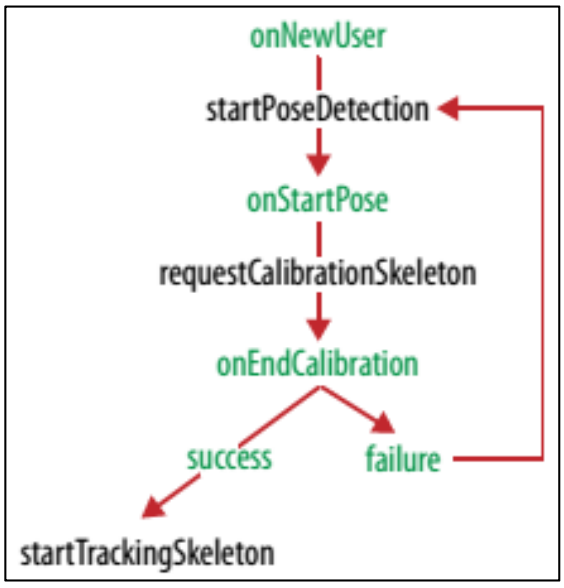
Las etapas del seguimiento son:

- La detección básica de usuario
- Calibración
- Usuario seguido

Cuando el programa inicia, OpenNI no sigue al usuario, porque el proceso comienza cuando el usuario entra a la escena, esto se llama función onNewUsero. Dentro de ella se inicia el proceso de seguimiento llamando startPoseDetection. Esta función le pregunta a OpenNI si el usuario está en la posición de la calibración inicial. Después, que el usuario asume la posición inicial, OpenNI llama a la función onStartPose, para continuar con el proceso de calibración, por lo que se llama a requestCalibrationSkeleton. Esta función inicia el proceso de detección del esqueleto de OpenNI, cuando dicho proceso se completa, OpenNI llamará a la función onEndCalibration.

La calibración no siempre es exitosa, por lo que OpenNI indicará el estado de la calibración, si esta es exitosa se llama a startTrackingSkeleton y en caso contrario, deberá comenzarse nuevamente llamando a startPoseDetection.

Figura 127. **Calibración**



Fuente: GREG Borenstein. *Making Things See*. p. 193.

5.8. Rastreo de la mano (*hand tracking*) sin calibración

Usa una función avanzada de OpenNI que minimiza el uso de píxeles de la imagen de profundidad para adoptar una posición útil en esta aplicación. Este sistema se llama NITE y puede detectar cuándo el usuario levanta su mano, también usa *callback function*.

Este sistema permite detectar la presencia de la mano del usuario y seguir su movimiento sin que el usuario realice la calibración. Por lo que el tiempo de inicio para empezar a interactuar con la aplicación es más rápido y permite el seguimiento del usuario a un cuando se encuentre demasiado cerca del Kinect

5.9. Centro de masa (*center of mass*) sin calibración

Usa una función avanzada de OpenNI que minimiza el uso de píxeles de la imagen de profundidad a una posición útil en esta aplicación. Este sistema se llama NITE y puede detectar cuándo el usuario levanta su mano, también usa *callback function*.

Esta técnica permite realizar un seguimiento de los usuarios que están lejos del Kinect y que no saben que están frente a la cámara. Cuando OpenNI detecta a un usuario, este informa sobre el centro de masa del usuario. Con el método del centro de masa se puede seguir a varios usuarios, porque es difícil que cuatro usuarios encajen en la vista del Kinect en un rango donde se puedan distinguir para realizar la calibración.

5.9.1. La función centro de masa (getCoM)

Es la función que se utiliza para encontrar el centro de masa del usuario, esta toma el ID (número de identificación) del usuario y almacena el centro de masa en un vector que se utiliza para convertir la posición del centro de masa en coordenadas proyectivas, para que coincida con la imagen de profundidad. El centro de masa se localiza aproximadamente en el ombligo.

5.10. Acceso a las articulaciones

A continuación se ilustra cómo mostrar las articulaciones y cómo estas corresponden con la imagen de profundidad. Para logarlo se deben traducir las coordenadas reales que proporciona OpenNI a coordenadas que encajen en la imagen bidimensional de profundidad. Para ello se utilizan las funciones de SimpleOpenNI. Para conseguir que las articulaciones se alineen con la imagen de profundidad hay que convertir la posición conjunta del mundo real en coordenadas proyectadas y SimpleOpenNI proporciona una función para realizar esta alineación: convertRealWorldToProjective.

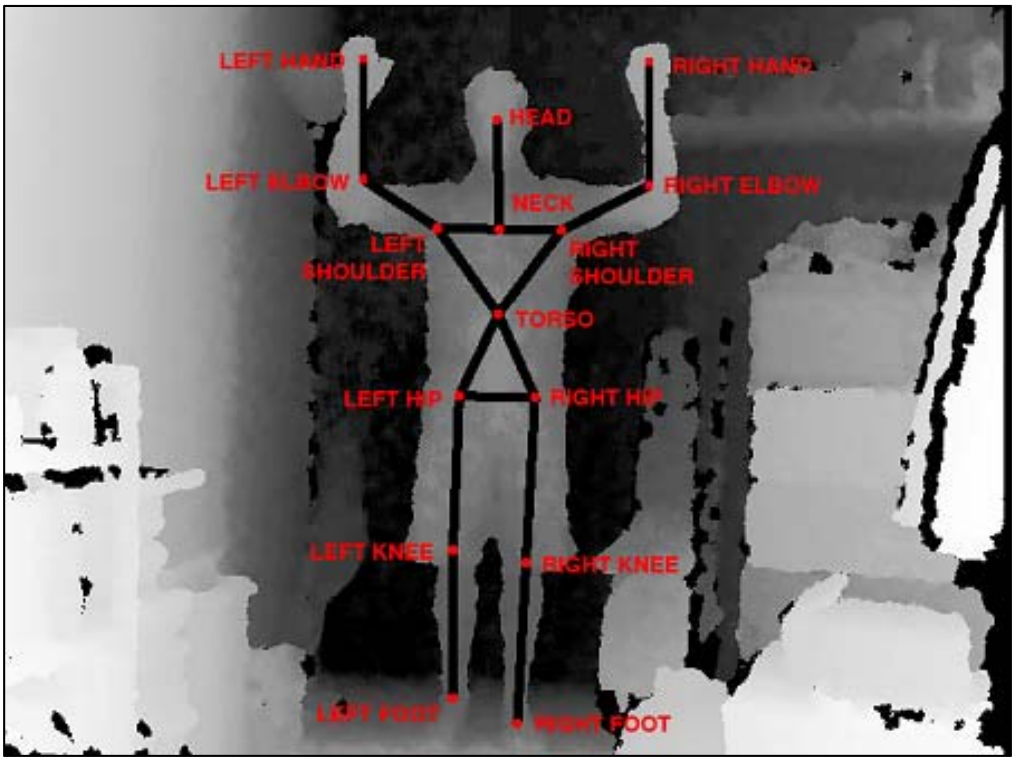
Es importante señalar, que acceder a las articulaciones se debe a que OpenNI piensa en la orientación de las articulaciones desde el punto de vista de la pantalla en lugar de la del usuario. Lo anterior es porque que el usuario se encuentra frente al Kinect, por ello la mano derecha se muestra en el lado izquierdo de la pantalla. Esta orientación invertida se mantiene en todas las articulaciones que tiene pares de izquierda a derecha: manos, brazos, piernas, entre otros.

5.11. El esqueleto de la interacción natural (OpenNI)

OpenNI proporciona elipses rojas en las posiciones de las articulaciones y líneas negras que conectan a las articulaciones.

El rastreo de las articulaciones y extremidades realizado por OpenNI sirve para la creación de aplicaciones interactivas, por lo que el esqueleto no es anatómicamente correcto. Las articulaciones que proporciona OpenNI son determinadas, por lo que el algoritmo puede detectar, y lo que es útil para los programadores.

Figura 128. **Esqueleto de la interacción natural (OpenNI)**



Fuente: GREG Borenstein. *Making Things See*. p. 193.

OpenNI tiene una variación en las posiciones de las partes reales del cuerpo, porque a veces terminan un tanto bajo, demasiado alto o están a un lado de la posición real de las articulaciones.

OpenNI puede rastrear 15 articulaciones y en ocasiones puede rastrear 9 articulaciones adicionales, las cuales son: WAIST, LEFT ANKLE, RIGHT ANKLE, LEFT COLLAR, RIGHT COLLAR, LEFT WRIST, RIGHT WRIST, LEFT FINGERTIP y RIGHT FINGERTIP. Pero para OpenNI rastrear estas articulaciones puede resultar muy difícil debido a las limitaciones del hardware del Kinect, por lo que no se utilizan en el presente trabajo.

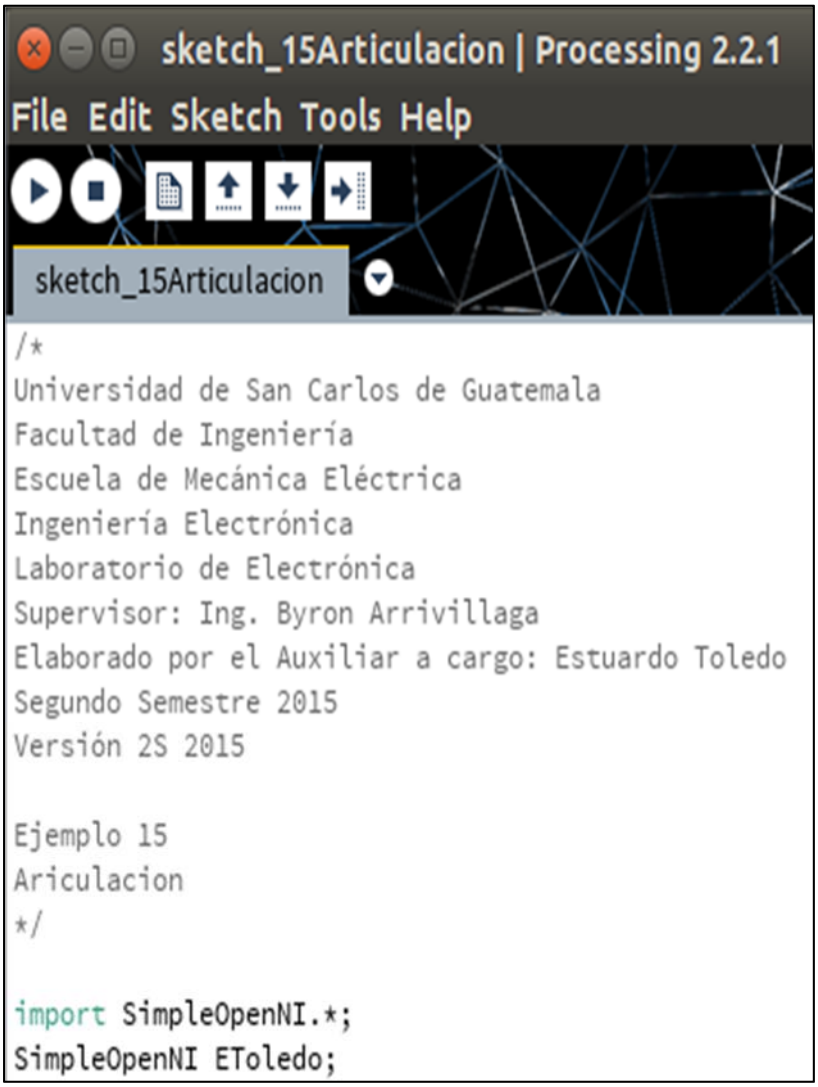
5.12. Ejemplos del esqueleto

A continuación se proporcionan ejemplos para entender, detectar y rastrear las articulaciones del esqueleto virtual y usar esta información para manipular aplicaciones interactivas para el usuario.

5.12.1. Ejemplo 15, articulación

En este ejemplo se manipula la posición y la escala, una elipse y una imagen por medio de la posición de las manos.

Figura 129. **Código del ejemplo 15, parte 1**



Fuente: elaboración propia.

Figura 130. **Código del ejemplo 15, parte 2**

```
//Variables de la imagen
PImage Imagen1;
float Imagen1Escala;

void setup() {
  size(640, 480, OPENGL);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  //Habilita el rastreo del usuario
  EToledo.enableUser();
  //Cargar imagen
  Imagen1 = loadImage("z5.jpg");
}

void draw() {
  EToledo.update();
  //Variable para almacenar las imágenes de profundidad
  PImage depth = EToledo.depthImage();
  //Desplegar imagen de profundidad en las coordenadas X y Y
  image(depth, 0, 0);
  //Vector que almacena la lista de usuarios
  IntVector userList = new IntVector();
  //Escribe la lista de usuarios en nuestro vector
  EToledo.getUsers(userList);
  //Si encontramos al menos un usuario
    if (userList.size() > 0) {
      //Obtener el primer usuario
      int userId = userList.get(0);
        //Si se ha calibrado correctamente se continua con el programa
          if ( EToledo.isTrackingSkeleton(userId)) {
            //Vector para la mano derecha
            PVector rightHand = new PVector();
            //Vector para la mano izquierda
            PVector leftHand = new PVector();
```

Fuente: elaboración propia.

Figura 131. **Código del ejemplo 15, parte 3**

```
//Poner la posición de la mano izquierda en el vector
//Confidence tiene valores de 0 a 1
//Si es menor que 1 la articulación no esta presente
float confidence1 = EToledo.getJointPositionSkeleton(userId,
                    SimpleOpenNI.SKEL_LEFT_HAND, rightHand);
float confidence2 = EToledo.getJointPositionSkeleton(userId,
                    SimpleOpenNI.SKEL_RIGHT_HAND, leftHand);
//Convertir la posición de la mano detectada a coordenadas
//que coincidan con la imagen de profundidad (plano X,Y)
 PVector convertedRightHand = new PVector();
 PVector convertedleftHand = new PVector();
 EToledo.convertRealWorldToProjective(rightHand,
                              convertedRightHand);
 EToledo.convertRealWorldToProjective(leftHand,
                              convertedleftHand);
 //--------------------------------------Desplegar Punto
 pushMatrix();
   fill(255,0,0);
   //Escala el tamaño de la elipse de 700 a 2500
   float ellipseSize = map(convertedRightHand.z,
                              700, 2500, 50, 1);
   //Si OpenNI esta siguiendo la mano dibuja la elipse
   if(confidence1 > 0.5){
     ellipse(convertedRightHand.x, convertedRightHand.y,
                              ellipseSize, ellipseSize);
   }
 popMatrix();
 //-------------------------------------Desplegar imagen
 pushMatrix();
   //Escala de la imagen de 1400 a 5000
   Imagen1Escala = map(convertedleftHand.z, 1400, 5000, 50, 4);
   //Si OpenNI esta siguiendo la mano dibuja la elipse
   if(confidence2 > 0.5){
```

Fuente: elaboración propia.

Figura 132. **Código del ejemplo 15, parte 4**

```
                image(Imagen1,convertedleftHand.x, convertedleftHand.y,
                                                Imagen1Escala, Imagen1Escala);
              }
            popMatrix();
      }
    }
}

// SimpleOpenNI user events
void onNewUser(SimpleOpenNI curContext,int userId){
  println("ID del usuario: " + userId);
  println("Inicia el rastreo de las articulaciones");
  EToledo.startTrackingSkeleton(userId);
}
```

Fuente: elaboración propia.

Figura 133. **Resultado del código del ejemplo 15**

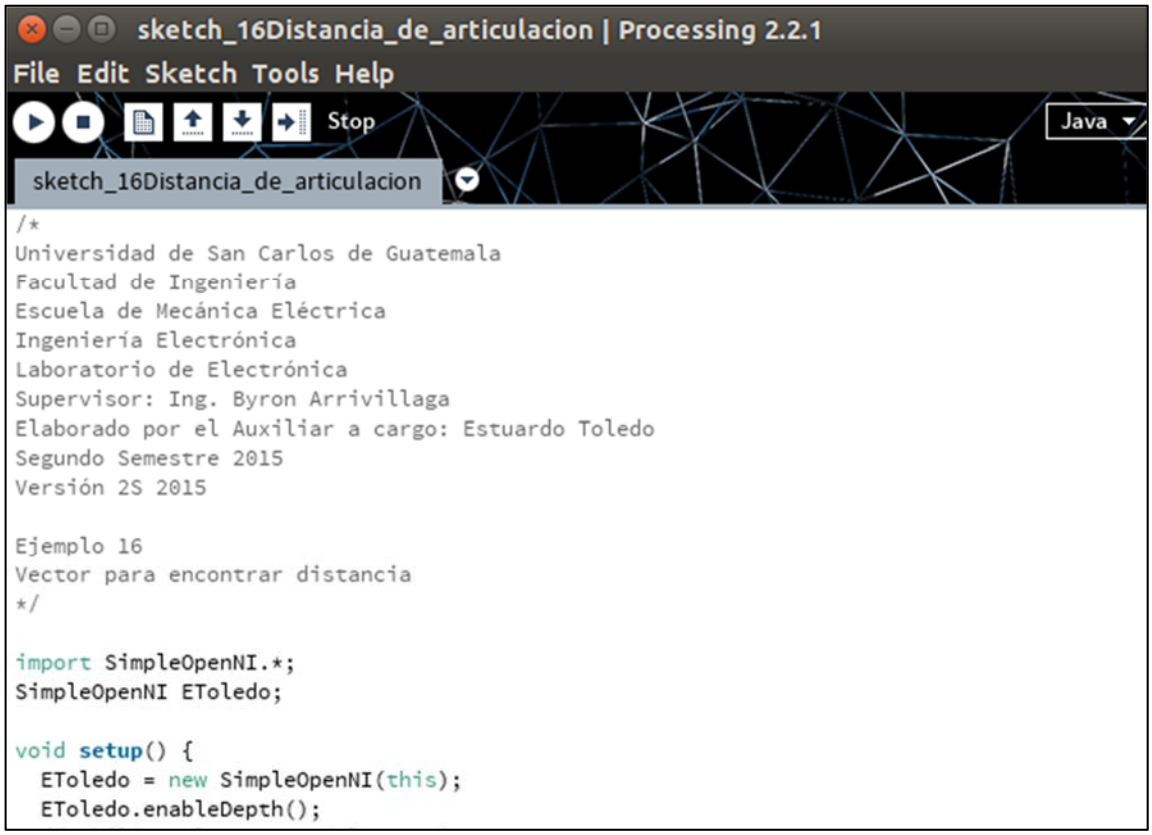
Fuente: elaboración propia.

De acuerdo a la figura 133, se cambia la posición y la escala de la elipse y la imagen, obteniendo las posiciones de las manos en el plano 3D y convirtiéndolo al plano 2D.

5.12.2. Ejemplo 16, distancia de la articulación

En este ejemplo se obtienen las posiciones de las manos y se mide la distancia que hay entre ellas y se crea un dibujo con los movimientos de las manos.

Figura 134. **Código del ejemplo 16, parte 1**



Fuente: elaboración propia.

Figura 135. **Código del ejemplo 16, parte 2**

```
  //Habilita el rastreo del usuario
  EToledo.enableUser();
  size(1280, 480, OPENGL);
  EToledo.setMirror(true);
}

void draw() {
  EToledo.update();
  //Desplegar imagen de profundidad en (X, Y)
  image(EToledo.depthImage(), 640, 0);
  //Vector que almacena la lista de usuarios
  IntVector userList = new IntVector();
  //Escribe la lista de usuarios en nuestro vector
  EToledo.getUsers(userList);
  //Si encontramos al menos un usuario
  if (userList.size() > 0) {
    //Obtener el primer usuario
    int userId = userList.get(0);
    //Si se ha calibrado correctamente se continua con el programa
    if ( EToledo.isTrackingSkeleton(userId)) {
      //Vector para la mano derecha
      PVector rightHand = new PVector();
      //Vector para la mano izquierda
      PVector leftHand = new PVector();
      //Poner la posición de las manos en el vector
      //Confidence tiene valores de 0 a 1
      //Si es menor que 1 la articulación no esta presente
      float confidence1 = EToledo.getJointPositionSkeleton(userId,
      SimpleOpenNI.SKEL_LEFT_HAND, leftHand);
      float confidence2 = EToledo.getJointPositionSkeleton(userId,
      SimpleOpenNI.SKEL_RIGHT_HAND, rightHand);

      //Resta de vectores
      PVector differenceVector = PVector.sub(leftHand, rightHand);
      //Obtener la magnitud del vector
      float magnitude = differenceVector.mag();
```

Fuente: elaboración propia.

Figura 136. **Código del ejemplo 16, parte 3**

```
//Obtener unidades de longitud
float millimeters = magnitude;
float centimeter = millimeters / 10;
float meters = centimeter / 100;
float feet = centimeter / 30.48;
float inches = millimeters / 25.4;
//Se normaliza el vector
differenceVector.normalize();
//Se despliega en la pantalla si el Kinect esta rastreando al
//usuario
if((confidence1 > 0.5) && (confidence2 > 0.5)){
  //-------------------------------------------------------
  //Dibuja la linea del lado derecho de la pantalla
  pushMatrix();
    translate(width/2, height/32);
    //Establece el color utilizado para dibujar
    //líneas y bordes alrededor de las formas (RGB)
    stroke(0, 0, 255);
    //Establece el ancho del trazo utilizado para líneas,
    //puntos, y el orden en torno a formas
    strokeWeight(5);
    //Se dibuja la línea entre las dos manos
    EToledo.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HAND,
                             SimpleOpenNI.SKEL_RIGHT_HAND);
  popMatrix();
  //-------------------------------------------------------
  //Dijuba la linea del lado izquierdo de la pantalla
  pushMatrix();
    //Establece el color utilizado para dibujar
    //líneas y bordes alrededor de las formas (RGB)
    stroke(abs(differenceVector.x) * 255,
           abs(differenceVector.y) * 255,
          abs(differenceVector.z) * 255);
    //Establece el ancho del trazo utilizado para líneas,
    //puntos, y el orden en torno a formas
    strokeWeight(5);
```

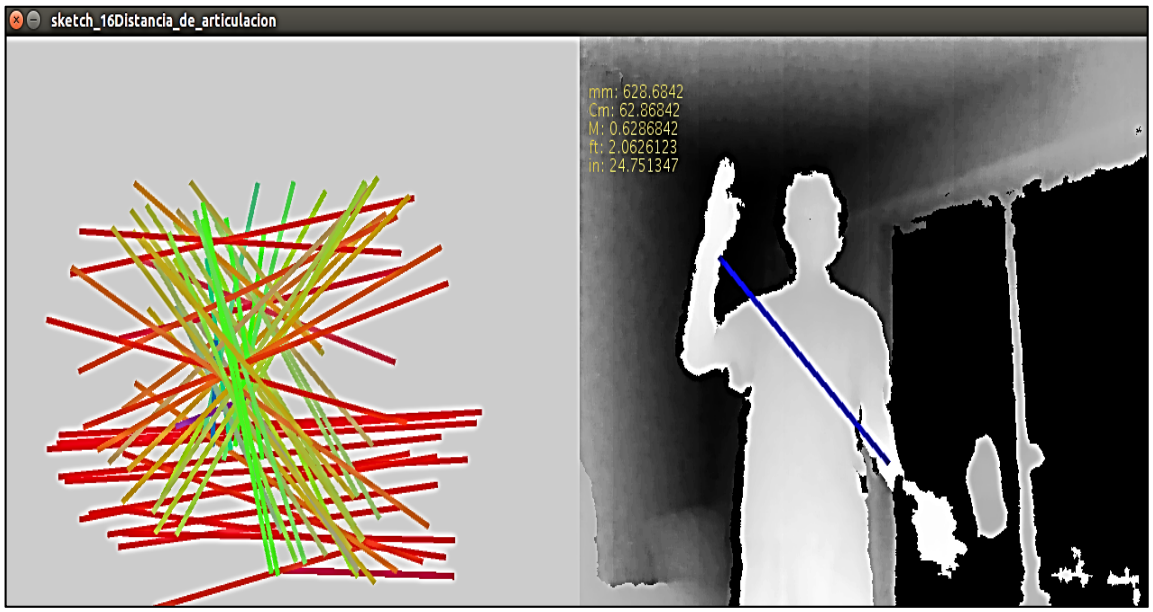
Fuente: elaboración propia.

Figura 137. **Código del ejemplo 16, parte 4**

```
          //Se dibuja la línea entre las dos manos
          EToledo.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HAND,
                                   SimpleOpenNI.SKEL_RIGHT_HAND);
        popMatrix();
        //-----------------------------------------------------------
        pushMatrix();
          //Establece el color utilizado para llenar formas (RGB)
          //abs es el valor absoluto de un numero
          fill(abs(differenceVector.x) * 255,
               abs(differenceVector.y) * 255,
              abs(differenceVector.z) * 255);
          //Establece el tamaño de la fuente
          textSize(15);
          //Funcion para escribir texto en (X, Y), del lado izquiedo
          //de la pantalla
          text("mm: " + millimeters, 650, 50);
          text("Cm: " + centimeter, 650, 65);
          text("M: " + meters, 650, 80);
          text("ft: " + feet, 650, 95);
          text("in: " + inches, 650, 110);
        popMatrix();
      }
    }
  }

}
void onNewUser(SimpleOpenNI curContext,int userId){
  println("ID del usuario: " + userId);
  println("Inicia el rastreo de las articulaciones");
  EToledo.startTrackingSkeleton(userId);
}

void onVisibleUser(SimpleOpenNI curContext,int userId){
  println("Usuario activo, ID: " + userId);
}

void onLostUser(SimpleOpenNI kinect, int userId) {
  println("Usuario no encontrado, ID: " + userId);
}
```

Fuente: elaboración propia.

Figura 138. **Resultado del código del ejemplo 16**



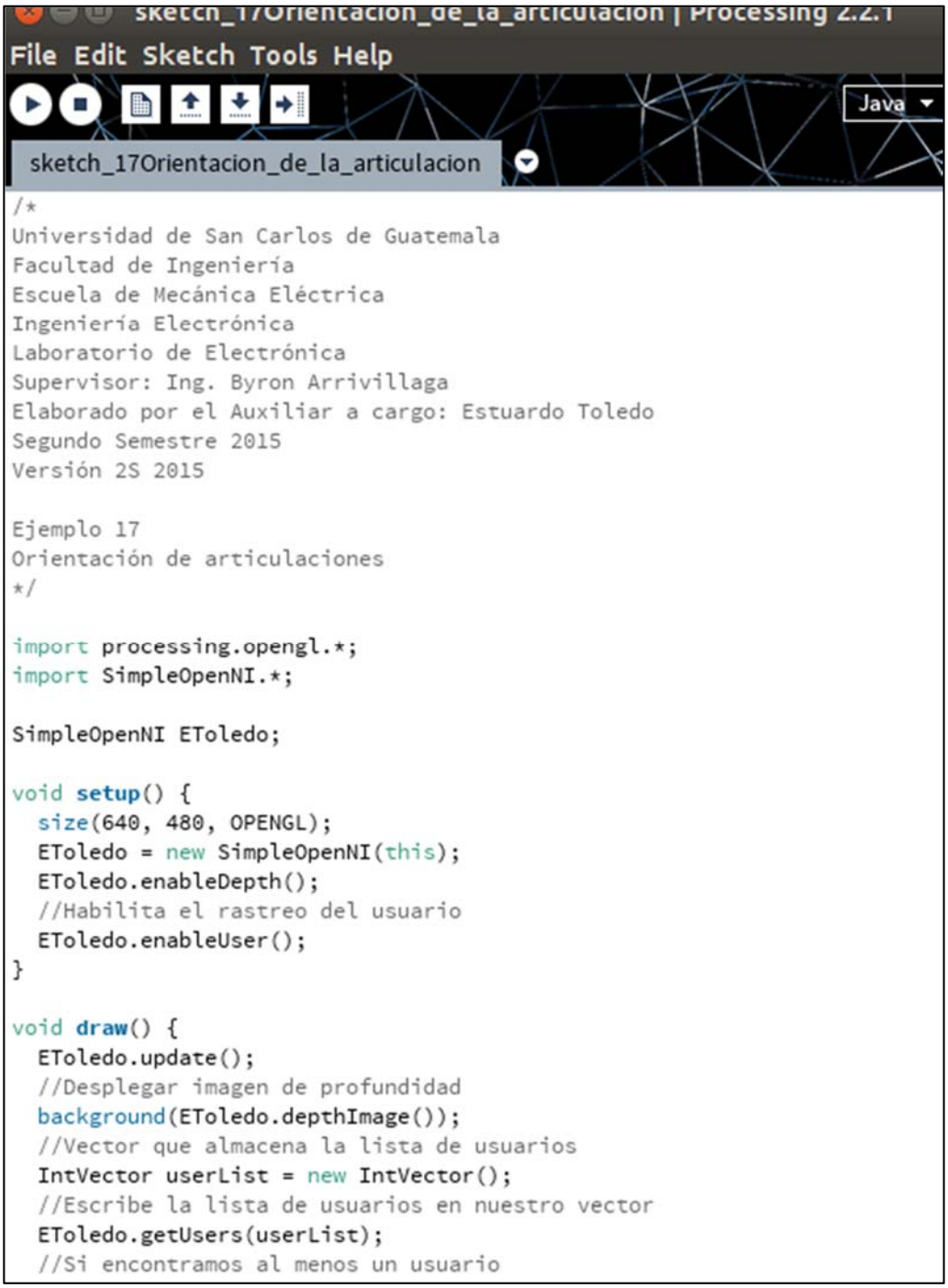
Fuente: elaboración propia.

Como se aprecia en la imagen, se manipula el color de la línea que une las manos y se obtiene la distancia que hay entre ellas.

5.12.3. Ejemplo 17, orientación de la articulación

En este ejemplo se obtiene la orientación de las articulaciones.

Figura 139. **Código del ejemplo 17, parte 1**




```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2015
Versión 2S 2015

Ejemplo 17
Orientación de articulaciones
*/

import processing.opengl.*;
import SimpleOpenNI.*;

SimpleOpenNI EToledo;

void setup() {
  size(640, 480, OPENGL);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  //Habilita el rastreo del usuario
  EToledo.enableUser();
}

void draw() {
  EToledo.update();
  //Desplegar imagen de profundidad
  background(EToledo.depthImage());
  //Vector que almacena la lista de usuarios
  IntVector userList = new IntVector();
  //Escribe la lista de usuarios en nuestro vector
  EToledo.getUsers(userList);
  //Si encontramos al menos un usuario
```

Fuente: elaboración propia.

Figura 140. **Código del ejemplo 17, parte 2**

```
if (userList.size() > 0) {
  //Obtener el primer usuario
  int userId = userList.get(0);
  //Si se ha calibrado correctamente se continua con el programa
  if ( EToledo.isTrackingSkeleton(userId)) {
      //------------------------------------------NECK datos
      //Vector para la posición
      PVector position1 = new PVector();
      //Poner la posición del NECK en el vector
      EToledo.getJointPositionSkeleton(userId,
           SimpleOpenNI.SKEL_NECK, position1);
      //Matriz de cuatro vectores, traslación y rotación ejes (X,Y,Z)
      PMatrix3D orientation1 = new PMatrix3D();
      //Poner la orientación de NECK en el vector
      //Confidence tiene valores de 0 a 1
      //Si es menor que 1 la articulación no esta presente
      float confidence1 = EToledo.getJointOrientationSkeleton(userId,
                              SimpleOpenNI.SKEL_NECK, orientation1);
      //------------------------------------------RIGHT ELBOW datos
      //Vector para la posición
      PVector position2 = new PVector();
      //Poner la posición de RIGHT ELBOW en el vector
      EToledo.getJointPositionSkeleton(userId,
           SimpleOpenNI.SKEL_RIGHT_ELBOW, position2);
      //Matriz de cuatro vectores, traslación y rotación ejes (X,Y,Z)
      PMatrix3D orientation2 = new PMatrix3D();
      //Poner la orientación de RIGHT ELBOW en el vector
      //Confidence tiene valores de 0 a 1
      //Si es menor que 1 la articulación no esta presente
      float confidence2 = EToledo.getJointOrientationSkeleton(userId,
                          SimpleOpenNI.SKEL_RIGHT_ELBOW, orientation2);

     if(confidence1 > 0.5){
       //------------------------------------------NECK desplegar
       pushMatrix();
         translate(width/2, height/2, 0);
         rotateX(radians(180));
```

Fuente: elaboración propia.

Figura 141. **Código del ejemplo 17, parte 3**

```
            //Lo traslada a la posición de NECK
            translate(position1.x, position1.y, position1.z);
            //Adopta la posición del torso para que sea el sistema de
            //coordenadas o punto de referencia
            applyMatrix(orientation1);
            //Establece el ancho del trazo utilizado para líneas,
            //puntos, y el orden en torno a formas
            strokeWeight(10);
            //------------------------El eje X se dibuja en rojo
            stroke(255, 0, 0);
            //Dibuja una linea en (X1, Y1, Z1, X2, Y2, Z2)
            //(Origien, Fin)
            line(0, 0, 0, 200, 0, 0);
            //------------------------El eje Y se dibuja en verde
            stroke(0, 255, 0);
            //Dibuja una linea en (X1, Y1, Z1, X2, Y2, Z2)
            //(Origien, Fin)
            line(0, 0, 0, 0, 200, 0);
            //------------------------EL eje Z se dibuja en azul
            stroke(0, 0, 255);
            //Dibuja una linea en (X1, Y1, Z1, X2, Y2, Z2)
            //(Origien, Fin)
            line(0, 0, 0, 0, 0, 200);
          popMatrix();
        }
        if(confidence2 > 0.5){
          //--------------------------------------RIGHT ELBOW desplegar
          pushMatrix();
            translate(width, height/32, 0);
            rotateX(radians(180));
            //Lo traslada a la posición de RIGHT ELBOW
            translate(position2.x, position2.y, position2.z);
            //Adopta la posición del torso para que sea el sistema de
            //coordenadas o punto de referencia
            applyMatrix(orientation2);
            //Establece el ancho del trazo utilizado para líneas,
            //puntos, y el orden en torno a formas
```

Fuente: elaboración propia.

Figura 142. **Código del ejemplo 17, parte 4**

```
            strokeWeight(10);
            //------------------------El eje X se dibuja en rojo
            stroke(255, 0, 0);
            //Dibuja una linea en (X1, Y1, Z1, X2, Y2, Z2)
            //(Origien, Fin)
            line(0, 0, 0, 200, 0, 0);
            //------------------------El eje Y se dibuja en verde
            stroke(0, 255, 0);
            //Dibuja una linea en (X1, Y1, Z1, X2, Y2, Z2)
            //(Origien, Fin)
            line(0, 0, 0, 0, 200, 0);
            //------------------------EL eje Z se dibuja en azul
            stroke(0, 0, 255);
            //Dibuja una linea en (X1, Y1, Z1, X2, Y2, Z2)
            //(Origien, Fin)
            line(0, 0, 0, 0, 0, 200);
        popMatrix();
       }
    }
  }
}

void onNewUser(SimpleOpenNI curContext,int userId){
  println("ID del usuario: " + userId);
  println("Inicia el rastreo de las articulaciones");
  EToledo.startTrackingSkeleton(userId);
}

void onVisibleUser(SimpleOpenNI curContext,int userId){
  println("Usuario activo, ID: " + userId);
}

void onLostUser(SimpleOpenNI kinect, int userId) {
  println("Usuario no encontrado, ID: " + userId);
}
/* Nota la obtención de la orientación no funciona
con RIGHT HAND, RIGHT FOOT, LEFT HAND Y LEFT FOOT
```

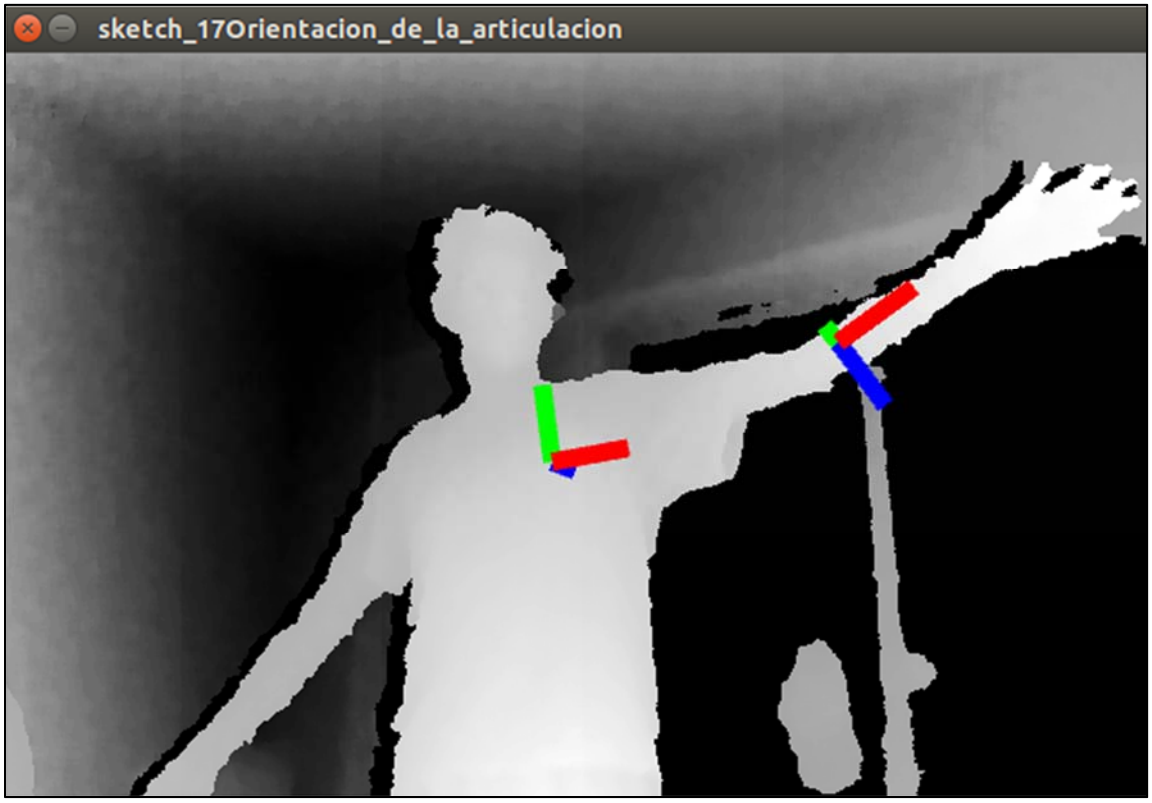
Fuente: elaboración propia.

Figura 143. **Código del ejemplo 17, parte 5**

```
por las limitaciones de OpenNI para determinar
la orientación de las articulaciones exteriores del esqueleto */
```

Fuente: elaboración propia.

Figura 144. **Resultado del código del ejemplo 17**



Fuente: elaboración propia.

Como resultado se obtiene la posición de las articulaciones del esqueleto virtual con la excepción de aquellas que se encuentran en los extremos debido a las limitaciones de OpenNI.

5.12.4. Ejemplo 18, objeto 3D y articulación

En este se usa la posición y la orientación de las articulaciones para manipular la posición, orientación y la escala de un objeto 3D.

Figura 145. **Código del ejemplo 18, parte 1**

Fuente: elaboración propia.

Figura 146. **Código del ejemplo 18, parte 2**

```
//-----------------------------------------------Delay
int Ti = 0;
int To = 0;
int Modulo = 0;
int Time = 100;
void setup() {

  size(1440, 720, P3D);
  //Cargar fondo
  fondo = loadImage("ven0aaa2.jpg");
  //Carga el modelo
  //-----------------------------------------------Modelo
  model = loadShape("Arc-170ship/Arc_obj_Sh3d adapted/Arc170.obj");
  //escala inicial del modelo
  model.scale(escala);
  //Rota el modelo para corregir su posición.
  model.rotateZ(11);
  //Constructor del modelo
  noStroke();
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  //Habilita el rastreo del usuario
  EToledo.enableUser();
  Ti = millis(); //-------------------------------Tiempo inicial
}

void draw() {
  EToledo.update();
  background(fondo);
  //Proporciona el efecto de luces y
  //sombras al modelo OBJ para hacerlo más real
  lights();
  translate(width/2, height/2, -100);
  //translate(width/2, height/2, -0);
```

Fuente: elaboración propia.

Figura 147. **Código del ejemplo 18, parte 3**

```
rotateX(radians(180));
//Vector que almacena la lista de usuarios
IntVector userList = new IntVector();
//Escribe la lista de usuarios en nuestro vector
EToledo.getUsers(userList);
//Si encontramos al menos un usuario
if (userList.size() > 0) {
  Modulo = (To - Ti); //-------------------------cambiar tiempo
  //Obtener el primer usuario
  int userId = userList.get(0);
  //Si se ha calibrado correctamente se continua con el programa
  if ( EToledo.isTrackingSkeleton(userId)) {
    //Vector para la posición
    PVector position = new PVector();
    //Obtener la posicion
    EToledo.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
                                                              position);
    //Matrix de cuatro vectores, traslación y rotación ejes (X,Y,Z)
    PMatrix3D orientation = new PMatrix3D();
    //Obtener la orientación
    float confidence = EToledo.getJointOrientationSkeleton(userId,
                      SimpleOpenNI.SKEL_RIGHT_ELBOW, orientation);
    if(Modulo > Time) {
      pushMatrix();
        translate(position.x, position.y, position.z);
        //Traslada la posición y la orientación
        //de la articulación para el modelo
        applyMatrix(orientation);
        shape(model);
      popMatrix();
    }
  }
}
//-------------------------------------------------Tiempo final
To = millis();
```

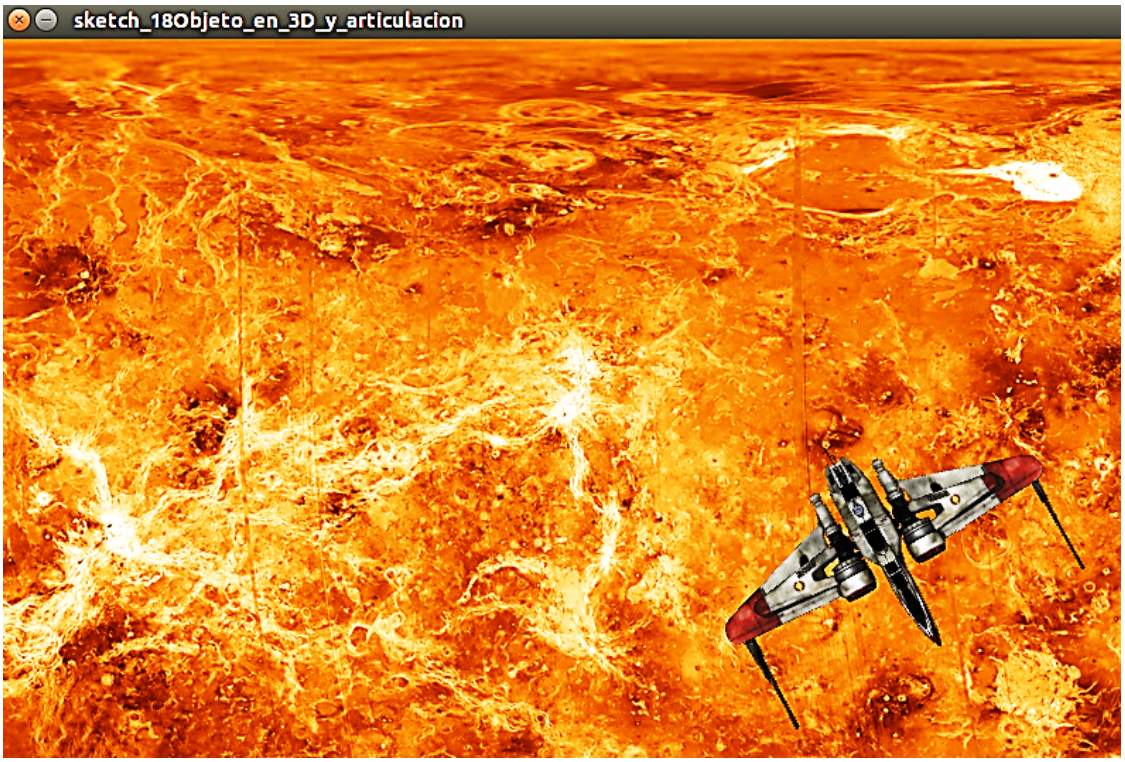
Fuente: elaboración propia.

Figura 148. **Código del ejemplo 18, parte 4**

```
void onNewUser(SimpleOpenNI curContext,int userId){
  println("ID del usuario: " + userId);
  println("Inicia el rastreo de las articulaciones");
  EToledo.startTrackingSkeleton(userId);
}
void onVisibleUser(SimpleOpenNI curContext,int userId){
  println("Usuario activo, ID: " + userId);
}
void onLostUser(SimpleOpenNI kinect, int userId) {
  println("Usuario no encontrado, ID: " + userId);
}
```

Fuente: elaboración propia.

Figura 149. **Resultado del código del ejemplo 18**



Fuente: elaboración propia.

Como resultado se manipula la posición, la orientación y la escala del modelo con el movimiento del codo.

5.12.5. Ejemplo 19, centro de masa sin calibración

Se obtiene el centro de masa del usuario sin necesidad de realizar el proceso de calibración del usuario.

Figura 150. **Código del ejemplo 19, parte 1**

```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2015
Versión 2S 2015

Ejemplo 19
Centro de masa
*/

import SimpleOpenNI.*;
SimpleOpenNI EToledo;

void setup() {
  size(640, 480);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  //Habilita el rastreo del usuario
  EToledo.enableUser();
}

void draw() {
  EToledo.update();
  //Carga la imagen de profundidad en (X, Y)
  image(EToledo.depthImage(), 0, 0);
  //Vector que almacena la lista de usuarios
  IntVector userList = new IntVector();
  //Escribe la lista de usuarios en nuestro vector
  EToledo.getUsers(userList);
```

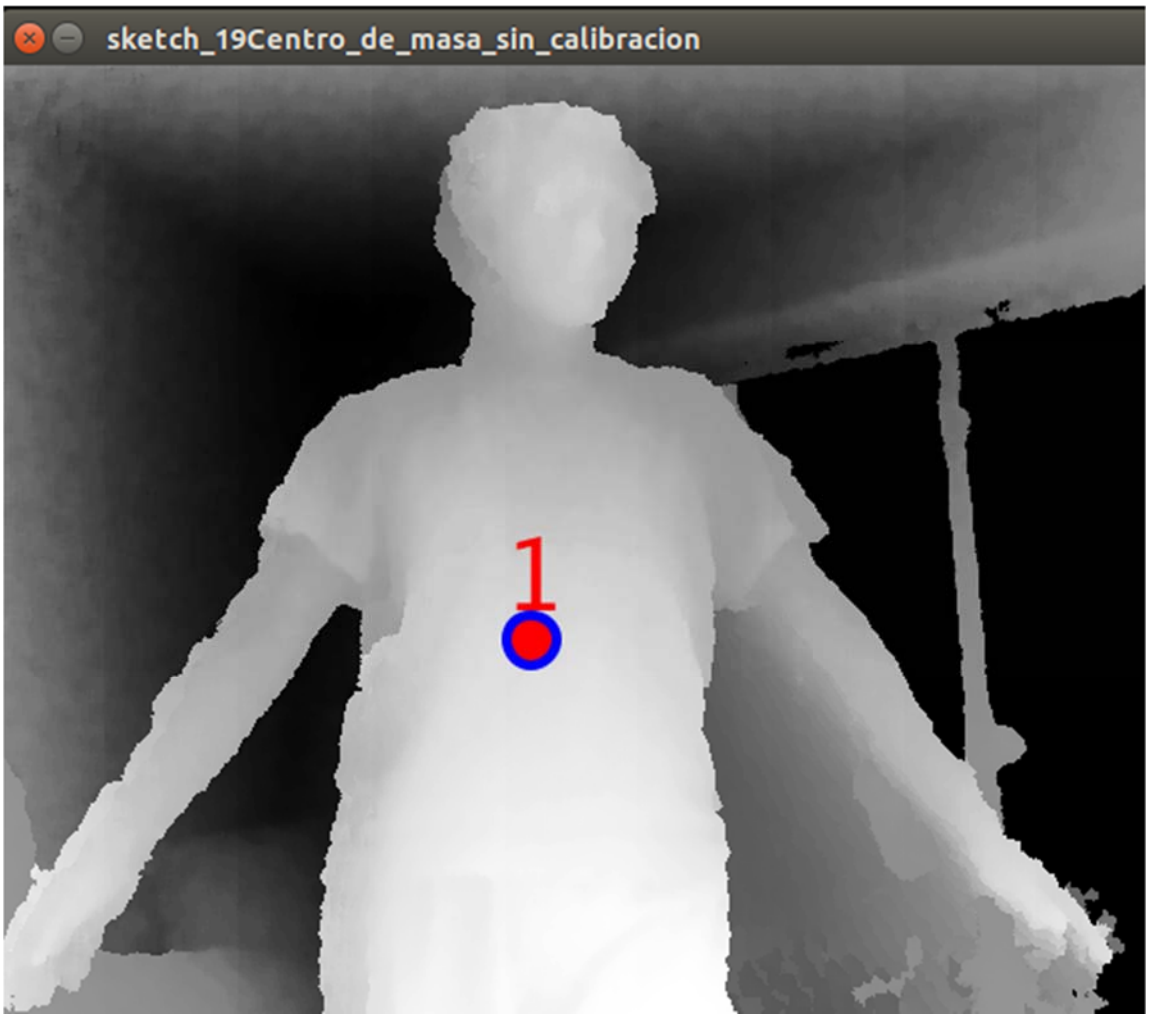
Fuente: elaboración propia.

Figura 151. **Código del ejemplo 19, parte 2**

```
  //Controla la secuencia de obtención de los datos
  //de los usuarios usuarios
  for (int i = 0; i < userList.size(); i++) {
    //Obtener el primer usuario
    int userId = userList.get(i);
    //Vector para la posición
    PVector position = new PVector();
    //getCoM "CoM" es la función de centro de masa
    //Obtiene la posición del centro de masa del usuario
    EToledo.getCoM(userId, position);
    //Convertir la posición del centro de masa para que
    //coincida con la imagen de profundidad (plano X,Y)
    EToledo.convertRealWorldToProjective(position, position);
    //Establece el color utilizado para dibujar
    //líneas y bordes alrededor de las formas (RGB)
    stroke(0, 0, 255);
    //Establece el ancho del trazo utilizado para líneas,
    //puntos, y el orden en torno a formas
    strokeWeight(5);
    //Establece el color utilizado para llenar formas (RGB)
    //abs es el valor absoluto de un numero
    fill(255,0,0);
    ellipse(position.x, position.y, 25,25);
    //Establece el tamaño de la fuente
    textSize(50);
    //Muestra el ID del usuario
    text(userId, position.x -15, position.y - 15);
  }
}
```

Fuente: elaboración propia.

Figura 152. **Resultado del código del ejemplo 19**



Fuente: elaboración propia.

Como resultado se localiza el centro de masa del usuario. La detección y rastreo del usuario es más rápida con dicho método y se manipulan las interfaces interactivas casi instantáneamente.

5.12.6. Ejemplo 20, objeto 3D y articulación

Finalmente, es posible formar el esqueleto virtual y usar lo visto anteriormente para interactuar con el usuario, habrá capacidad de detección y rastreo de las articulaciones del esqueleto virtual.

Figura 153. **Código del ejemplo 20, parte 1**


```
/*
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Mecánica Eléctrica
Ingeniería Electrónica
Laboratorio de Electrónica
Supervisor: Ing. Byron Arrivillaga
Elaborado por el Auxiliar a cargo: Estuardo Toledo
Segundo Semestre 2015
Versión 2S 2015

Ejemplo 20
Formación del esqueleto
*/

import SimpleOpenNI.*;
SimpleOpenNI EToledo;

void setup() {
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  //Habilita el rastreo del usuario
  EToledo.enableUser();
  size(640, 480);
}
```

Fuente: elaboración propia.

Figura 154. **Código del ejemplo 20, parte 2**

```
//función de dibujo
void draw() {
  EToledo.update();
  //Carga la imagen de profundidad en (X, Y)
  image(EToledo.depthImage(), 0, 0);
  //Vector que almacena la lista de usuarios
  IntVector userList = new IntVector();
  //Escribe la lista de usuarios en nuestro vector
  EToledo.getUsers(userList);
  //Si encontramos al menos un usuario
  if (userList.size() > 0) {
    //Obtener el primer usuario
    int userId = userList.get(0);
    //Si se ha calibrado correctamente se continua
    //con el programa
    if ( EToledo.isTrackingSkeleton(userId)) {
      //Llama a la función para dibujar el esqueleto
      drawSkeleton(userId);
    }
  }
}

void drawSkeleton(int userId) {
  //Establece el color utilizado para dibujar
  //líneas y bordes alrededor de las formas (RGB)
  stroke(0, 0, 255);
  //Establece el ancho del trazo utilizado para líneas,
  //puntos, y el orden en torno a formas
  strokeWeight(8);
```

Fuente: elaboración propia.

Figura 155. **Código del ejemplo 20, parte 3**

```
//funciones que dibujan lineas que unen las
//articulaciones del esqueleto
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_HEAD,
        SimpleOpenNI.SKEL_NECK);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_NECK,
        SimpleOpenNI.SKEL_LEFT_SHOULDER);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_LEFT_SHOULDER,
        SimpleOpenNI.SKEL_LEFT_ELBOW);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_LEFT_ELBOW,
        SimpleOpenNI.SKEL_LEFT_HAND);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_NECK,
        SimpleOpenNI.SKEL_RIGHT_SHOULDER);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_RIGHT_SHOULDER,
        SimpleOpenNI.SKEL_RIGHT_ELBOW);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_RIGHT_ELBOW,
        SimpleOpenNI.SKEL_RIGHT_HAND);

EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_LEFT_SHOULDER,
        SimpleOpenNI.SKEL_TORSO);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_RIGHT_SHOULDER,
        SimpleOpenNI.SKEL_TORSO);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_TORSO,
        SimpleOpenNI.SKEL_LEFT_HIP);
```

Fuente: elaboración propia.

Figura 156. **Código del ejemplo 20, parte 4**

```
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_LEFT_HIP,
        SimpleOpenNI.SKEL_LEFT_KNEE);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_LEFT_KNEE,
        SimpleOpenNI.SKEL_LEFT_FOOT);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_TORSO,
        SimpleOpenNI.SKEL_RIGHT_HIP);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_RIGHT_HIP,
        SimpleOpenNI.SKEL_RIGHT_KNEE);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_RIGHT_KNEE,
        SimpleOpenNI.SKEL_RIGHT_FOOT);
EToledo.drawLimb(userId,
        SimpleOpenNI.SKEL_RIGHT_HIP,
        SimpleOpenNI.SKEL_LEFT_HIP);
//Desactiva el dibujo de la pantalla
noStroke();
//Establece el color utilizado para llenar formas (RGB)
fill(0,0,255);
//Establece el color utilizado para dibujar
//líneas y bordes alrededor de las formas (RGB)
stroke(255, 0, 0);
//Llama a la función que dibuja puntos en
//las articulaciones del esqueleto
drawJoint(userId, SimpleOpenNI.SKEL_HEAD);
drawJoint(userId, SimpleOpenNI.SKEL_NECK);
drawJoint(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER);
drawJoint(userId, SimpleOpenNI.SKEL_LEFT_ELBOW);
drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HAND);
drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER);
drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW);
```

Fuente: elaboración propia.

Figura 157. **Código del ejemplo 20, parte 5**

```
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HAND);
  drawJoint(userId, SimpleOpenNI.SKEL_TORSO);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HIP);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_KNEE);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_FOOT);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HIP);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_KNEE);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_FOOT);
}

void drawJoint(int userId, int jointID){
  PVector joint = new PVector();
  //Poner la posición de la articulación jointID en el vector
  //joint
  //Confidence tiene valores de 0 a 1
  //Si es menor que 1 la articulación no esta presente
  float confidence = EToledo.getJointPositionSkeleton(userId,
                                                      jointID, joint);
  //Si no esta la articulacion no la dibuja
  if(confidence < 0.5){
    return; //Sale de la función
  } else { //En caso contrario la dibuja
      //Convertir la posición de joint a coordenadas
      //que coincidan con la imagen de profundidad (plano X,Y)
      PVector convertedJoint = new PVector();
      EToledo.convertRealWorldToProjective(joint,
                                           convertedJoint);
      ellipse(convertedJoint.x, convertedJoint.y, 20, 20);
    }
}

void onNewUser(SimpleOpenNI curContext,int userId){
  println("ID del usuario: " + userId);
  println("Inicia el rastreo de las articulaciones");
  EToledo.startTrackingSkeleton(userId);
```

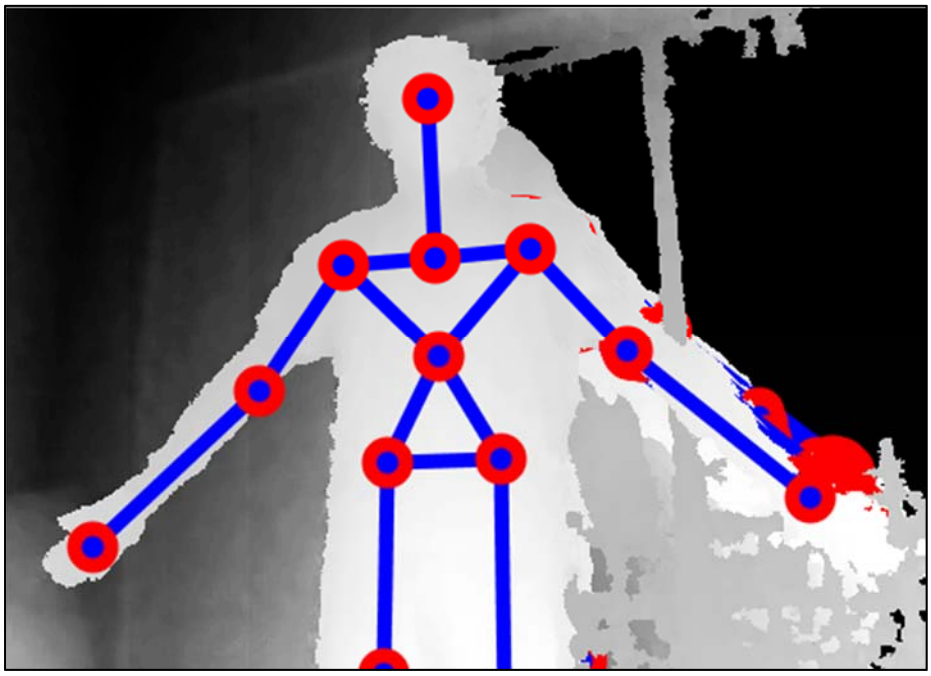
Fuente: elaboración propia.

Figura 158. **Código del ejemplo 20, parte 6**

```
}

void onVisibleUser(SimpleOpenNI curContext,int userId){
  println("Usuario activo, ID: " + userId);
}

void onLostUser(SimpleOpenNI kinect, int userId) {
  println("Usuario no encontrado, ID: " + userId);
}
```

Fuente: elaboración propia.

Figura 159. **Resultado del código del ejemplo 20**



Fuente: elaboración propia.

Como resultado se obtendrá el esqueleto virtual del usuario y con ello se manipulan aplicaciones virtuales y de igual manera hardware por medio de los movimientos del cuerpo.

5.12.7. Ejemplo 21, objeto 3D y articulación

En el ejemplo se controla el encendido y apagado de unos *leds* RGB por medio de la posición de los brazos del usuario.

Figura 160. **Código del ejemplo 21, parte 1**

Fuente: elaboración propia.

Figura 161. **Código del ejemplo 21, parte 2**

```
Microcontralodor
ls /dev/ttyACM0
*/

import SimpleOpenNI.*;
SimpleOpenNI EToledo;
import processing.serial.*;
Serial port;

float Angle1RE;
float Angle1LE;

//-------------------------------------------------Delay
int Ti = 0;
int To = 0;
int Modulo = 0;
int Time = 200;

void setup(){
  size(640, 480);
  EToledo = new SimpleOpenNI(this);
  EToledo.enableDepth();
  EToledo.enableUser();
  EToledo.setMirror(true);
  //-----------------------------------Puerto Serial
  println(Serial.list());
  String portName = Serial.list()[0];
  port = new Serial(this, portName, 9600);
  port.buffer(100);
  Ti = millis(); //--------------------Tiempo inicial
}

void draw() {
  EToledo.update();
```

Fuente: elaboración propia.

Figura 162. **Código del ejemplo 21, parte 3**

```
EToledo.update();
//Carga la imagen de profundidad en (X, Y)
image(EToledo.depthImage(), 0, 0);
//Vector que almacena la lista de usuarios
IntVector userList = new IntVector();
//Escribe la lista de usuarios en nuestro vector
EToledo.getUsers(userList);
//Si encontramos al menos un usuario
if (userList.size() > 0) {
  Modulo = (To - Ti); //cambiar tiempo
  //Obtener el primer usuario
  int userId = userList.get(0);
  //Si se ha calibrado correctamente se continua
  //con el programa
  if (EToledo.isTrackingSkeleton(userId)) {
    //-----------------------------------Posicion 3D
    //Vectores para las posiciones de
    //las articulaciones
    //Mano derecha
    PVector rightHand = new PVector();
    EToledo.getJointPositionSkeleton(userId,
              SimpleOpenNI.SKEL_RIGHT_HAND,
                                  rightHand);
    //Mano izquierda
    PVector leftHand = new PVector();
    EToledo.getJointPositionSkeleton(userId,
                SimpleOpenNI.SKEL_LEFT_HAND,
                                    leftHand);
    //Codo derecho
    PVector rightElbow = new PVector();
    EToledo.getJointPositionSkeleton(userId,
              SimpleOpenNI.SKEL_RIGHT_ELBOW,
                                  rightElbow);
```

Fuente: elaboración propia.

Figura 163. **Código del ejemplo 21, parte 4**

```
//Codo izquierdo
PVector leftElbow = new PVector();
EToledo.getJointPositionSkeleton(userId,
           SimpleOpenNI.SKEL_LEFT_ELBOW,
                                leftElbow);
//Hombro derecho
PVector rightShoulder = new PVector();
EToledo.getJointPositionSkeleton(userId,
       SimpleOpenNI.SKEL_RIGHT_SHOULDER,
                            rightShoulder);
//Hombro izquierdo
PVector leftShoulder = new PVector();
EToledo.getJointPositionSkeleton(userId,
        SimpleOpenNI.SKEL_LEFT_SHOULDER,
                             leftShoulder);
//----------------------------------Posicion 2D
//Convertir la posiciones de las articulaciones
//para que coincidan con la imagen de profundidad
//(plano X,Y)
//Mano derecha
PVector rightHand2D = new PVector(rightHand.x,
                                  rightHand.y);
//Mano izquierda
PVector leftHand2D = new PVector(leftHand.x,
                                   leftHand.y);
//Codo derecho
PVector rightElbow2D = new PVector(rightElbow.x,
                                   rightElbow.y);
//Codo izquierdo
PVector leftElbow2D = new PVector(leftElbow.x,
                                  leftElbow.y);
//Hombro derecho
```

Fuente: elaboración propia.

Figura 164. **Código del ejemplo 21, parte 5**

```
      PVector rightShoulder2D = new PVector(rightShoulder.x,
                                            rightShoulder.y);
      //Hombro izquierdo
      PVector leftShoulder2D = new PVector(leftShoulder.x,
                                           leftShoulder.y);
        //----------------------------Orientacion 2D (Ejes)
      //Poner la orientación de las articulaciones
      //Orientacion del codo derecho respecto
      //al hombro derecho
      PVector OrientationRERS = PVector.sub(rightElbow2D,
                                        rightShoulder2D);
      //Orientacion del codo izquierdo respecto
      //al hombro izquierdo
      PVector OrientationLELS = PVector.sub(leftElbow2D,
                                        leftShoulder2D);
      //--------------------------------Angulos
      //La funcion angleOf usa los 3 datos de los 3 vectores
      //vectores que representan los extremos de la extremidad
      //cuyo ángulo debemos  encontrar y un tercer vector que
      //representa el eje de orientación
      //calculamos el ángulo del codo derecho
      Angle1RE = angleOf(rightHand2D, rightElbow2D,
                                        OrientationRERS);
      //calculamos el ánglo del codo izquierdo
      Angle1LE = angleOf(leftHand2D, leftElbow2D,
                                       OrientationLELS);
    }
  }
  //Llamamos a la comandos
  comandos();
  //Tiempo final
  To = millis();
}

//función para calcular el ángulo entre dos vectores
//respecto a un vector de referencia
float angleOf(PVector one, PVector two, PVector axis) {
  PVector limb = PVector.sub(two, one);
  return degrees(PVector.angleBetween(limb, axis));
```

Fuente: elaboración propia.

Figura 165. **Código del ejemplo 21, parte 6**

```
}

public void comandos(){
  fill(255,0,0);
  scale(3);
  text("Angle rightElbow: " + int(Angle1RE) + "\n" +
  "Angle leftElbow: " + int(Angle1LE), 20, 20);
  //-----------------------------------------------
  if( (Angle1RE >= 170) && (Angle1LE >= 170) ){
    if (Modulo > Time){
      text("Adelante", 20, 60);
      port.write('a');
      port.clear();
      Ti = millis();
    }
  }
  if( (Angle1RE <= 50) && (Angle1LE <= 50) ){
    if (Modulo > Time){
      text("Atras", 20, 60);
      port.write('c');
      port.clear();
      Ti = millis();
    }
  }
  if( (Angle1RE >= 170) && (Angle1LE <= 50) ){
    if (Modulo > Time){
      text("Derecha", 20, 60);
      port.write('b');
      port.clear();
      Ti = millis();
    }
  }
  if( (Angle1RE <= 50) && (Angle1LE >= 170) ){
    if (Modulo > Time){
      text("Izquierda", 20, 60);
      port.write('d');
      port.clear();
      Ti = millis();
    }
```

Fuente: elaboración propia.

Figura 166. **Código del ejemplo 21, parte 7**

```
  }
  if( (Angle1RE >= 90 && Angle1RE <= 130) &&
      (Angle1LE >= 90 && Angle1LE <= 130) ){
    if (Modulo > Time){
      text("Detener", 20, 60);
      port.write('e');
      port.clear();
      Ti = millis();
    }
  }
}

void onNewUser(SimpleOpenNI curContext,int userId){
  println("ID del usuario: " + userId);
  println("Inicia el rastreo de las articulaciones");
  EToledo.startTrackingSkeleton(userId);
}

void onVisibleUser(SimpleOpenNI curContext,int userId){
  println("Usuario activo, ID: " + userId);
}

void onLostUser(SimpleOpenNI kinect, int userId) {
  println("Usuario no encontrado, ID: " + userId);
}
```

Fuente: elaboración propia.

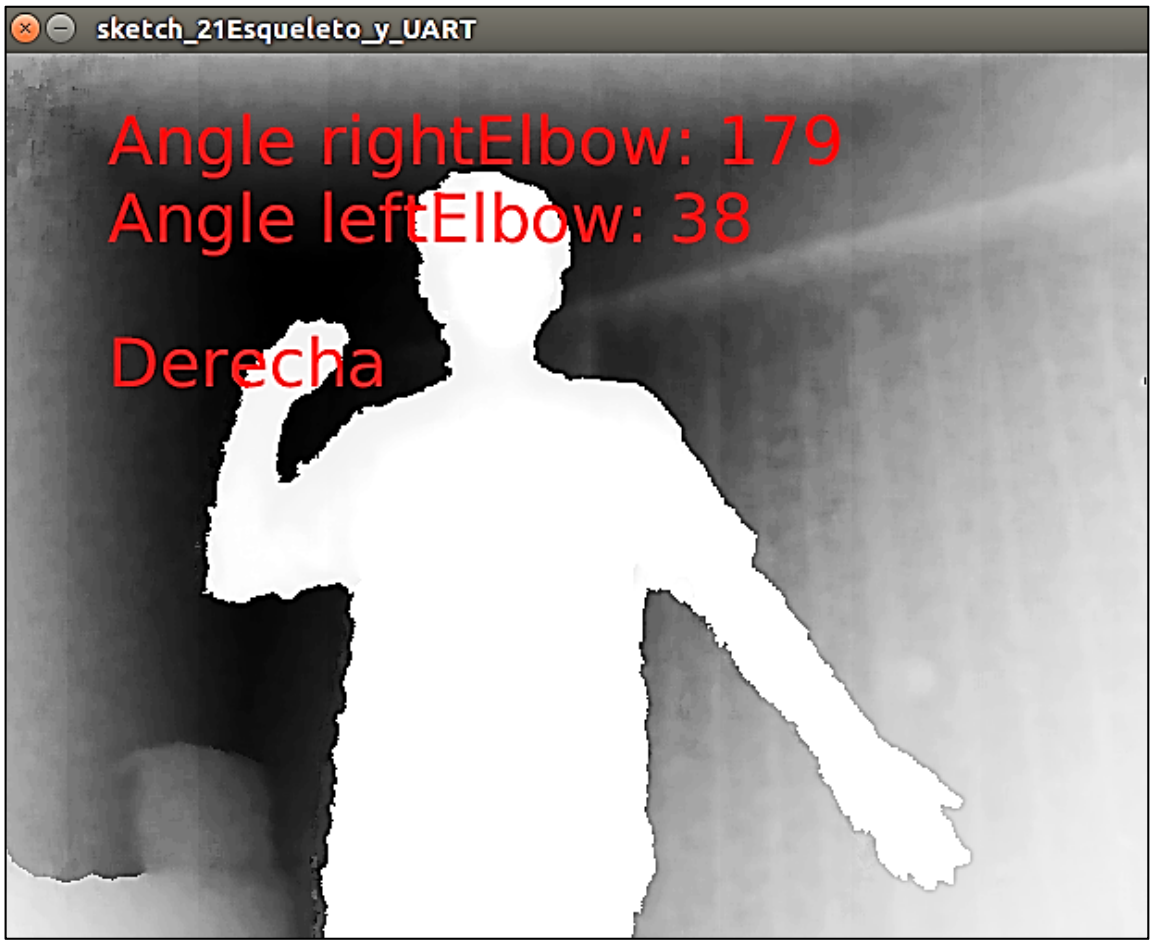
Figura 167. **Resultado del código del ejemplo 21, parte 1**

Fuente: elaboración propia.

Figura 168. **Resultado del código del ejemplo 21, parte 2**

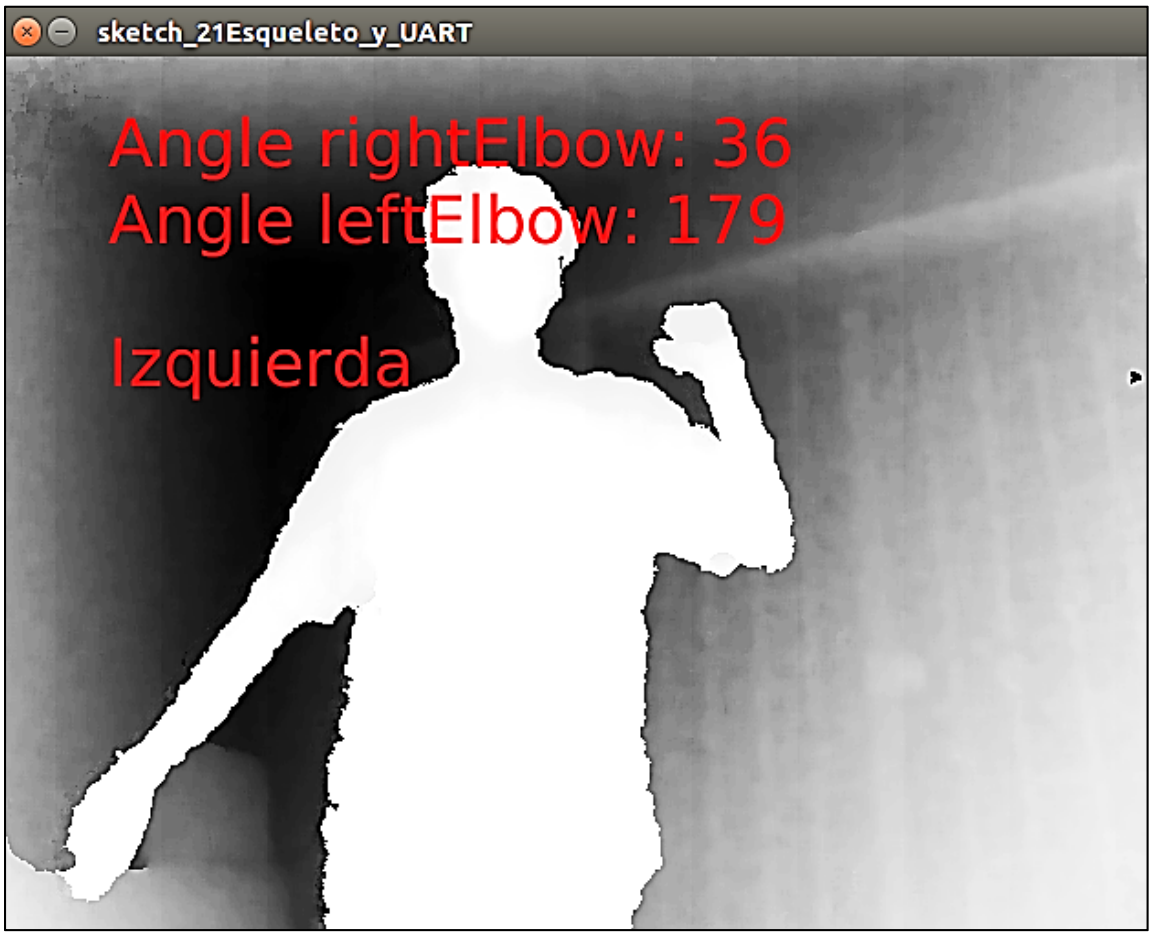
Fuente: elaboración propia.

Figura 169. **Resultado del código del ejemplo 21, parte 3**



Fuente: elaboración propia.

Figura 170. **Resultado del código del ejemplo 21, parte 4**



Fuente: elaboración propia.

Figura 171. **Resultado del código del ejemplo 21, parte 5**

Fuente: elaboración propia.

Como se aprecia en la imagen, es posible controlar el color y el encendido y apagado de los *leds* (hardware), dependiendo del ángulo de los brazos.

El código del microcontrolador msp430 y el resultado del control de los led es el mismo que aparece en el ejemplo 14, a excepción de que resulta más sencillo para el usuario interactuar con el hardware, debido a que el control está en el movimiento de los brazos y no en buscar un cubo en el espacio 3D.

CONCLUSIONES

1. Cuando se trabaja con el software libre se obtienen datos del Kinect con los cuales es posible manipular aplicaciones interactivas y de hardware.

2. Es viable implementar y utilizar los *drivers* y librerías necesarias para controlar y obtener datos del Kinect en el sistema operativo Linux Ubuntu 14.04LTS.

3. Al gestionar el software de programación *processing* se obtiene información de las cámaras del Kinect desde el sistema operativo Linux Ubuntu 14.04LTS.

4. A pesar de que es posible detectar y rastrear las articulaciones del esqueleto que proporciona OpenNI, este no es anatómicamente semejante al fisiológico, debido a las limitaciones del software y, a que solamente se utiliza una parte de la información del esqueleto fisiológico para el desarrollo de aplicaciones interactivas.

5. Si se combinan los datos de la imagen de profundidad y de color, el procesamiento de datos se torna complejo, por lo que la interfaz gráfica del usuario deja de ser interactiva por la lentitud con la que funciona.

6. OpenNI permite procesar la imagen de profundidad para detectar usuarios y localizar la posición de las articulaciones en tres dimensiones. Gracias a ello se controlan aplicaciones a través los

movimientos del cuerpo, midiendo las distancias y los ángulos de las articulaciones virtuales del esqueleto.

7. Las articulaciones que OpenNI puede detectar y rastrear de forma correcta son: la cabeza, el cuello, los hombros, los codos, las manos, el torso, las caderas, las rodillas y los pies. En cuanto a las demás articulaciones no fue posible detectar y rastrear de forma certera debido a las limitaciones del hardware del Kinect.

RECOMENDACIONES

1. Apoyar el software libre como Ubuntu, OpenNI, Processig, entre otros, para seguir trabajando con este y efectuar mejoras en el mismo.

2. Implementar los *drivers* y librerías para el sistema operativo Linux Ubuntu, para obtener datos del Kinect y de esta forma apoyar el uso de software libre.

3. Gestionar el software de programación *processing* para que sus desarrolladores puede mejorar el rastreo y la detección de las articulaciones del esqueleto virtual.

4. Informar de las fallas en el software libre y proponer soluciones, ya que es posible disminuir las limitaciones del Kinect y mejorar la detección y rastreo de los movimientos de las articulaciones del esqueleto virtual.

5. Combinar los datos de la imagen de profundidad y de color cuando se pretenda obtener exploraciones realistas en 3D y para que el procesamiento de datos no sea extenso. Hay que disminuir el número de píxeles que se utilizan de la imagen de profundidad.

6. Atender con especial cuidado la ubicación del Kinect respecto a las fuentes de luz natural, ya que estas contienen todas las longitudes de onda, incluida la infrarroja, por lo se corre el riesgo de generar información errónea a la cámara de infrarrojos.

BIBLIOGRAFÍA

1. GREG, Borenstein. *Making Things See 3D Vision with Kinect, Processing, Arduino, and MakerBot*. 1a ed. O'Reilly Media, 2012. 440 p.

2. Hyperphysics. Centro *de masa*. [en línea]. <http://hyperphysics.phy-astr.gsu.edu/hbasees/cm.html>. [Consulta: 1 de noviembre de 2015].

3. Microteknologias. *¿Qué es exactamente un paquete de software?* [en línea]. <https://microteknologias.wordpress.com/2009/03/13/%C2%BF que-es-exactamente-un-paquete-de-software/>. [Consulta: 1 de noviembre de 2015].

4. Neoteo. *¿Cómo es Kinect por dentro?* [en línea]. <http://www.neoteo.com/como-es-Kinect-por-dentro>. [Consulta: 23 de noviembre de 2014].

5. Processing. *Processing*. [en línea]. <https://processing.org>. [Consulta: 16 de diciembre de 2014].

6. Ramsrigoutham. *Getting started with Kinect on Ubuntu 12.04 OpenNI, Nite, SimpleopenNI and Processing*. [en línea]. <http://ramsrigoutham.com/2012/07/08/getting-started-with-Kinect-on- Ubuntu-12-04-openni-nite-simpleopenni-and-processing/#comment-199>. [Consulta: 28 de agosto de 2014].

7. Tallerdinamo*. Kinect y Processing / Parte 2*. [en línea]. <http://talkingaboutme.tistory.com/560>. [Consulta: 15 de septiembre de 2014].