PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
ESCUELA DE INGENIERIA INFORMATICA

IMPLEMENTACIÓN DE FUNCIONALIDADES MULTITOUCH UTILIZANDO LA TECNOLOGÍA KINECT

NICOLÁS SEBASTIÁN MIRANDA SAAVEDRA
SEBASTIÁN AGUSTÍN SANDOVAL VILLEGAS

INFORME FINAL
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

Diciembre, 2012

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

ESCUELA DE INGENIERIA INFORMATICA

IMPLEMENTACIÓN DE FUNCIONALIDADES MULTITOUCH UTILIZANDO LA TECNOLOGÍA KINECT

NICOLÁS SEBASTIÁN MIRANDA SAAVEDRA
SEBASTIÁN AGUSTÍN SANDOVAL VILLEGAS
Profesor Guía: Iván Mercado Bermúdez
Profesor Co-referente: Cristian Alexandru Rusu

Carrera: Ingeniería de Ejecución en Informática

Dedicatoria

Dedicado a mis padres, profesores y amigos. Agradezco todo el apoyo, conocimiento y paciencia que me han entregado durante todo el proceso de crecimiento y formación en una de las etapas más importantes de mi vida.

Agradezco a mi compañero de proyecto Sebastián Sandoval por todo el esfuerzo, apoyo y confianza que tuvo en mí para llevar a cabo el proyecto y permitiéndome emprender un viaje el cual ha sido de gran relevancia en mi desarrollo profesional.

Nicolás Miranda

Dedicatoria

A mis padres, quienes me han apoyado para poder llegar a esta instancia de mis estudios, ya que ellos siempre han estado presentes para apoyarme moral y psicológicamente.

A mi profesor guía que nos apoyó y orientó durante el desarrollo del proyecto.

A mis amigos con quienes compartí durante mis años de estudios y me apoyaron durante la carrera

A Nicolás Miranda por su apoyo durante la realización del presente proyecto.

Sebastián Sandoval

Resumen

Hoy en día la tecnología ha permitido nuevas formas de interactuar con los dispositivos digitales. Con el éxito y la masificación de la tecnología multitáctil en dispositivos digitales, la interacción humano-computadora se ha vuelto más "natural" y más intuitivo, facilitando el uso de nuevos dispositivos. Este hecho es la principal motivación de este trabajo, que consiste en la ejecución de los gestos multitouch usando la tecnología multitouch óptica proporcionada por el sensor Kinect de Microsoft para interactuar con un ordenador sin dispositivos de puntero. Usando el lenguaje JAVA y el desarrollo basado en prototipos funcionales se desarrollaron dos aplicaciones diferentes con base en los dos paradigmas que presenta el SDK para crear aplicaciones OpenNI con el sensor Kinect. Estas aplicaciones implementan la mayoría de los gestos multitáctiles para controlar un ordenador en diferentes plataformas como Windows de Microsoft, Apple OSX y Linux. El propósito de este trabajo es presentar nuevas maneras de interactuar con otros dispositivos que utilizan las nuevas tecnologías como la tecnología multitouch óptico y también proporcionan una sólida base para nuevas ideas o mejoras para futuros trabajos.

Palabras Claves: Kinect**,** Interacción Natural, Multitouch.

Abstract

Nowadays technology has allowed new ways of interact with digital devices. With the success and massification of multitouch technology in digital devices, human-computer interaction has become more “natural” and more intuitive making easier the use of new devices. This fact is the main motivation of the present work, which consists in the implementation of multitouch gestures using the optical multitouch technology provided by the Microsoft’s KINECT sensor to interact with a computer without pointer devices. Using the JAVA language and prototype-oriented development, two different applications were developed based in both paradigms presented by the OpenNI SDK to create applications with the KINECT sensor. This application implements most of the multitouch gestures to control a computer in different platforms as Microsoft's Windows, Apple's OSX, and Linux. The purpose of this paper is to present new ways to interact with other devices using new technologies as the optical multitouch technology and also provide a solid base to new ideas or improvements for future works.

Keywords: Kinect, Natural Interaction, Multitouch

Tabla de contenido

Índice de Figuras

Índice de Tablas

Introducción

El presente proyecto tiene como propósito mostrar la implementación de las funcionalidades multitouch utilizando el sensor Kinect, sin embargo debido a que el desarrollo de aplicaciones utilizando este dispositivo está en actual crecimiento debido al reciente lanzamiento de los kits de desarrollo oficial por parte de Microsoft y de código libre por parte de otras comunidades como OpenKinect, aún no existen muchos proyectos similares e información detallada de las herramientas que permiten su uso.

De los diferentes SDK disponibles se optó por utilizar OpenNI/NITE para la realización del proyecto, ya que al ser multiplataforma permite la implementación tanto en Windows como en Linux.

OpenNI soporta múltiples lenguajes, y por un tema de conocimiento de y experiencia en programación se optó por el lenguaje Java. Para eso se utilizará un Wrapper de OpenNI que permite el desarrollo de aplicaciones utilizando el sensor Kinect utilizando el lenguaje de programación Java. Otra ventaja de crear la aplicación en Java es para hacer más fácil una posible integración al Sistema Operativo Android.

En los siguientes capítulos se abordará el tema de la implementación y el funcionamiento del Framework de OpenNI y NITE y como se llevó a cabo la implementación de las funcionalidades utilizando la herramienta de desarrollo Processing y la librería SimpleOpenNI.

1.1. Descripción del proyecto

El proyecto se basa en la implementación de las funcionalidades de las pantallas multitouch usando la interfaz de Kinect, el cual es un sensor de movimiento basado en 2 cámaras RGB y un proyector infrarrojo, creado para la consola de Microsoft Xbox 360. Este dispositivo permite controlar e interactuar con los distintos tipos de software desarrollados para consola sin la necesidad de tener contacto físico con un dispositivo de control.

La implementación está basada en la información obtenida del estudio de los diferentes SDK existentes hasta el momento, la cual se llevará a cabo utilizando el Framework de desarrollo de OpenNI, el cual permite hacer uso de todas las capacidades del sensor Kinect y el Framework NITE, que implementa middlewares que permiten el reconocimiento y seguimiento de manos, así como también el rastreo y detección de gestos definidos por el usuario. Para la implementación de las funcionalidades se utilizó el paradigma de reconocimiento esquelético del usuario, trabajando con las coordenadas de los puntos de las manos para realizar las acciones multitouch.

1.2. Objetivo general

Lograr la implementación de las funcionalidades multitouch desarrollando un software o modificando un S.O que permita al usuario controlar e interactuar con un computador utilizando un dispositivo Kinect.

Para lograr este objetivo se deberán llevar a cabo los siguientes objetivos específicos:

1.3. Objetivos y actividades específicas

- Seleccionar los SDK disponibles para desarrollar aplicaciones con Kinect.
- Identificar las herramientas disponibles para la implementación de las funcionalidades multitouch utilizando Kinect.
- Creación de prototipos funcionales incrementales.
- Clasificar los métodos que permiten implementar las funcionalidades multitouch (Handtracking y Skeleton tracking).
- Comparar las diferentes formas de implementar la solución.

1.4. Plan de trabajo

El presente trabajo se realizó en un plazo de 2 semestres, en donde la primera parte consistió básicamente en analizar el problema, estudiar los SDK disponibles y analizar el tipo de solución a crear, para posteriormente comenzar a desarrollar la solución. El detalle de la planificación de puede ver a continuación:

Tabla 1 Planificación

Tarea	Duración de la Tarea
Estudio de la tecnología Kinect	17 días.
Estudio SDK Microsoft Kinect for Windows	5 días.
Estudio SDK libfreenect	7 días.
Estudio SDK OpenNI	7 días.
Estudio Funcionalidades Multitouch	9 días.
Prueba SDK de Microsoft	17 días.
Prueba SDK libfreenect	12 días.
Prueba SDK OpenNI	9 días.
Diseño de prototipo	13 días.
Prueba de prototipo	4 días.
Estudio Middleware NITE	25 días.
Implementación Click Simple	3 días.
Implementación Click Doble	2 días.
Implementación SWIPE UP	2 días.
Implementación SWIP DOWN	2 días.
Implementación SWIPE LEFT	2 días.
Implementación SWIPE RIGHT	1 día.
Implementación Mouse	7 días.
Preparación informe de avance	11 días.
Estudio Herramienta Processing	22 días.
Estudio SimpleOpenNI	8 días.
Implementación usando Processing y rastreo esquelético	17 días.
Implementación usando Processing y rastreo de manos	6 días
Pruebas prototipo usando rastreo esquelético	6 días.

2. Estado del arte

Kinect es un dispositivo creado por Microsoft en conjunto con la empresa israelí PrimeSense, el cual se dio a conocer en el E3 2009 (Electronic Entertainment Expo) bajo en nombre "Proyecto Natal" y fue lanzado el 4 de noviembre del 2010. Luego de su lanzamiento industrias Adafruit ofreció una recompensa para quienes pudieran desarrollar un controlador de código abierto para Kinect, la cual fue otorgada al español Héctor Martín el 10 de noviembre, quien por medio de ingeniería inversa logró desarrollar un controlador para Linux que permite hacer uso de la cámara RGB y funciones de profundidad.

En la actualidad el dispositivo Kinect además de ser utilizado para interactuar con juegos en la consola Xbox 360, se está utilizando como instrumento para desarrollo de software en nuevas áreas de investigación utilizando el controlador desarrollado por Héctor Martín, Libfreenect, lo que ha permitido entregarle nuevos usos a este dispositivo. Es por éste motivo que el 31 de enero del 2012 Microsoft decidió liberar un SDK con los drivers para desarrollar aplicaciones utilizando Kinect con fines comerciales para la plataforma Windows 7, este SDK permite hacer uso de todas las funcionalidades del dispositivo y no solo de las cámaras como es el caso de Libfreenect, y que además incluye documentación detallada, ya sea para programadores novatos, intermedios o expertos, lo que abre nuevas puertas en lo que respecta a innovación en el uso de este dispositivo.

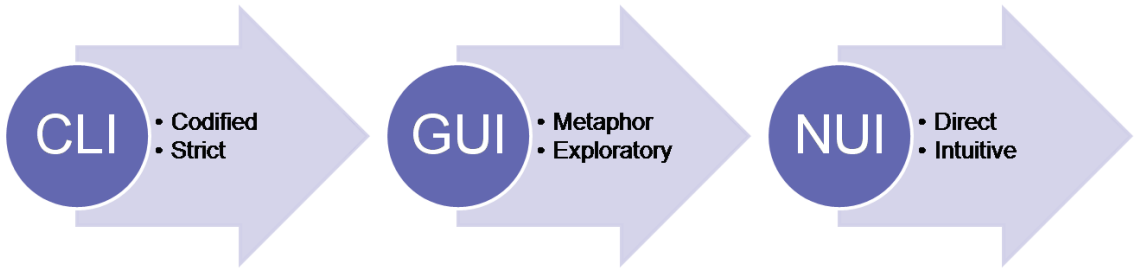
3. Antecedentes

3.1. Interfaces de Interacción Natural (NUI)

A lo largo de la historia han existido diferentes formas de interacción persona-computador, comenzando por las interfaces de línea de comando (CLI), las cuales evolucionaron a las interfaces graficas de usuarios (GUI)

Natural User Interfaces (NUI) es un nuevo paradigma que cambió la forma de interacción persona-computador. La palabra natural es usada porque la mayoría de las interfaces usan dispositivos de control artificiales, cuyo funcionamiento se debe de aprender, por ejemplo, teclado, mouse, puntero laser, etc. Una interfaz de interacción natural (NUI) se basa en que el usuario puede controlar una aplicación o manipular objetos en la pantalla mediante movimientos relativamente naturales o algún tipo de gestos, los cuales son aprendidos rápidamente.

Se basa en las habilidades humanas tradicionales como: tacto, visión, voz, escritura, movimiento y más importantemente los procesos de más alto nivel, como son el conocimiento, la creatividad y la exploración. Algunos ejemplos se interfaces de interacción natural serían las pantallas táctiles, y los dispositivos Kinect y Wavi Xtion.



3.2. Kinect

3.2.1. Introducción

Este capítulo tiene como objetivo presentar al lector un poco de historia sobre la creación y el lanzamiento del dispositivo Kinect creado por Microsoft, así como algunos otros dispositivos de interacción natural creados por otras empresas para la industria de los Videojuegos, como sería el Wii Remote y el PlayStation Move, creados por la compañía Sony para la consola Wii y Playstation respectivamente.

Figura 1 Dispositivo Kinect

3.2.2. Historia

Microsoft invirtió varios años en el desarrollo de la tecnología Kinect, la cual fue anunciada por primera vez el 1 de Junio de 2009 en la Electronic Entertainment Expo 2009[1] bajo el nombre de "Proyecto Natal". El proyecto estuvo a cargo de Alex Kipman y fue desarrollado por Microsoft para la consola de videojuegos Xbox360.

El lanzamiento oficial en Estados Unidos se llevó a cabo el 4 de noviembre de 2010 y el lanzamiento para Europa se llevó a cabo el 10 de noviembre de 2010. Además fue premiado con el Récord Guinness por ser el dispositivo electrónico que más rápido se vendió, llegando a los 8 millones de unidades vendidas en los primeros 60 días[2].

La pieza angular sobre la que funciona Kinect se basa en un diseño y una tecnología creada por la empresa israelí PrimeSense[3], ya que el sensor Kinect se basa en la tecnología 3D creada por ésta empresa. PrimeSense trabaja en conjunto Asus, empresa con la cual sacó al mercado un dispositivo similar a la Kinect llamado Wavi Xtion, ya que se basan en la tecnología creada por la misma empresa, pero Wavi Xtion fue pensada para ser utilizada en PC.

[1] http://es.wikipedia.org/wiki/Electronic_Entertainment_Expo_2009
[2] http://www.xbox.com/en-us/press/archive/2011/0308-ten-million-kinects
[3] http://www.primesense.com/

PrimeSense y ASUS también están trabajando juntos para promover y apoyar la comunidad de desarrolladores OpenNI, una organización sin fines de lucro formada para promover y certificar la compatibilidad e interoperabilidad de dispositivos, aplicaciones y middleware para la Interacción Natural (NI). Cabe recordar que OpenNI fue la responsable de liberar el framework que proporciona la interfaz para los sensores NI de audio y video, o también conocido como el driver Open Source oficial de Kinect.

Figura 2 Logo OpenNI

Figura 3 Logo PrimeSense

3.2.3. Especificaciones de Hardware

Como se puede ver en la Figura 4, Kinect está formado por diferentes componentes y tecnologías, los cuales ofrecen al usuario una nueva experiencia que envuelve diferentes sentidos.

Figura 4 Componentes Kinect

Físicamente el sensor Kinect es una barra horizontal, de aproximadamente 23 centímetros, la cual está conectada a una pequeña base circular que posee un motor que

permite el movimiento del sensor en el eje vertical aproximadamente unos 27º. El sensor está hecho para ser colocado longitudinalmente por encima o debajo de una pantalla de video.

Posee además 2 cámaras y un proyector infrarrojo, como se puede apreciar en la Figura 3. El primer componente de la izquierda es el proyector infrarrojo, el componente central es un Color Complementary Metal Oxide Semiconductor (CMOS), es decir una simple cámara RGB con una resolución de 640x480, 32bits de color y 30fps y finalmente el componente de la derecha es el IR CMOS o receptor IR con una resolución de 320x240, este sensor permite ver la habitación en 3D bajo cualquier condición de luz ambiental.

Kinect posee además en la parte inferior un micrófono multiarray capaz de detectar voces y eliminar el sonido ambiente. El array del micrófono tiene cuatro cápsulas, y opera con cada canal procesando 16-bit de audio con un ratio de frecuencia de 16 kHz.

Para el correcto funcionamiento de las cámaras y sensores de Kinect se requiere de un espacio ideal de:

- 1.2m a - 3.5m de distancia entre el objetivo y el sensor Kinect.
- Campo de visión horizontal de 58º.
- Campo de visión vertical de 45º.

Figura 5 Dispositivo Kinect

Este es un diagrama de PrimeSense (Figura 6) que da una referencia de cómo funciona su plataforma. Kinect, es la primera, y hasta el momento la única, implementación de esta plataforma.

Una cámara y un transmisor infrarrojo proveen la entrada para crear un mapa de profundidad, mientras que otra cámara detecta el espectro visual humano con una resolución de 640x480, éstas 2 se mezclan para formar una imagen 3D de la habitación.

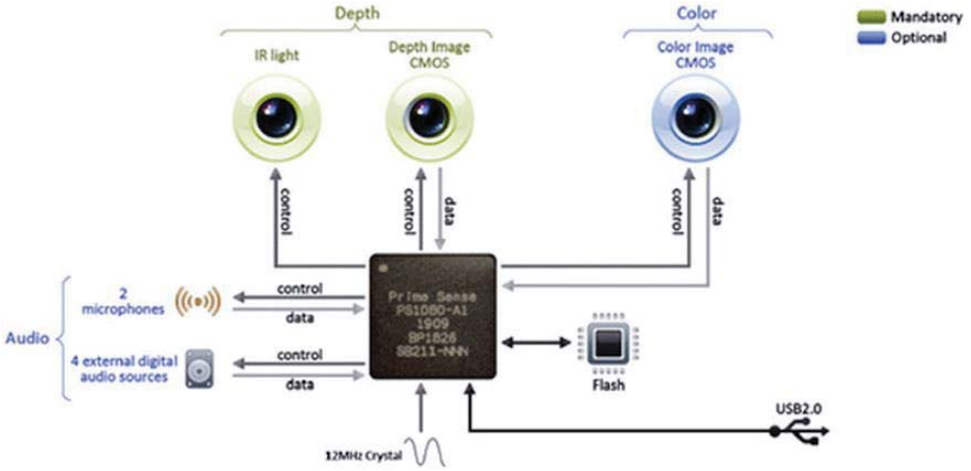


Figura 6 Diagama Kinect

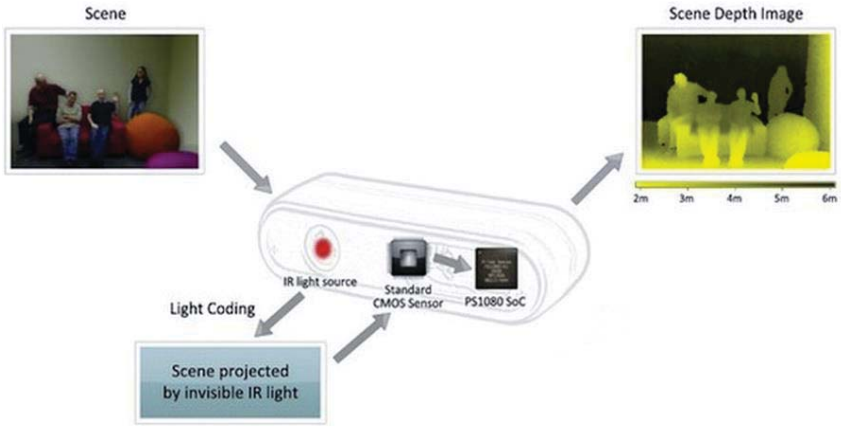


Figura 7 Sensor de Profundidad Kinect

La siguiente es una tabla resumen con las principales características del sensor Kinect.

Tabla 2 Especificaciones

Item	Especificación
Angulo de visión	43º vertical y 58º horizontal.
Movilidad del Motor	27º verticalmente.
Sensor de profundidad IR	QVGA Resolución 320x240.
Cámara a color RGB	VGA Resolución 640x480
Frame Rate	30fps (Sensor de profundidad y cámara a color).
Formato de audio	16 bits de audio con un radio de frecuencia de 16kHz

3.3. Dispositivos Multitouch

Desarrollada en el año 1982, en la Universidad de Toronto y los Laboratorios Bell. La Tecnología Multitouch, es una técnica de interacción persona-computador, en la que generalmente utilizan pantallas táctiles o touchpad, y su principal característica es que reconocen simultáneamente múltiples puntos de contactos. Los distintos tipos de superficies táctiles de clasifican según su funcionamiento en:

- **Superficies Resistivas:** Las cuales están compuestas por una superficie flexible sensible a la presión, la que se activa según un determinado nivel de presión sobre ella.

- **Superficies Capacitivas:** Sobre las cuales poseen una determinada carga eléctrica, la cual es alterada en una coordenada especifica dentro del área de la superficie cuando es tocada por un dedo, el cual también posee carga eléctrica, esta señal es enviada al software para activar o reconocer la acción realizada sobre la superficie.

- **Dispositivos ópticos:** Como sensores de movimiento (Kinect – ASUS Xtion).

El primer dispositivo comercial con esta tecnología apareció el año 2005 y se trató de un controlador multimedia profesional de la empresa francesa JazzMutant, sin embargo esta tecnología es muy común verla en dispositivos móviles como Tablet PC o Smartphones.

Funciones multitouch básicas a implementar utilizando Kinect:

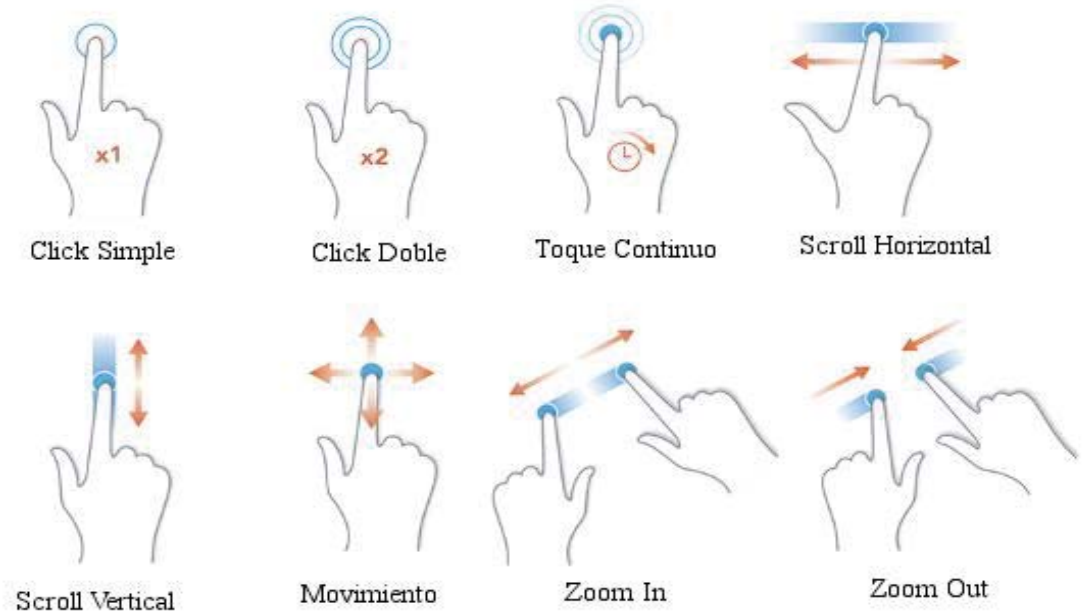


Figura 8 Funcionalidades Multitouch

4. Análisis y diseño

4.1. Definición de Requerimientos

4.1.1. Funcionales

Es importante destacar el hecho de que al crear una interfaz para interactuar con un computador por medio de gestos multitouch ópticos, implica que las acciones llevadas a cabo por la aplicación están limitadas a las características bajo las cuales fue diseñado el sistema operativo en uso por el computador. Es decir que las acciones a llevar a cabo dependen de la aplicación que este en uso por el usuario. Bajo este hecho los requerimientos que deben cumplir ambas aplicaciones desarrolladas se presentan a continuación:

- La aplicación debe reconocer al usuario en el momento en que este entra al campo de visión del sensor KINECT.
- La aplicación debe mantener la sesión activa y estable mientras este en uso por el usuario.
- La aplicación sólo debe reconocer 1 usuario activo.
- La aplicación debe reconocer los siguientes gestos multitouch:
    - Click (Simple y doble)
    - Push
    - Scroll Horizontal y Vertical
    - Pan
    - Zoom in y Zoom out.
- La aplicación debe permitir la ejecución de acciones asociadas al mouse y teclado en el sistema operativo en uso al detectar los gestos antes mencionados.
- La aplicación no debe reconocer falsos-verdaderos al detectar gestos ejecutados de formas incorrectas o diferentes a los antes mencionados por parte del usuario.
- La aplicación no debe reconocer a más de un usuario de forma simultánea.

4.1.2. No Funcionales

- El sistema debe funcionar con el usuario posicionado de pie y a 1.5mts mínimo de distancia del sensor Kinect en el caso de utilizar la aplicación con rastreo esquelético.
- El sistema debe funcionar con el usuario posicionado a 1 metro de distancia en caso de utilizar la aplicación con rastreo de manos.

4.1.3. Restricciones

- Ambas aplicaciones funcionan solo bajo sistemas de 64 bits.
- Se deben tener los drivers de OpenNI de 64 bits instalados para que funcionen las aplicaciones.

4.2. Escenarios

4.2.1. Aplicación basada en rastreo de manos

Los presentes escenarios están asociados a la aplicación basada en el rastreo de manos:

Nombre Escenario	1- Comenzar a utilizar la aplicación (Iniciar sesión)
Actor	Usuario
Requisitos	-----
Flujo de eventos	1. El usuario debe posicionarse al frente del sensor KINECT a una distancia mínima de 1 metro. 2. Posicionar una mano con la palma hacia el frente y los dedos abiertos para iniciar sesión y controlar el puntero del mouse (La primera mano detectada es la mano de control). 3. Por último posicionar la otra mano con la palma hacia el frente con los dedos abiertos para poder realizar acciones (La última mano detectada es la mano de acciones).

Nombre Escenario	2- Controlar el puntero del mouse
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para mover el puntero del mouse es necesario mantener la mano de control con los dedos abiertos y moverla en la dirección deseada.

Nombre Escenario	3- Realizar Doble Click
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para realizar la acción de doble click, se debe acercar y alejar la mano de acciones con la palma hacia el sensor y los dedos abiertos a una velocidad moderada.

Nombre Escenario	4- Realizar Click Derecho
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para activar el click derecho, se debe acercar y alejar la mano de acciones con la palma hacia el sensor y los dedos juntos a una velocidad moderada.

Nombre Escenario	5- Activar Zoom (In - Out)
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para activar el modo Zoom, se deben juntar los dedos de ambas manos. 2- Luego, para hacer Zoom-in se deben alejar las manos con las palmas hacia el sensor y los dedos juntos. 3- Zoom-out se activa cuando se acercan las manos con las palmas hacia el sensor y los dedos juntos. 4- Para desactivar el modo zoom solo basta con abrir los dedos de las manos.

Nombre Escenario	6- Activar scroll
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para realizar scroll, se deben mantener juntos los dedos de la mano de control y luego mover la mano hacia la dirección deseada. 2- Para desactivar el modo scroll solo basta con abrir los dedos de la mano de control.

Nombre Escenario	7- Activar pan
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- La función Pan o Selección es activada cuando los dedos de la mano de acción se mantienen juntos. 2- Al escuchar el sonido de activación se puede mover la mano de control para seleccionar los objetos deseados. 3- Para desactivar esta funcionalidad solo basta con abrir los dedos de la mano de acciones.

Nombre Escenario	8- Activar Swipe
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- La función swipe es activada cuando los dedos de ambas manos están abiertos, y la mano de acciones se mueve en una dirección específica. (arriba, abajo, izquierda o derecha).

4.2.2. Aplicación basada en rastreo esquelético

Los presentes escenarios están asociados a la aplicación basada en el rastreo esquelético:

Nombre Escenario	1- Comenzar a utilizar la aplicación (Iniciar sesión)
Actor	Usuario
Requisitos	-----
Flujo de eventos	1. El usuario debe posicionarse al frente del sensor KINECT a una distancia mínima de 1,5 metros. 2. El usuario debe hacer una pose de calibración. 3. Mantener la pose de calibración hasta que se detecte correctamente el esqueleto del usuario. 4. El usuario por defecto puede mover el puntero con la mano izquierda y realizar un click simple o click continuo con la mano derecha.

Nombre Escenario	2- Controlar el puntero del mouse
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para mover el puntero del mouse una vez calibrado el esqueleto solo se debe mover la mano izquierda. El puntero del mouse comenzara a seguir el movimiento de la mano.

Nombre Escenario	3- Realizar Click simple
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para realizar la acción de click simple, se debe acercar la mano y retirar la mano derecha hacia el sensor a una velocidad moderada.

Nombre Escenario	4- Realizar Click continuo
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para realizar la acción de click continuo, se debe acercar la mano derecha hacia el sensor y mantenerla presionada.

Nombre Escenario	5- Realizar Doble Click
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Primero se debe activar la acción de doble click posicionando la mano derecha por detrás de la cadera hasta que se escuche un sonido. 2- Para realizar la acción de doble click, se debe acercar y alejar la mano derecha del sensor a una velocidad moderada. 3- Para desactivar el Doble Click solo se debe posicionar la mano derecha detrás de la cadera.

Nombre Escenario	6- Activar Zoom (In - Out)
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Primero se debe activar la acción de Zoom posicionando ambas manos por detrás de la cadera hasta que se escuche un sonido. 2- Para realizar el Zoom se deben acercar ambas manos al sensor (como realizando un Click) y posteriormente se deben juntar o separar ambas manos una distancia mínima de 5cms. 3- Si el usuario separa las manos se producirá un Zoom In. 4- Si el usuario junta las manos se producirá un Zoom Out. 5- Para desactivar el Zoom solo basta con posicionar ambas manos detrás de la cadera hasta que se escuche un sonido.

Nombre Escenario	7- Activar scroll
Actor	Usuario
Requisitos	Iniciar sesión
Flujo de eventos	1- Para activar el scroll el usuario debe posicionar la mano izquierda detrás de la cadera. 2- Para realizar scroll, se deben se debe mantener presionada la mano izquierda contra el sensor (como haciendo un click) y luego mover la mano hacia arriba o abajo para el scroll vertical o izquierda y derecha para el scroll horizontal. 3- Para desactivar el scroll solo basta posicionar la mano izquierda detrás de la cadera.

5. Software de desarrollo para Kinect

Actualmente existen distintos kits de desarrollo (SDK) que permiten desarrollar aplicaciones utilizando Kinect en distintas plataformas. En este apartado se analizarán los SDK disponibles que pueden ayudar a cumplir con el objetivo final del proyecto, estudiando las ventajas y limitaciones de cada uno.

5.1. Microsoft Kinect for Windows:

5.1.1. Introducción

SDK oficial de Microsoft para crear aplicaciones con Kinect sobre la Plataforma Windows 7 y Windows Embedded Standard 7, actualmente en su versión 1.0 lanzada en febrero del 2012, presenta mejorías sobre su versión beta, la cual solo soportaba el hardware Kinect para Xbox 360, la versión actual soporta el hardware Kinect para Windows como también el disponible para Xbox 360, permitiendo además la conexión de hasta 4 sensores Kinect en un mismo computador. En el aspecto legal Microsoft liberó este SDK permitiendo el uso comercial de las aplicaciones que usen el Hardware Kinect.

5.1.2. Características

- Soporte para lenguajes C++, C#, o Visual Basic (Utilizando Microsoft Visual Studio 2010)
- Mejoras en el Rastreo esquelético (Skeleton Tracking)
- Permite el uso del sensor de profundidad para detectar objetos hasta un mínimo de 40 centímetros de distancia, entregando más información sobre los valores de profundidad detectados.
- Provee componentes para reconocimiento de voz más exacto (Actualmente inglés).

Requerimientos Hardware:

- Procesador 32-bit (x86) o 64-bit (x64).
- Procesador Dual-Core 2.66 GHz o superior.
- USB 2.0 dedicado.
- 2 GB RAM

Requerimientos Sistema Operativo:

- Windows 7 o Windows Embedded Standard 7

5.1.3. Capacidades

- **Acceso a la información en bruto:** Permite acceder al flujo de datos sin procesar generado por los diferentes sensores, ya sea sensor de profundidad, cámara RGB, micrófonos multiarray.
- **Rastreo Esquelético:** Permite el rastreo del esqueleto de uno o más usuarios posicionados dentro del campo de visión de la Kinect. Permite el desarrollo de aplicaciones basadas en gestos.
- **Procesamiento de audio:** Permite el procesamiento de audio usando los 4 micrófonos posicionados en la parte inferior de la Kinect. Provee supresión de ruido, cancelación de eco y detectar la posición de la fuente que emite el sonido entre otras cosas.

5.2. Libfreenect

Librería de código libre creada a partir del "Hackeo" de la Kinect a los 3 días de su lanzamiento, actualmente es mantenida por una comunidad abierta de usuarios llamada OpenKinect. Esta librería provee los drivers para el acceso a la cámara USB del sensor Kinect permitiendo procesar la información captada por esta:

- RGB e Imágenes de Profundidad.
- Motores.
- Acelerómetro.
- LED.
- Actualmente no soporta audio.

Libfreenect posee interfaces (Wrappers) para múltiples lenguajes:

- C (using a synchronous API).
- Python.
- Actionscript.
- C#.
- Java (JNA).
- Lisp.
- C++.

Requerimientos

Para los driver:

- libusb-1.0 >= 1.0.3 (*nix- OS X).
- libusb-win32 (Windows).
- Cmake >= 2.6 (Todas las plataformas).

Para GLview:

- OpenGL.
- Glut.
- Pthreads (pthread-win32 para Windows).

Libfreenect es multiplataforma, pudiendo ejecutarse en Windows, Linux y Mac OS-X.

5.3. OpenNI[4]

5.3.1. Introducción

Framework basado en OpenNI (***Open Natural Interaction*,** es una organización sin fines de lucro enfocada en el desarrollo de tecnologías para la interacción natural con dispositivos. Uno de sus principales participantes es PrimeSense, una empresa israelí responsable del desarrollo del Kinect y dispositivos similares junto con la firma Asus), que provee una infraestructura genérica basada en API's de código abierto para acceder a los dispositivos de interacción natural, permitiendo acceder específicamente a los servicios provistos por el sensor Kinect. Adicionalmente esta empresa desarrolla el ***middleware*** NITE el cual no es de código abierto y provee los algoritmos necesarios para permitir el rastreo y seguimiento tanto del rastreo esquelético como del rastreo de manos.

5.3.2. Características

OpenNI soporta los siguientes lenguajes:

- C++.
- C#.
- Python.
- Visual Basic.
- Java.

5.3.3. Módulos[5]

SensorKinect utiliza el driver avin2 para conectar el sensor Kinect para utilizarlo con el Framework provisto por OpenNI, permitiendo acceder a distintos módulos de sensores:

- Sensores 3D (en algunos dispositivos).
- Cámaras RGB.
- Cámaras IR (Infrarrojas).
- Dispositivos de Audio.

[4] http://www.openni.org/
[5] http://openni.org/Documentation/ProgrammerGuide.html

Además de los siguientes módulos Middleware soportados:

- **Middleware de rastreo de cuerpo completo:** Componente de software que procesa la información captada por los sensores para generar un modelo del cuerpo humano (Orientación, centro de masa, articulaciones).
- **Middleware de análisis de Puntos de la mano:** Componente de software que procesa la información recopilada por los sensores para generar la posición de la mano.
- **Middleware de detección de gestos:** Componente de software que permite reconocer gestos y generar alertas.
- **Middleware de analizador de escena:** Componente de software que analiza la imagen de una escena para generar cierto tipo de información como:
    - Coordenadas del plano.
    - Identificación de figuras individuales en la escena.
    - Separación entre un objeto y el fondo.

5.3.4. Capacidades

- **Vista alternativa:** Permite a cualquier tipo de generador de mapas (profundidad, imagen, infrarrojo) transformar su información para aparentar como si el sensor estuviese posicionado en otra posición.
- **Sincronización de frame:** Permite que 2 sensores que produzcan información (por ejemplo profundidad e imagen) sincronizar sus frames para que arriben al mismo tiempo.
- **Espejo:** Permite reflejar la información producida por un sensor. Por ejemplo si el usuario mueve la mano izquierda el objeto en la escena moverá la mano derecha.
- **Detección de pose:** Permite detectar si el usuario está realizando alguna pose específica.
- **Esqueleto:** Habilita la generación de esqueleto del usuario.
- **Posición de usuario:** Permite identificar la posición del usuario dentro de la escena utilizando la información del mapa de profundidad.
- **Estado de error:** Habilita el nodo que reporta si es que se está en un estado de error.

5.3.5. Estructura

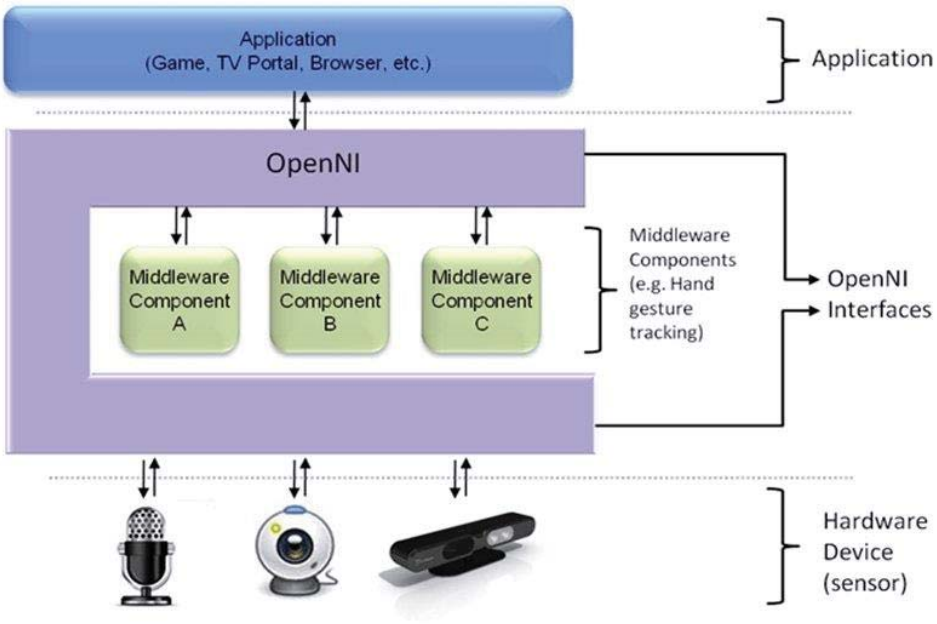


Figura 9 Estructura OpenNI

- La capa superior representa las aplicaciones que hacen uso de la interacción natural provistas por OpenNI
- La capa del medio representa a OpenNI, la cual provee interfaces de comunicación para comunicarse con los sensores y con los componentes middlewares para analizar la información de los sensores.
- La capa inferior muestra los dispositivos de hardware que capturan información, ya sea visual o de audio desde la escena.

5.4. NITE[6]

5.4.1. Introducción

Natural Interaction Technology for End-user NITE. Es el middleware que percibe el mundo en 3D, basado en las imágenes de profundidad captadas por los sensores, traduce estas percepciones en información significativa de la misma forma en que la gente lo haría.

[6] http://www.primesense.com/

NITE se basa en 2 paradigmas:

- Control basado en el uso de las manos.
- Rastreo de cuerpo completo.

Este middleware se compone de un motor de visión computarizada y de un framework que provee a las aplicaciones de controles para la interfaz de usuario (UI), específicamente estas capas son:

- Algoritmos NITE
- Controles NITE

Cada uno de estos componentes se explica a continuación.

5.4.2.Control basado en el uso de las manos (Hand Control)

Permite el control de una aplicación utilizando gestos con las manos, por ejemplo un gesto de saludo (Hand Wave). Este paradigma posee 2 estados:

- Modo de detección para gestos de enfoque: Este modo, el cual se activa cuando no hay usuarios en el campo de visión del sensor, permite que el sistema observe el ambiente o escenario en espera de que un usuario solicite el control de la aplicación por medio de un gesto de enfoque, por ejemplo un saludo.

- Modo de control: Esta modalidad es activada cuando un usuario ya tiene el control de la aplicación, por lo que el sistema rastrea sus manos en la espera de gestos específicos para realizar acciones dentro de la aplicación.

5.4.3.Rastreo de cuerpo completo

Este paradigma fue pensado para el uso de juegos, en donde la posición del usuario permite ejecutar acciones dentro del juego. Para este propósito es usado el rastreo esquelético, en donde a través de la detección de las articulaciones es posible detectar posiciones para ser usadas dentro de una aplicación.

A continuación se presenta una imagen que muestra las articulaciones del cuerpo humano detectadas por el sensor Kinect:

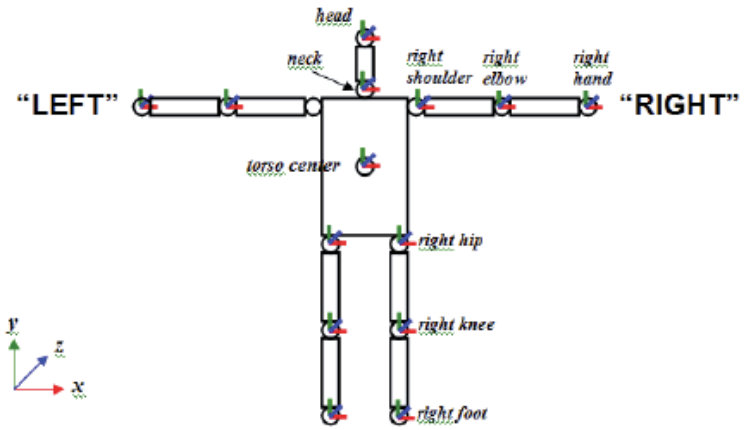


Figura 10 Articulaciones reconocidas

5.4.4. Algoritmos NITE[7]

Corresponde a la capa de bajo nivel del middleware, la cual está encargada de procesar las imágenes de profundidad capturadas por el sensor. Para llevar a cabo este proceso, esta capa utiliza algoritmos de visión computarizada, los cuales permiten:

- Segmentación del escenario: Es el proceso por el cual se separa a los usuarios y objetos en movimiento del fondo del escenario, permitiendo asociarles un TAG para su identificación.

- Detección y rastreo de manos: Proceso por el cual son detectadas las manos del usuario por medio de un gesto específico para luego ser rastreadas en la espera de otros gestos para llevar ejecutar acciones.

- Rastreo de cuerpo completo: Este proceso funciona en base a los resultados del algoritmo de segmentación del escenario, permitiendo el rastreo de los cuerpos de los usuarios para calcular la posición de sus articulaciones las cuales son utilizadas por la aplicación.

5.4.5. Controles NITE[8]

Es la capa del framework de aplicación que analiza los puntos de las manos de los usuarios generadas por los algoritmos NITE, entregando las herramientas necesarias para manejar el flujo de la aplicación de acuerdo a los movimientos de las manos, identificando gestos específicos y entregando un conjunto de controles para la interfaz de usuario basados en los gestos antes mencionados.

[7] http://pr.cs.cornell.edu/humanactivities/data/NITE.pdf
[8] http://andrebaltazar.files.wordpress.com/2011/02/nite-controls-1-3-programmers-guide.pdf

5.4.6. Gestión de sesión

En todas las aplicaciones que utilicen controles NITE se debe gestionar la sesión de usuario, la cual es un estado en que el sistema espera un gesto específico luego de que el usuario ganó el control de la aplicación con un gesto de enfoque, en otras palabras el usuario tiene el control de la aplicación usando sus manos.

NITE maneja tres estados de sesión:

- **Sesión no iniciada** (Not in Session): En la cual no hay una sesión activa, por consecuencia el sistema se encuentra en modo de detección para gestos de enfoque, y una vez que el sistema reconoce un gesto de enfoque, el estado cambia a "En Sesión".

- **En Sesión** (In Session): En este estado el usuario tiene el control de la aplicación usando sus manos, y el sistema espera un gesto específico para llevar a cabo una acción dentro de la aplicación.

- **Re-Enfoque rápido** (Quick Refocus): Este estado se alcanza cuando la mano del usuario sale del campo de visión del sensor dentro de una sesión activa, en sí es un estado intermedio entre los estados "En Sesión" y "Sesión no Iniciada" que permite volver a retomar la sesión activa, por un tiempo determinado, con un gesto específico simple antes de finalizarla.

A continuación se presenta el diagrama de flujos para los distintos estados de NITE:

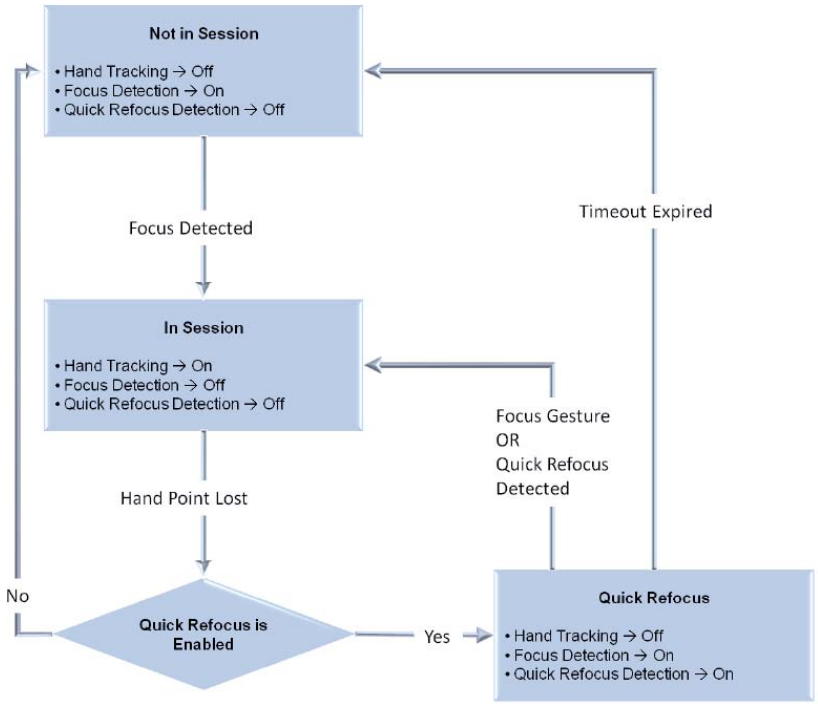


Figura 11 Diagrama de flujos de NITE

5.4.7. Gestos de Control

Para que los gestos realizados por los usuarios sean reconocidos correctamente se asume lo siguiente:

- La mano que controla la aplicación está dentro del campo de visión del sensor, y ésta no es obstruida por otros usuarios u objetos. Si la mano sale del campo de visión del sensor, se perderá el control de la aplicación.

- La distancia entre el usuario y el sensor es de 1mt a 3,5mt.

Al utilizar una aplicación de interacción natural, lo primero que se debe realizar es obtener el control de esta (Iniciar un sesión) por medio de un gesto de enfoque. NITE permite el uso de dos gestos de enfoque para ganar el control de una aplicación, los cuales son:

- **Hand Wave** (Saludo): el cual debe consistir a lo mínimo de 4 o 5 movimientos horizontales, ya sea de derecha a izquierda o viceversa.
- **Click**: Este gesto de enfoque consiste en acercar la mano hacia el sensor y alejarla inmediatamente de este, el movimiento debe ser de al menos 10cms y debe ser ejecutado estando de frente al sensor.

Los gestos soportados por NITE para el manejo de aplicaciones son:

- **Click**: Consiste en acercar la mano hacia el sensor y alejarla inmediatamente de éste.
- **Wave**: Consiste como mínimo de 4 o 5 movimientos horizontales.
- **Push**: Similar al Click, pero se debe mantener la mano cerca del sensor un cierto tiempo antes de retirarla para que el gesto sea reconocido como tal.
- **Swipe**: Es un movimiento específico en una cierta dirección, ya sea hacia arriba, abajo, izquierda o derecha.
- **Circle**: Se debe formar un círculo de 40cms de diámetro como mínimo para que sea reconocido.
- **Raise Hand**: Consiste en levantar la mano.

Para el correcto reconocimiento de estos gestos se recomienda:

- Mantener las manos alejadas del cuerpo y de otros objetos.
- Mantener las manos abiertas al momento de realizar los gestos.
- Los movimientos no deben ser ni muy rápidos o muy lentos.

5.4.8.Estructura

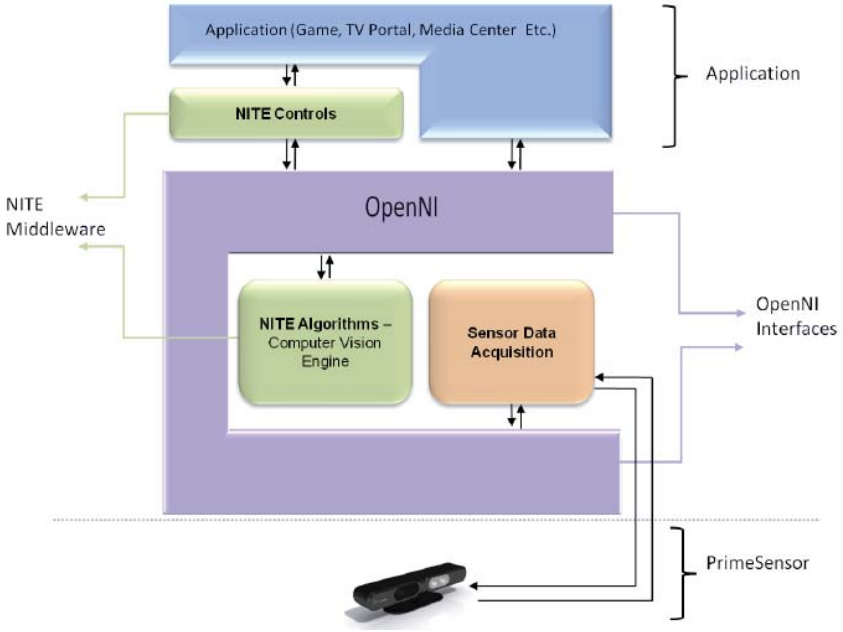


Figura 12Estructura NITE

- La capa inferior corresponde a los sensores físicos que entregan datos a ser interpretados (RAW Data).
- La capa superior (Morada) corresponde a OpenNI, la cual es una infraestructura, que una vez instalada en un host (PC u otro dispositivo como una Tablet por ejemplo), provee interfaces para la comunicación con los drivers de los sensores y con el middleware que analiza y procesa los datos generados por los sensores.
- El módulo de adquisición de datos de los sensores es una API que permite al host trabajar con los sensores conectados.
- Las capas de algoritmos y control NITE componen el middleware NITE conectado a la infraestructura OpenNI, el cual procesa las imágenes de profundidad producidas por los sensores y además provee la identificación de gestos y controles para interfaz de usuario basada en gestos de acuerdo a los datos capturados y procesados por los algoritmos NITE.
- Por último la capa superior corresponde a la aplicación de interacción natural, la cual utiliza los controles NITE y por medio de la infraestructura OpenNI puede acceder a los datos generados por los algoritmos NITE e incluso a los datos (RAW Data) generados por los sensores.

5.5. Comparación entre Kinect for Windows / OpenNI

Se analizaron 2 SDK para Kinect durante el desarrollo del proyecto, el SDK oficial de Microsoft y OpenNI el cual es OpenSource. Las ventajas y desventajas encontradas se listan a continuación.

Tabla 3 Tabla comparativa

SDK	Ventajas	Desventajas
Microsoft Kinect For Windows	• Soporta audio. • Soporta Manejo de motor. • Rastreo de cuerpo completo. • No necesita pose de calibración. • Detecta cabeza, manos, pies y clavícula. • Soporta múltiples sensores. • Documentación detallada para programadores de distinto nivel.	• Solo rastrea cuerpo completo, no posee modo de rastrear solo las manos. • Solo calcula las posiciones de las articulaciones, no las rotaciones. • Alto consumo de CPU. • No posee sistema de reconocimiento de gestos. • Solo soporta Windows 7 (x86 y x64).
OpenNI/NITE	• Incluye soporte para rastreo de manos y gesticulaciones. • Calcula la rotación de las articulaciones. • Bajo consumo de CPU. • Soporta sensores Primesense y Asus WAVI Xtion. • Provee soporte para la conexión de múltiples sensores. • Multiplataforma: Windows (7, XP, Vista), Linux y Mac OSX. • Soporte para grabar/reproducir desde/hacia el HDD. • Provee eventos para detectar cando un nuevo usuario entra al rango de visión del sensor.	• No soporta Audio. • No soporta manejo del motor del sensor. • Necesita pose de calibración para comenzar el rastreo. • Carece de rotaciones para la cabeza, manos, pies y clavículas.

6. Implementación de la solución

Para la implementación de las funcionalidades multitouch se optó por usar OpenNI, ya que es un Framework Open Source y al ser multiplataforma permite su utilización tanto para la plataforma Windows como para la plataforma Linux. Asimismo se optó por el lenguaje Java, ya que también es multiplataforma, pudiendo crear el proyecto tanto para Windows como para Linux.

La herramienta de desarrollo escogida inicialmente fue NetBeans, en donde se crearon prototipos funcionales utilizando el rastreo de manos y el reconocimiento de gestos provistos por el Middleware NITE. Posteriormente se cambió la herramienta de desarrollo a Processing utilizando la librería SimpleOpenNI, ya que el desarrollo con esta herramienta es mucho más simple, con lo cual se procedió a implementar las funcionalidades utilizando el rastreo esquelético en lugar de usar el rastreo de manos como se venía haciendo anteriormente.

A continuación se describirá la forma en la que trabaja OpenNI tanto con el rastreo esquelético como con el rastreo de manos. Pero partiremos hablando sobre las herramientas utilizadas.

6.1. Processing

Processing[9] es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Fue iniciado por Ben Fry y Casey Reas a partir de reflexiones en el Aesthetics and Computation Group del MIT Media Lab dirigido por John Maeda.

Es una herramienta multiplataforma, que además permite exportar las aplicaciones creadas tanto para Windows como para Linux o Mac.

6.2. SimpleOpenNI[10]

SimpleOpenNI representa la librería principal usada para la realización del presente proyecto. Es una versión simplificada de las librerías de OpenNI y NITE, la cual está adaptada para ser usada con la herramienta Processing. Esta librería implementa alguna de las funcionalidades que poseen las librerías OpenNI/NITE, pero no todas, como por ejemplo: Detección y rastreo esquelético, detección de gestos, detección y rastreo de manos, análisis de escena, entre otras funcionalidades.

[9] http://www.processing.org/
[10] http://code.google.com/p/simple-openni/

Si bien la librería SimpleOpenNI no implementa el 100% de las funcionalidades que tienen las librerías OpenNI/NITE, provee las funcionalidades necesarias para la realización del presente proyecto, además de ser más simple de utilizar con la herramienta Processing.

6.3. Usando Rastreo Esquelético

El proyecto creado en Processing está separado de 5 archivos El archivo principal llamado "*Proyecto_Final.pde*" en donde se inicializan las variables y se ejecuta el programa. Además de 4 archivos en donde se implementan las funcionalidades multitouch que realiza la aplicación. Cabe destacar que se usaron archivos separados para implementar las diferentes funcionalidades para hacer más ordenado la lectura del código, pero todas las funcionalidades podrían estar perfectamente dentro del mismo archivo, es decir, todo el código de la aplicación dentro del mismo archivo.

Ahora se analizará el código fuente mencionando las secciones importantes para entender cómo se llevó a cabo el proyecto.

6.3.1. Función principal

Lo primero es agregar la librería principal utiliazada en el proyecto, la cual es SimpleOpenNI, además de las librería Robot para hacer uso de las funcionalidades tanto del mouse como del teclado y una librería llamada ddf.minim la cual nos proveerá del uso de sonidos que se verá más adelante.

```
import SimpleOpenNI.*;
import java.awt.Robot;
import ddf.minim.*;
```

Se definen ciertas variables a utilizar, por ejemplo, las articulaciones del cuerpo humano se trabajan como vectores, un vector es una variable de 3 dimensiones que sirve para guardar las posiciones X, Y, Z de la articulación, se crean variables de tipo boolean, para comprobar, por ejemplo si una mano esta presionada o no. Algunas de las variables utilizadas se definen a continuación:

```
boolean lpressed = false, rpressed = false;
PVector handLeft; //Guardar posicion de la mano izquierda
PVector handRight; //Guardar posicion de la mano derecha
PVector cadera; //Guardar posicion de la cadera
PVector shoulder; //Guarda posición del hombro
```

En Processing en la función Setup() es donde se inicializan todas las variables utilizadas en el proyecto, además de habilitar los sensores de profundidad, RGB, habilitar el rastreo

esquelético a todas las articulaciones del cuerpo, cargar un archivo de sonido para cuando se habiliten o deshabiliten acciones (se verá más adelante), entre otras cosas.

```
kinect.enableDepth(640,480,30); //Habilita mapa de profundidad a 30fps
kinect.enableRGB(); //Habilita cámara RBG
kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL); //Habilita esqueleto
snare = minim.loadSample("Sound.wav", 512); //Carga un archivo de sonido
```

La función Draw() es la función principal del proyecto, se actualiza en cada frame y es la que se encarga de llamar a las funciones que realizan ciertas acciones, llamar a la función que dibuja el esqueleto, entre otras cosas.

```
kinect.update(); //Recibe información actualizada de los sensores
if (kinect.isTrackingSkeleton(1)) {
//Todo lo que este dentro del if se realizara solo si hay un usuario al
//que se le este realizand seguimiento esqueletico
}
```

Las siguientes 2 funciones sirven para obtener la posición de alguna articulación, en este caso serán las manos izquierda y derecha, las cuales se guardaran en sus respectivas variables (vectores):

```
//Guarda la posición de la mano izquierda en el vector leftHand
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_LEFT_HAND,leftHand);
//Guarda la posición de la mano derecha en e vector rightHand
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_RIGHT_HAND,rightHand)
```

Para el movimiento del mouse se utiliza la función robot, donde le pasamos la posición X e Y de la mano izquierda

```
robot.mouseMove((((int)hand.x-30)*displayWidth)/560,(((int)hand.y-
30)*displayHeight)/400);
```

El funcionamiento de la aplicación está basado básicamente en el cálculo de distancias, en la función Zoom que se verá más adelante se verá el uso del cálculo de distancias, ya que la función se basa en el cálculo de la distancia entre ambas manos en el espacio 3D. El cálculo de la distancia de la mano se ve reflejado en el siguiente código:

```
handDistance =
get_Distances(1,SimpleOpenNI.SKEL_LEFT_HAND,SimpleOpenNI.SKEL_RIGHT_HAND);
```

Como solo se tienen 2 manos y se deben realizar varias acciones, como por ejemplo, Click simple, Click continuo, Click doble, Zoom in, Zoom out y Scroll se optó por habilitar y deshabilitar acciones. Así se puede utilizar por ejemplo la mano derecha para realizar Click simple o Click continuo al mismo tiempo o bien usarla para realizar solo click doble (deshabilitando el click simple y continuo). La habilitación o deshabilitación de funcionalidades se hace dentro de la función ClickEnable().

La función pressed() es la encargada de verificar si una mano esta presionada contra el sensor o no. Se hace distinción entre la mano izquierda y la derecha.

La función zoom() Es la encargada de realizar las acciones de Zoom in y Zoom out basándose en la distancia de la mano.

La función SwipeDetector aún no tiene una funcionalidad definida, ya que se pensó para emular el gesto SWIPE de NITE, pero en el presente proyecto no está habilitada aun dicha opción. Aun así esta función se verá con más detalle más adelante.

6.3.2. Función Pressed

Esta función es la que se encarga de detectar si una mano esta presionada contra el sensor o no. La realización de esta acción se pensó de la siguiente manera:

Al comenzar el rastreo esquelético del usuario lo primero que se realiza es calcular la distancia del brazo. No interesa que sea la distancia exacta del brazo, sino que solo interesa un valor aproximado, por lo que la distancia del brazo se calcula midiendo la distancia entre el hombro y la cadera.

¿Por qué se pensó de esta manera y no definiendo un valor fijo? Pues las personas poseen diferentes tamaños, algunos tienen brazos más largos que otros, por lo que si definíamos un valor fijo puede darse el caso de que personas con brazos muy cortos deban estirar mucho su brazo para poder realizar la acción de "Click", caso contrario, si la persona posee los brazos muy largos y la distancia definida por defecto es muy corta puede resultar muy sensible al movimiento de la persona. Es por esto que se decidido trabajar con valores que varíen de persona a persona, pero que se mantengan fijos durante la ejecución de la aplicación.

El cálculo de la distancia entre el hombro y la cadera de calcula de manera simple, restando ambas variables en el eje Y.

```
      kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_LEFT_SHOULDER,shoulder);
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_LEFT_HIP,hip);
armLenght = shoulder.y - hip.y;
```

Para determinar si una mano esta presionada o no también se usó el cálculo de distancia simple. Para esto usamos 2 puntos de referencia:

- La posición de la mano.
- La posición del hombro.

Tenemos 2 variables booleanas, una por cada mano, que indican si dicha mano esta presionada o no. Por defecto vienen definidas como falsas

```
boolean lpressed = false, rpressed = false;
```

Para ver si la mano derecha esta presionada lo primero es obtener la posición la mano derecha y la posición del hombro derecho como se puede apreciar en el siguiente código:

```
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_RIGHT_HAND, handRight);
rhandZ = handRight.z;
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_RIGHT_SHOULDER,
shoulderRight);
rshoulderZ = shoulderRight.z;
```

Teniendo estos 2 puntos de referencia se procede a determinar si la mano derecha esta presionada, para eso se debe cumplir ciertas condiciones:

- **rpressed debe de ser falso**, es decir, la mano no debe de estar presionada.
- **armLenght > 0**, indica que la longitud del brazo es mayor a cero, es decir que hay un usuario detectado al que se le está haciendo seguimiento esquelético.
- **rshoulderZ - rhandZ > armLenght**. La distancia en el eje Z entre la mano y el hombro debe de ser igual a la distancia del brazo.

Dependiendo de qué gesto este activado será la acción que se realice, por ejemplo puede ser un Click simple o continuo (si se deja presionada la mano) o bien un Click doble. El código completo se presenta continuación:

```
if(!rpressed && armLenght > 0 && rshoulderZ - rhandZ > armLenght)
{
    rpressed = true;
    if(clickEnable){
      robot.mousePress(java.awt.event.InputEvent.BUTTON1_MASK);
    }
    else if(dobleClickEnable)
    {
      println("DOBLE CLICK");
      robot.mousePress(java.awt.event.InputEvent.BUTTON1_MASK);
      robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_MASK);
      robot.mousePress(java.awt.event.InputEvent.BUTTON1_MASK);
      robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_MASK);
    }
    println("RIGHT PRESSED");
}
```

Para determinar si la mano derecha se suelta (una vez que ha sido presionada) se utiliza una lógica similar:

- **rpressed debe de ser verdadero**, es decir, la mano debe de estar presionada.
- **rshoulderZ - rhandZ < armLenght,** la distancia entre la mano y el hombro debe de ser menor a la distancia del brazo.

El código completo se muestra a continuación:

```
else if(rpressed && rshoulderZ - rhandZ < armLenght)
{
  rpressed = false;
  if(clickEnable){
    robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_MASK);
  }
  println("RIGHT RELEASED");
}
```

Análogamente se realizan las mismas acciones para determinar si la mano izquierda esta presionada o bien si se soltó después de haber sido presionada.

6.3.3. Función clickEnable

La función clickEnable está definida dentro del archivo "*ClickEnable.pde*". En esta función es donde se habilitan o deshabilitan las acciones a realizar cuando se presionan las manos contra el sensor. Por defecto está habilitado el movimiento del mouse con la mano izquierda y el Click simple y Click continuo con la mano derecha. Se pueden habilitar más acciones, como por ejemplo el Click doble, o el Zoom, para esto también se usa el cálculo de distancias simple.

Para activar o desactivar acciones se utiliza el cálculo de distancias simple, para esto nos basaremos en 2 puntos de referencia:

- La posición de las manos (izquierda y derecha).
- La posición de la cadera.

La acción de desactivar o activar gestos se pensó de la siguiente manera basándose en la distancia entre las manos y la cadera:

- Para activar o desactivar gestos se deben mover las manos atrás de la cadera.
- Solo la mano derecha detrás de la cadera activa o desactiva el Click doble.
- Solo la mano izquierda detrás de la cadera activa o desactiva el Scroll.
- Ambas manos detrás de la cadera activa o desactiva el Zoom.

Para estas acciones se tienen 2 variables booleanas, que indican si existen ambas manos o no, el hecho de que una mano no existe quiere decir que dicha mano se encuentra detrás de la cadera.

```
boolean right_hand = true;
boolean left_hand = true;
```

Para determinar si una mano está detrás de la cadera se utiliza la posición en el eje Z. Si la posición de la mano en el eje Z es 8cms (80.0) mayor a la posición en el eje Z de la cadera, entonces la mano estará detrás de la cadera. Además se deberá cumplir que la posición de dicha mano en el eje Y debe de ser menor o igual a 0, con esta restricción se restringe a que la mano se deba mover hacia atrás solo desde la altura de la cadera hacia abajo.

Por ejemplo, para activar el Click doble con la mano derecha se deben de cumplir ciertas condiciones:

- DobleClick desactivado.
- Ambas manos deben de estar delante de las caderas.
- La mano izquierda debe de estar delante de la cadera.
- La mano derecha debe de estar detrás de la cadera.

Si se cumplen estas condiciones se habilita el Click doble con la mano derecha y se desactivan las otras acciones (Zoom, click simple, click continuo, Scroll)

El código fuente de esta acción se muestra a continuación:

```
//////// MANO DERECHA DETRAS DEL CUERPO ACTIVA CLICK DOBLE ////////////
if(   left_hand && right_hand && !dobleClickEnable &&
      armLenght > 0 && (handRight.z > (cadera.z + 80.0)) &&
      ((handLeft.z < (cadera.z + 80.0))) && handRight.y < 0)
{
  ///// Habilito el Doble Click
  dobleClickEnable = true;
  ///// Deshabilito todas las demas acciones
  right_hand = false;
  click2 = false;
  zoomEnable = false;
  clickEnable = false;
  println("CLICK DOBLE ACTIVADO");
  //// Emite un sonido avisando del cambio de gestos
  snare.trigger();
}
```

Si se vuelve a colocar la mano derecha detrás de la cadera se desactiva el Click doble y se vuelve a las acciones por defecto. El código es el siguiente:

```
//////// MANO DERECHA DETRAS DEL CUERPO DESACTIVA CLICK DOBLE ////////
if(   left_hand && right_hand && dobleClickEnable &&
      armLenght > 0 && (handRight.z > (cadera.z + 80.0)) &&
      ((handLeft.z < (cadera.z + 80.0))) && handRight.y < 0)
{
  ///// Habilita el Click
  clickEnable = true;
  ///// Deshabilita todas las otras acciones
  dobleClickEnable = false;
  right_hand = false;
  click2 = false;
  zoomEnable = false;
  println("CLICK DOBLE DESACTIVADO");
  //// Emite un sonido avisando del cambio de gestos
  snare.trigger();
}
```

Análogamente se realizan las mismas acciones para las otras acciones, ya sea si solo la mano izquierda está detrás de la cadera o bien si ambas manos están detrás de la cadera.

Como la aplicación está pensada para ser usada a cierta distancia de la Kinect y como no posee la interfaz que avise sobre los cambios que se realizan durante el uso de la aplicación es que se le incorporo sonido. Cada vez que se habiliten o deshabiliten acciones se escuchara un sonido, lo cual le indicara al usuario que la activación o desactivación de alguna acción se llevó a cabo satisfactoriamente. De momento y para fines de prueba solo se le incorporo 1 sonido, es decir que si se activa o desactiva alguna acción sonara siempre el mismo sonido.

6.3.4. Función Zoom

Esta función está dentro del archivo "*Zoom.pde*". Esta función es la que implementa las acciones de Zoom IN y Zoom OUT en la aplicación. Para esto se basa en el cálculo de distancias entre ambas manos.

Para realizar la acción Zoom primero se debe de haber habilitado la acción Zoom (colocando ambas manos detrás de la cadera como se vio en la función ClickEnable). Una vez activado el Zoom se procede a presionar ambas manos contra el sensor. Al presionar ambas manos se guarda la distancia entre ambas manos en una variable llamada "zoom" Manteniendo ambas manos presionadas se procede a juntar o a separar ambas manos. Cada vez que se detecte una diferencia en la distancia de 50 milímetros `(zoomDistance + 50.0)` se procederá a realizar el Zoom correspondiente:

- Si la distancia crece se realiza el Zoom IN.
- Si la distancia disminuye se realiza el Zoom OUT

Las acciones de Zoom se realizas mediante el uso de la función Robot de Java. Para el Zoom IN se presiona la tecla CTRL y se mueve el Scroll del mouse en el eje negativo. El código se puede ver a continuación:

```
// Si la distandia de las manos aumenta, se activa el ZOOM IN
if(zoom && (handDistance > (zoomDistance + 50.0)))
{
  println("ZOOM IN");

  robot.keyPress(KeyEvent.VK_CONTROL);
  robot.mouseWheel(-1);
  zoomDistance = handDistance;
  robot.keyRelease(KeyEvent.VK_CONTROL);
}
```

Análogamente para el Zoom OUT

```
//// Si la distancia de las manos disminuye se activa el ZOOM OUT
else if(zoom && (handDistance < (zoomDistance - 50.0)))
{
  println("ZOOM OUT");

  robot.keyPress(KeyEvent.VK_CONTROL);
  robot.mouseWheel(1);
  zoomDistance = handDistance;
  robot.keyRelease(KeyEvent.VK_CONTROL);
}
```

El cálculo de la distancia entre ambas manos para este caso se hizo mediante el teorema de Euclides[11] (Pitágoras, pero llevado a 3 dimensiones) como se muestra a continuación:

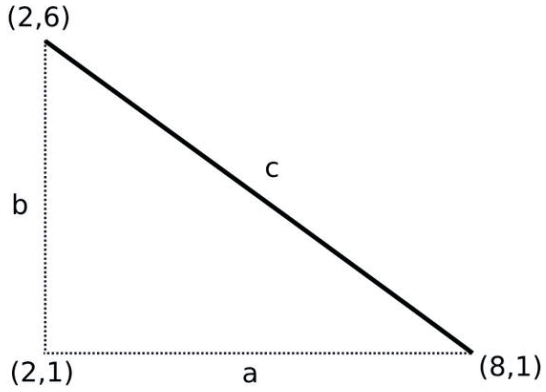


Figura 13 Cálculo de distancia según Pitágoras

La distancia C se calcula de la siguiente manera:

$c^2 = a^2 + b^2$

Se sabe que:

a = (X1 – X2)

b = (Y1 – Y2)

Luego se tiene que

$$c = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Llevado a 3 dimensiones quedaría:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

En la aplicación el cálculo de la distancia está definido en 2 funciones, una que obtiene los puntos y otra que realiza los cálculos:

```
//Funcion para obtener distancias
float get_Distances(int userId,int joint1,int joint2)
{
  //Almacenar la distancia final
  float distance;

  //Vectores que almacenaran la posicion de las manos
  PVector hand1 = new PVector();
```

[11] http://learning.codasign.com/index.php?title=Distance_in_3D_Space

```
  PVector hand2 = new PVector();
  //Obtiene la posicion 3D de las manos
  kinect.getJointPositionSkeleton(1,joint1,hand1);
  kinect.getJointPositionSkeleton(1,joint2,hand2);

  //LLama a la funcion para calcular distancias
  distance = calculate_Distances(hand1,hand2);

  return distance;
}

// Funcion para calcular la ditancia entre 2 puntos en espacion de 3
dimensiones
// la cual se basa en la norm Euclidea
float calculate_Distances(PVector point1, PVector point2)
{
  //Guardamos el largo del eje x, eje y, eje z
  float difx, dify, difz;
  //Guardamos la distancia final
  float dist_f;
  //Calculamos la distancia de cada punto
  difx = point1.x - point2.x;
  dify = point1.y - point2.y;
  difz = point1.z - point2.z;

  //Calculamos la distancia final
  dist_f = sqrt(pow(difx,2)+pow(dify,2)+pow(difz,2));

  return dist_f;
}
```

6.3.5. Función SwipeDetector

La función SwipeDetector está incluida dentro del archivo "*SwipeDetector.pde*". Esta función trata de emular el gesto SWIPE de NITE, pero utilizando el rastreo esquelético. Para ello se hace uso de un arreglo de vectores llamado swipe, en donde se van almacenando las posiciones de por las que pasa la mano.

Para detectar el gesto swipe en alguna dirección se realiza un cálculo en la distancia entre el primer punto y el último punto del arreglo.

- Si hay un cambio en el eje X positivo de una cierta distancia entonces se reconoce como un SWIPE LEFT.
- Si hay un cambio en el eje X negativo de una cierta distancia entonces se reconoce como un SWIPE RIGHT.
- Si hay un cambio en el eje Y positivo de una cierta distancia entonces se reconoce como un SWIPE DOWN.
- Si hay un cambio en el eje Y negativo de una cierta distancia entonces se reconoce como un SWIPE UP.

Esta función esta solo definida en la aplicación, pero no está activada pues aun no está funcional al 100%.

6.4. Utilizando rastreo de manos y dedos

Es un programa que permite controlar el sistema operativo Windows 7 utilizando las funcionalidades multitouch ópticas por medio del sensor Kinect creado para la consola de videojuegos Xbox 360. Este programa permite el reconocimiento y rastreo de los dedos del usuario para implementar las funcionalidades multitouch ópticas.

Para hacer uso del programa, el usuario debe:

- Estar a una distancia de 1 metro a 1,5 metros del sensor Kinect.
- Utilizar ambas manos para la ejecución de funciones.

Al comenzar a utilizar el programa, la primera mano detectada será la mano de acciones. La mano de acciones es la que puede realizar las funciones de Doble Click y Click Derecho. La segunda mano detectada corresponde a la mano de control. La mano de control permite controlar el puntero del Mouse del sistema operativo.

Importante: Este programa utiliza reconocimiento de los dedos del usuario y lleva a cabo funciones en base a estos. Por ello es importante que las funcionalidades descritas a continuacion se lleven a cabo tal y como aparecen en las imagines.

6.4.1. Funciones Disponibles

Manejo del puntero:

Para controlar el puntero del mouse ambas manos deben estar abiertas, como se muestra en la imagen, y se debe mover la mano de control en cualquier dirección. Este modo es el modo principal, por ello las funcionalidades de Doble Click, Click Derecho y Swipe solo pueden ser realizadas en esta modalidad.

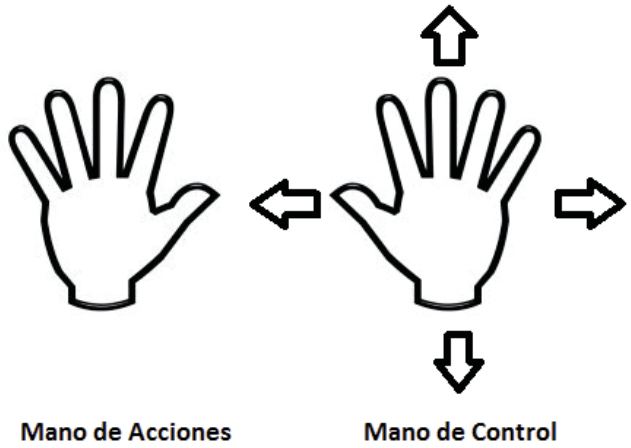


Figura 14 Manejo del puntero

Doble Click:

Para realizar la acción de doble click, se debe acercar y alejar la mano abierta, como se ilustra en la foto, a una velocidad moderada del sensor Kinect.

Figura 15 Click Doble

Click Derecho:

Para activar el click derecho del mouse, se debe acercar y alejar la mano con los dedos juntos, como se ilustra en la foto, a una velocidad moderada del sensor Kinect.

Figura 16 Click Derecho

Zoom:

Para activar el zoom, de deben mantener juntos los dedos de ambas manos y cuando se escuche el sonido de activación, proceder a alejar o acercar las manos para aumentar o disminuir el zoom. Para desactivar esta función basta con abrir los dedos de cualquiera de las manos. Al activar esta función el control del puntero del mouse se desactiva.

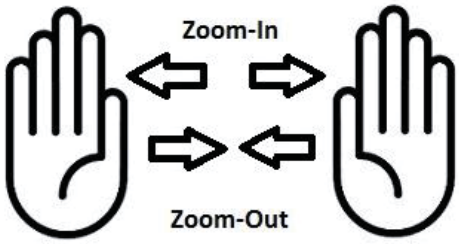


Figura 17 Zoom In Zoom Out

Scroll:

Para activar la función scroll, solo basta mantener juntos los dedos de la mano de control, y luego de que escuche el sonido de activación, proceder a mover la mano de control. Para desactivar esta función se debe abrir los dedos de la mano de control.

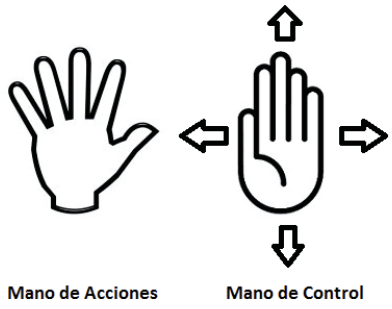


Figura 18 Scroll

Pan - Selección:

La función Pan o Selección es activada cuando los dedos de la mano de acción se mantienen juntos. Al escuchar el sonido de activación se puede mover la mano de control para seleccionar los objetos deseados. Para desactivar esta función basta con abrir los dedos de la mano de acciones.

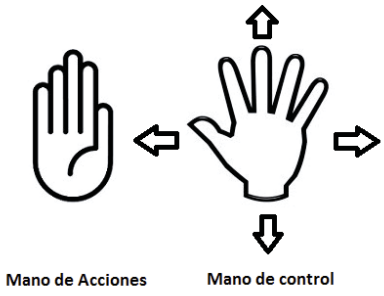


Figura 19 Pan-Seleccion

Swipe:

Esta funcionalidad se activa en el Modo de "Manejo del puntero", para ello basta con mover la mano de acciones en cualquiera de las 4 direcciones que se desee avanzar de acuerdo a la aplicación en uso.

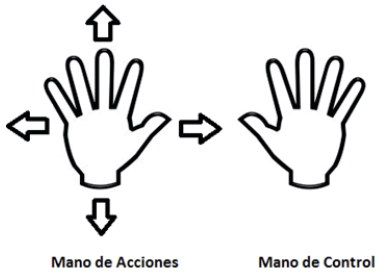


Figura 20 Swipe

6.4.2. Uso de Rastreo de Manos (HandTracking) y Detección de dedos (FingerTracking)

Para implementar este programa se utilizó el rastreo de dedos por medio de la librería *Isolines* de Processing.

La librería *Isolines* permite detectar siluetas y figuras de una imagen. En base a esta funcionalidad se pueden detectar los dedos de una mano como circunferencias de 1 a 2cms al obtener la silueta de la mano por medio de la librería *Isolines*. Con ello se puede almacenar la posición de cada dedo, en este caso el centro de cada circunferencia detectada, como una coordenada en pixeles (x,y) en un arreglo, para luego hacer uso de esta información.

Debido a que la librería *Isolines* detecta las circunferencias de toda la imagen, se debe limpiar la imagen de profundidad, eliminando los pixeles oscuros y dejando solo los pixeles blancos en la imagen, que son los que corresponden a objetos detectados por la cámara de profundidad, para luego proceder a detectar las circunferencias en los objetos de la imagen.

Para reconocer los dedos del usuario se utiliza la información provista por los puntos de control generados por la librería SimpleOpenNI para el reconocimiento de manos. En donde de cada mano detectada (Punto de control) se obtiene su posición en coordenadas (x,y) medidas en pixeles por medio del método

```
context.convertRealWorldToProjective(auxHand,screenPos);
```

y luego filtrar la posición de todas las circunferencias encontradas en la imagen que estén entre -50 y +50 pixeles en el eje x desde el punto de control y entre -60 y -25 pixeles en el eje y para contarlas como dedos de las manos.

Este proceso es llevado a cabo en el siguiente método:

```
public void drawFingers(PVector righthand, FingerTracker
fingers,XnVHandPointContext hand){
```

```
    float distance;
    PVector position,prevpos;
    fingers.setThreshold((int)hand.getPtPosition().getZ()+70);
    for(int i = 0; i < fingers.getNumFingers(); i++){
    position = fingers.getFinger(i);
    if (position.x<(righthand.x)+50 &
        &position.x>(righthand.x)-50&&position.y<(righthand.y)+20
      &&position.y>(righthand.y)-50){
    stroke(255,0,0);
    strokeWeight(5);
    point(position.x, position.y);
    if(hand==right){rightfinqty++;}
    else{leftfinqty++;}
    finqty++;}
    }
  }
}
```

En donde la variable righthand es la posición en pixeles de la mano detectada y la variable position es utilizada para almacenar la posición de cada circunferencia detectada en la sentencia:

```
for(int i = 0; i < fingers.getNumFingers(); i++)
```

Y poder filtrarla en la sentencia

```
if (position.x<(righthand.x)+50 && position.x>(righthand.x)-50 &&
    position.y<(righthand.y)+20 && position.y>(righthand.y)-50)
```

Para contarlas como dedos de las manos en las variables *rightfinqty* (Cantidad de dedos de la mano derecha), *leftfinqty* (Cantidad de dedos de la mano izquierda) y *finqty* (cantidad de dedos total, la cual no debe ser mayor que 10).

La cantidad de dedos detectada es utilizada en los métodos:

```
public void leftgrab()// Function Pan y Seleccion
public void rightgrab()// Function Scroll
public void zoom()//    Función Zoom
```

Para activar las funciones asociadas en cada método. En cada uno de estos métodos hay un tiempo determinado que se debe cumplir para activar la función correspondiente almacenado en las variables ltime (para el método leftgrab), rtime (para el método rightgrab), y zoomtime (para el método zoom). El tiempo de activación es utilizado para evitar la ejecución de acciones involuntarias por parte del usuario.

Las variables de tipo boolean zoomstate (función zoom), lgrabstate (función leftgrab), y rgrabstate (funcion rightgrab) permiten controlar el flujo de la aplicación y actúan como flag señalando la función que está activa en un momento determinado. Dentro del flujo de la aplicación solo está permitido que solo una de estas variables sea verdadera, es decir, que solo una función pueda ser activada en un momento determinado.

Por su parte, el método zoom utiliza dos métodos auxiliares. El primero de ellos es: gethandsDistance() y por último los métodos isgrowing() y isdecreasing(). El método

gethandsDistance()permite obtener la distancia en 2D (No considera el eje Z) la cual es calculada por el método distance2D(), para luego entregar la distancia como parámetro a los métodos isgrowing() y isdecreasing() los cuales calculan y retornan un valor booleano si la distancia aumenta o disminuye respectivamente.

El método movemouse() recibe como parámetro un vector mouse con la posición de la mano de control en coordenadas reales (RealWorld coordinates) y un vector screenPos que almacena la posición actual del mouse en coordenadas (x,y) medidas en pixeles. Este método utiliza el método smoother() que recibe como parámetro el vector mouse y multiplica cada coordenada por un escalar con la finalidad de maximizar el movimiento del mouse, para así evitar movimientos extensos por parte del usuario para alcanzar los límites de la pantalla. Luego de procesar el vector mouse por el método smoother, el vector mouse es entregado como parámetro en el método

```
context.convertRealWorldToProjective(mouse,screenPos)
```

Para convertirlo a coordenadas (x,y) medidas en pixeles. El resultado de esta conversión es almacenada en el vector screenPos el cual es utilizado para posicionar el mouse en las coordenadas correspondientes en la pantalla del usuario por medio de la librería Robot con el método

```
robot.mouseMove((((int)screenPos.x-30)*width)/560,(((int)screenPos.y-
30)*height)/400).
```

Por ultimo está el método draw() principal de la clase KinectTracking a la cual pertenecen los métodos mencionados anteriormente. El método draw() es en donde se llama a los métodos leftgrab(), rightgrab(), movemouse(), y zoom().

7. Comparación de las 2 soluciones propuestas

Para el presente proyecto se desarrollaron 2 soluciones utilizando los 2 paradigmas de OpenNI: Rastreo de manos y Rastreo esquelético. Ambas soluciones cumplen con las especificaciones planteadas inicialmente y permiten manipular el computador mediante gestos.

Si se comparan ambas aplicaciones en cuanto a su funcionamiento y su aplicabilidad se pueden ver algunas diferencias como por ejemplo:

- La aplicación basada en rastreo esquelético requiere que la persona se encuentre de pie frente al sensor Kinect y a una distancia de a lo menos 1.5mts, ya que se necesita registrar prácticamente el cuerpo completo del usuario. Por lo cual la aplicación debe de ser usada en un lugar con un amplio espacio.
- La aplicación basada en el rastreo de manos no requiere tanto espacio, el usuario debe de estar a lo menos 1 metro de distancia del sensor Kinect. Al no necesitar el rastreo esquelético, ya que solo utiliza las manos, el usuario puede utilizar la aplicación incluso estando sentado en una silla, lo que se podría ser aplicado por ejemplo para que personas discapacitadas que se encuentren en una silla de ruedas puedan realizar presentaciones o manipular objetos en la pantalla a distancia.

En cuanto a la usabilidad de ambas aplicaciones luego de ser usada por algunos usuarios, se pudo notar lo siguiente:

- La aplicación basada en el rastreo esquelético resulto más fácil de usar ya que los gestos que se deben realizar resultan más naturales para los usuarios. No son gestos tan complejos y tratan de imitar los gestos que se utilizan en superficies táctiles, como por ejemplo el touchpad del notebook. Lo que si puede resultar un tanto complicado en un comienzo es el hecho de que se deban activar y desactivar acciones, ya que como el usuario solo posee 2 manos no es posible realizar todas las acciones disponibles, por lo cual se deben activar o desactivar acciones mediante un gesto.
- La aplicación basada en rastreo de manos resulto un poco más difícil de usar, ya que además de utilizar las manos para realizar gestos también se utilizan los dedos, algo que no resulta natural para el usuario. La combinación de gestos que se deben de hacer para realizar las acciones también es algo complejo, lo que genera un problema para el usuario el memorizar las combinaciones para realizar los gestos.

En cuanto a la forma de desarrollo de ambas aplicaciones se pudo determinar lo siguiente:

- En la aplicación basada en el rastreo esquelético, la forma de trabajo es más sencilla, ya que solo necesita que un usuario sea detectado frente al sensor Kinect y que la calibración se haya realizado correctamente. Luego de que se haya generado el

esqueleto del usuario se pueden acceder directamente a cualquiera de los puntos de generados en el esqueleto (principalmente articulaciones). Con esto yo puedo saber la posición en el espacio de cualquiera de las articulaciones utilizando si identificador. En caso de trabajar con más de un usuario, cada usuario posee un ID que lo identifica, cada usuario posee su propio esqueleto y si un usuario sale del campo de visión dicho usuario (ID) junto con su esqueleto son destruidos.

- En la aplicación basada en rastreo de manos la forma de trabajo es un poco más complicada, ya que se necesita manejo de sesión. Se deben definir gestos de enfoque, los cuales serán usados para iniciar y mantener una sesión. Ya no se trabaja con un ID de usuario, sino que cada mano detectada posee su propio ID. El manejo de gestos es más complejo y son funciones que ya vienen definidas. El manejo de detección de dedos también es algo complicado ya que Kinect no está hecha para el reconocimiento de dedos, pero se pudo hacer usando una librería nueva llamada "Isolines" la cual no posee mucha documentación aun y algunas veces genera ciertos errores en la detección de dedos detectando falsos positivos (detecta dedos donde no los hay).

Tabla 4 Tabla Resumen

Funcionamiento y aplicabilidad	
Skeleton Tracking	***Hand Tracking***
• Requiere que el usuario se encuentre de pie frente al sensor Kinect. • Distancia mínima entre el sensor y el usuario es de 1,5mts. • Requiere de un lugar con un amplio espacio. • Puede ser usado para realizar presentaciones en oficinas o salones que posean un espacio libre tal que se pueda detectar al usuario de cuerpo completo.	• No es necesario el rastreo esquelético, solo usa las manos. • Distancia mínima entre el sensor y el usuario es de 1 metro. • Puede ser usado incluso sentado en una silla. • Puede ser usados por personas discapacitadas que se encuentren sentados en una silla de ruedas ya que solo se necesitan las manos.
Usabilidad	
Skeleton Tracking	***Hand Tracking***
• Fácil de usar. • Utiliza gestos relativamente naturales por parte del usuario. • Los gestos se implementaron para que funcionaran de manera análoga a un touchpad de notebook, por lo que no sería algo nuevo para el usuario. • La única complejidad es el hecho de tener que activar y desactivar acciones.	• Un poco compleja de usar. • Sus gestos no resultan tan naturales para el usuario. • El usar reconocimiento de dedos podría ser algo complicado o incómodo para el usuario. • La combinación de gestos para realizar acciones es más compleja. • La aplicación algunas veces detecta dedos que no existen (falsos positivos).

Desarrollo	
Skeleton Tracking	***Hand Tracking***
• La forma de trabajo es más sencilla. • El usuario solo necesita posicionarse frente al sensor Kinect y realizar una pose de calibración para que poder utilizar la aplicación. • Una vez iniciada la aplicación se pueden acceder a cualquiera de los puntos del esqueleto de forma directa, permitiendo saber su posición en el espacio. • Cada usuario posee un ID único, y cada esqueleto detectado está asociado a un usuario (ID), si el usuario sale del campo de visión se destruye si ID y la información de su esqueleto.	• La forma de trabajo es algo más compleja. • Necesita manejo de sesiones para comenzar a utilizar la aplicación. • De deben definir gestor para iniciar y mantener una sesión activa. • No se trabaja con ID de usuario, sino que cada mano detectada posee su propia ID. • Se puede definir un tiempo máximo en que la mano puede permanecer fuera del campo de visión antes de ser destruida. • Kinect no está hecho para reconocimiento de dedos, pero se puso implementar usando una librería nueva llamada "*Isolines*". • La aplicación a veces puede detectar dedos que no existen, por lo que de sebe de restringir el campo de visión máximo para así eliminar toda imagen que se encuentre detrás de la mano (fondo de la imagen).

En resumen, luego de analizar ambas aplicaciones se pudo notar que en cuanto a funcionamiento, usabilidad y desarrollo la aplicación basada en el rastreo esquelético es mucho más simple y resulta más natural para el usuario al momento de utilizarla ya que posee gestos que resultan más naturales para el usuario, el único inconveniente es que requiere de un amplio espacio para su uso y en algunos casos puede generar problemas en cuanto a la calibración del esqueleto del usuario. En cuanto a la parte de desarrollo de la aplicación también posee cierta ventaja sobre la aplicación de rastreo de manos en cuanto a su simplicidad y facilidad de entendimiento, lo que hace que el desarrollo de aplicaciones sea más rápido.

Si bien la aplicación de rastreo de manos resulta más compleja, tanto en su desarrollo como para el uso de los usuarios también posee características importantes como es el reconocimiento de dedos, además como solo requiere hacer uso de las manos del usuario podría ser usado por usuarios que posean alguna discapacidad que les impida ponerse de pie. Haciendo uso del rastreo de dedos podría incluso aplicarse en una aplicación que sirva para interpretar el lenguaje de señas utilizando los dedos.

8. Conclusión

Durante el desarrollo del proyecto se analizaron las diferentes herramientas disponibles para trabajar con Kinect y se optó por la mejor opción disponible para la implementación de las funcionalidades. Se optó por OpenNI ya que satisfacía los objetivos del proyecto, además dee permitir la implementación para diferentes plataformas como Linux, Windows y Mac.

Luego de un frustrante intento por implementar las funcionalidades multitouch utilizando el reconocimiento de manos y gestos de NITE y programando en NetBeans se optó por modificar la forma de trabajo y se optó por realizar la implementación usando el rastreo esquelético y la herramienta Processing. Con lo cual se hizo mucho más fácil la realización del proyecto, ya que la forma de trabajo usando rastreo esquelético es mucho más simple que el rastreo de manos. Se lograron implementar la totalidad de las funcionalidades propuestas inicialmente usando solo las 2 manos.

También se logró implementar las funcionalidades utilizando el rastreo de manos, para esto se utilizó una librería que permite el reconocimiento de dedos, con lo que finalmente se presentan 2 soluciones, utilizando los 2 paradigmas utilizados por OpenNI/NITE como son el rastreo esquelético y el rastreo de manos.

Se espera que en el futuro se masifique el uso de la tecnología Kinect para interactuar con los computadores o dispositivos móviles, no solo en el ámbito de los videojuegos, sino que en aplicaciones multimedia o el mismo sistema operativo, ya que como quedó demostrado con el presente proyecto, es posible controlar ciertas funcionalidades del PC utilizando el dispositivo Kinect.

A la fecha de finalización del proyecto, la versión más reciente del driver de OpenNI usado era la versión 1.5.4. Con la cual se desarrolló el proyecto. A finales de diciembre de 2012 salió la versión 2.0 de OpenNI, la cual trae cambios en la estructura, simplificación en el uso de las clases, se eliminaron los algoritmos de OpenNI ya que ahora forman parte de las librerías middleware (como NITE), además ahora se utiliza el SDK de Microsoft como base. Debido a esto último se perdió el soporte para Linux y Mac, ya que OpenNI 2.0 utiliza el driver de Microsoft, el cual actualmente está disponible sólo para Windows.

Para trabajos futuros se podría hacer uso de otras funcionalidades que provee la Kinect, que para este proyecto no fueron utilizadas, como es el caso del reconocimiento de voz, se podría reemplazar las funcionalidades multitouch por el sistema de reconocimiento de voz, permitiendo así abrir, cerrar o controlar aplicaciones no con gestos, sino que mediante sonidos.

Otra funcionalidad interesante es el uso de la librería *Isolines* la cual se usó para el reconocimiento de dedos, con lo cual se podrían desarrollar aplicaciones que reconozcan por ejemplo el lenguaje de señas. El SDK de Microsoft posee una aplicación que permite el reconocimiento facial de un usuario, además de permitir el reconocimiento de voz en diferentes idiomas.

9. Referencias

[1] Electronic Entertainment Expo 2009
http://es.wikipedia.org/wiki/Electronic_Entertainment_Expo_2009

[2] Kinect *Sales Surpass Ten Million –xbox.com*
http://www.xbox.com/en-us/press/archive/2011/0308-ten-million-kinects

[3] PrimeSense Natural Interaction
http://www.primesense.com/

[4] OpenNI
http://openni.org/

[5] OpenNI Programmer Guide
http://openni.org/Documentation/ProgrammerGuide.html

[6] PrimeSense
http://www.primesense.com/

[7] Prime Sensor™ NITE 1.3 Framework Programmer's Guide
http://pr.cs.cornell.edu/humanactivities/data/NITE.pdf

[8] NITE Controls 1.3 – Programmer's Guide
http://andrebaltazar.files.wordpress.com/2011/02/nite-controls-1-3-programmers-guide.pdf

[9] Processing
http://www.processing.org/

[10] SimpleOpenNI
http://code.google.com/p/simple-openni/

[11] Cálculo de distancias en espacios 3D
http://learning.codasign.com/index.php?title=Distance_in_3D_Space

Anexo

1. Guía de instalación

A.1 Instalación de OpenNI en Ubuntu 12.04

Requisitos

```
$ sudo apt-get install build-essential python libusb-1.0-0-dev freeglut3-
dev openjdk-7-jdk doxygen graphviz
```

Si se desea instalar el soporte para .NET es necesario contar previamente con el soporte para Mono instalado.

```
$ sudo apt-get install mono-complete
```

Instalación OpenNI

Para instalar OpenNi lo primero es crear una nueva carpeta en el directorio personal del usuario, para luego dirigirse a la carpeta creada.

```
$ mkdir ~/kinect ; cd ~/kinect
```

Se procede a descargar la última versión de OpenNi dentro de la carpeta creada.

```
$ git clone https://github.com/OpenNI/OpenNI.git -b unstable
```

* Esta ubicación puede variar dependiendo de la version para 32 o 64 bits.

```
$ cd OpenNI/Platform/Linux/CreateRedist

$ bash RedistMaker

$ cd ../Redist/OpenNI-Bin-Dev-Linux-x64-v1.5.4.0*/

$ sudo ./install.sh
```

Instalación Driver Avin2/SensorKinect

Una vez instalado el framework de OpenNi, se procede a la instalación de Sensor Kinect.

Acceder a la carpeta y clonar la última versión de Avin2/SensorKinect.:

```
$ cd ~/kinect

$ git clone https://github.com/avin2/SensorKinect
```

Por último se procede a compilar el código fuente:

```
$ cd SensorKinect/Platform/Linux/CreateRedist
$ bash RedistMaker
$ cd ../Redist/Sensor-Bin-Linux-x64-v5.1.2.1*
$ sudo sh install.sh
```

Instalación middleware NITE

Descargar la distribución binaria para GNU/Linux más reciente desde la siguiente ubicación.

http://www.openni.org/Downloads/OpenNIModules.aspx

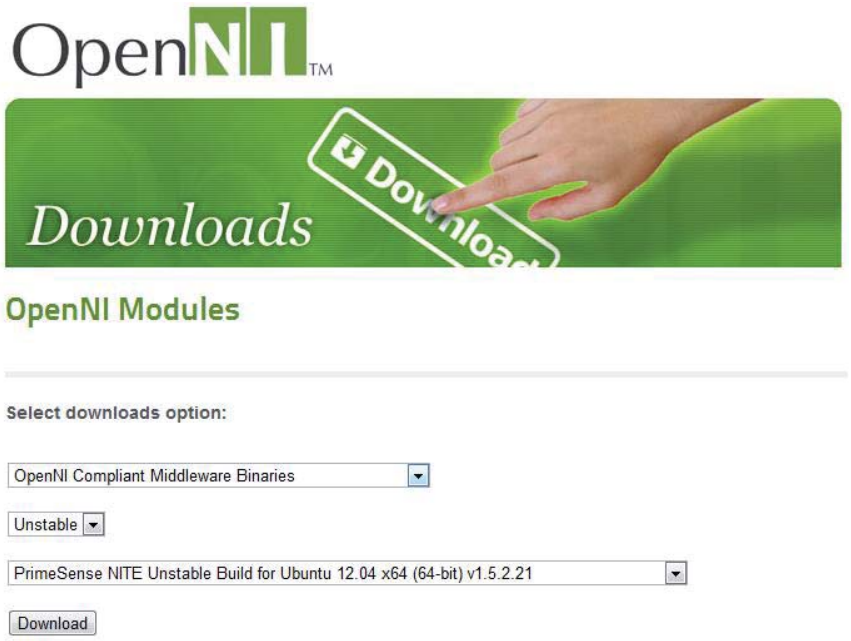


Figura 21 Instalacion OpenNi

Y ejecutar:

```
$ cd ~/kinect
$ tar jvxf nite-bin-linux-x64-v1.5.2.21.tar.bz2
$ cd NITE-Bin-Dev-Linux-x64-v1.5.2.21/Data
$ chmod a+w *
$ vi *.xml
```

Para ingresar la licencia:

```
<License vendor="PrimeSense" key="insert key here"/>
```

… reemplazar por…

```
<License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>
```

```
$ cd ..
$ cd ./install.sh
```

Con esto queda instalado OpenNI en el sistema. Para probar los ejemplos provistos por OpenNI solo basta con ingresar a algunas de las siguientes carpetas:

```
cd ~/kinect/OpenNI/Platform/Linux/Bin/x64-Release/
cd ~/kinect/NITE-Bin-Dev-Linux-x64-v1.5.2.21/Samples/Bin/x64-Debug/
```

Dentro de estas carpetas se podrán encontrar ejemplos ya compilador aplicaciones que hacen uso de las cámaras y sensores de kinect. Para ejecutar estas aplicaciones solo basta con ejecutar los script en modo consola, por ejemplo dentro de la carpeta:

```
cd ~/kinect/NITE-Bin-Dev-Linux-x64-v1.5.2.21/Samples/Bin/x64-Debug/
```

Corremos el ejemplo Sample-Players de la siguiente manera:

```
./Sample-Players
```

A.2 Instalación OpenNI/NITE en Windows 7 x64

Descargar el archivo https://dl.dropbox.com/u/52421086/OpenNI_NITE_Installer-win64-0.27.zip, descomprimirlo e instalar los componentes en el siguiente orden:

- OpenNI
- NITE
- PrimeSense Sensor
- PrimeSense SensorKinect

Para verificar el correcto funcionamiento de los componentes, conectar el sensor Kinect y ejecutar los ejemplos disponibles en la carpeta OpenNI o PrimeSense dentro del menú Inicio>Programas.

Figura 22 Ejecución del ejemplo NiViewer64

2. Código Fuente

B.1 Implementación utilizando rastreo esquelético

Proyecto_Final.pde

```
import SimpleOpenNI.*;
import java.awt.Robot;
import java.awt.AWTException;
import ddf.minim.*; // Libreria para hacer uso de audio


//Definiendo variables
SimpleOpenNI  kinect;
Robot robot;
boolean lpressed = false, rpressed = false;

float rhandZ, rhandY, rshoulderZ, rshoulderY;
float lhandZ, lhandY, lshoulderZ, lshoulderY;
float armLenght = 0;
PVector shoulder, hip;
float handDistance;

///// Variables de la funcion ClickEnable()
PVector handLeft;
PVector handRight;
PVector cadera;
boolean right_hand = true;
boolean left_hand = true;
boolean clickEnable= true;
boolean zoomEnable = false;
boolean dobleClickEnable = false;
boolean click2 = false;

PVector[] swipe;
int max_size = 10;
int frame = 0;
PVector hand;

////////////Variables de Zoom////////////////////
PVector manoIzq = new PVector();
PVector manoDer = new PVector();
float zoomDistance;
boolean zoom = false;

Minim minim;
AudioSample snare;
```

```
///////////////// Inicializando variables //////////////////////
void setup() {

  minim = new Minim(this);
  snare = minim.loadSample("Sound.wav", 512);
  if ( snare == null ) println("Error al cargar el archivo");
  kinect = new SimpleOpenNI(this);
  kinect.setMirror(true);
  kinect.enableDepth(640,480,30);
  kinect.enableRGB();
  shoulder = new PVector();  // Hombro
  hip = new PVector();       // Cadera
  hand = new PVector();      // Mano
  handRight = new PVector(); // Mano Derecha
  handLeft = new PVector();  // Mano Izquierda
  cadera = new PVector();

/// Arreglo de vectores para reconocer el gesto Swipe
  swipe = new PVector[max_size];
  for ( int i = 0; i < max_size; ++i)
  {
    swipe[i] = new PVector();
  }

  //Habilitando rastreo esqueletico con todos los puntos
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);

  PVector shoulder = new PVector();
  PVector hip = new PVector();

  try{
    robot   = new Robot();
  }
  catch (AWTException e) {
    e.printStackTrace();
  }
  smooth();
  size(640,480);
  stroke(255,0,0);
  strokeWeight(3);
}


void draw() {
  kinect.update();
  image(kinect.depthImage(), 0, 0);
  //image(kinect.rgbImage(), 0, 0);
  IntVector userList = new IntVector();
  kinect.getUsers(userList);

  drawSkeleton(1);
  //En caso que haya un usuario calibrado
  if (kinect.isTrackingSkeleton(1)) {
    //Calcula longitud del brazo (aproximado)
    if(armLenght == 0)
    {
```

```
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_LEFT_SHOULDER,shoulder);
      kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_LEFT_HIP,hip);
      armLenght = shoulder.y - hip.y;
    }

    //Vector para la mano Izquierda
    PVector leftHand = new PVector();
    //Vector para la mano Derecha
    PVector rightHand = new PVector();

    //Obtiene los puntos de la mano Izquierda y los guarda en leftHand
    kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_HAND,
leftHand);
    //Obtiene los puntos de la mano Derecha y los guarda en rightHand
    kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_HAND,
rightHand);

    //Guardo la posicion X e Y en el vector Swipe
    swipe[frame % max_size].x = rightHand.x;
    swipe[frame % max_size].y = rightHand.y;

    //Convierte las coordenadas de la mano a coordenadas 3D y
    //Mueve el mouse segun la posicion de la mano derecha
    kinect.convertRealWorldToProjective(leftHand, hand);

    //Habilita y deshabilita el uso de clicks
    ClickEnable();

    //Mueve el mouse
    robot.mouseMove((((int)hand.x-30)*displayWidth)/560,(((int)hand.y-
30)*displayHeight)/400);

    //Calcula la distancia entre las 2 manos y dibuja una linea
    handDistance =  get_Distances(1,
SimpleOpenNI.SKEL_LEFT_HAND,SimpleOpenNI.SKEL_RIGHT_HAND);
    //kinect.drawLimb(1, SimpleOpenNI.SKEL_LEFT_HAND,
SimpleOpenNI.SKEL_RIGHT_HAND);
  }

  //Funcion para ver si esta presionado
  pressed();
  //Funcion para realizar Zoom
  zoom();
  //Funcion para detectar Swipe en las 4 direcciones
  //SwipeDetector();

}


//Funcion para obtener distancias
float get_Distances(int userId,int joint1,int joint2)
{
  //Almacenar la distancia final
  float distance;

  //Vectores que almacenaran la posicion de las manos
```

```
  PVector hand1 = new PVector();
  PVector hand2 = new PVector();

  //Obtiene la posicion 3D de las manos
  kinect.getJointPositionSkeleton(1,joint1,hand1);
  kinect.getJointPositionSkeleton(1,joint2,hand2);

  //LLama a la funcion para calcular distancias
  distance = calculate_Distances(hand1,hand2);

  return distance;
}

// Funcion para calcular la ditancia entre 2 puntos en espacion de 3
dimensiones
// la cual se basa en la norm Euclidea
float calculate_Distances(PVector point1, PVector point2)
{
  //Guardamos el largo del eje x, eje y, eje z
  float difx, dify, difz;
  //Guardamos la distancia final
  float dist_f;

  //Calculamos la distancia de cada punto
  difx = point1.x - point2.x;
  dify = point1.y - point2.y;
  difz = point1.z - point2.z;

  //Calculamos la distancia final
  dist_f = sqrt(pow(difx,2)+pow(dify,2)+pow(difz,2));

  return dist_f;
}

// USERS TRACKING CALLBACKS!!!!!!
void onNewUser(int userId) {
  println("Nuevo usuario Detectado");
  kinect.startPoseDetection("Psi", userId);
}

void onStartPose(String pose, int userId) {
  println("Comenzando calibracion......");
  kinect.stopPoseDetection(userId);
  kinect.requestCalibrationSkeleton(userId, true);
}

void onEndCalibration(int userId, boolean successful) {
  if (successful) {
    println("  Usuario Calibrado!!");
    kinect.startTrackingSkeleton(userId);
  }
  else {
    println("  Error Calibrando Usuario");
    kinect.startPoseDetection("Psi", userId);
  }
}
```

```
void onLostUser(int userId)
{
  println("  Usuario Perdido!!!!");
  armLenght = 0;
}

////////////////// DIBUJA EL ESQUELETO ////////////////////////
void drawSkeleton(int userId)
{
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD, SimpleOpenNI.SKEL_NECK);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_LEFT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_LEFT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW,
SimpleOpenNI.SKEL_LEFT_HAND);

  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_RIGHT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
SimpleOpenNI.SKEL_RIGHT_HAND);

  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);

  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_LEFT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP,
SimpleOpenNI.SKEL_LEFT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE,
SimpleOpenNI.SKEL_LEFT_FOOT);

  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_RIGHT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,
SimpleOpenNI.SKEL_RIGHT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE,
SimpleOpenNI.SKEL_RIGHT_FOOT);
}
```

clickEnable.pde

```
void ClickEnable()
{
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_RIGHT_HAND,handRight);
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_RIGHT_HIP,cadera);
kinect.getJointPositionSkeleton(1,SimpleOpenNI.SKEL_LEFT_HAND,handLeft);

//////// 2 MANOS ATRAS DE LA ESPALDA DESACTIVA CLICKS ACTIVA ZOOOM /////////
if(   right_hand && left_hand && !clickEnable && armLenght > 0 &&
      (handRight.z > (cadera.z + 80.0)) &&
      (handLeft.z > (cadera.z + 80.0)) && handLeft.y < 0 &&
       handRight.y < 0)
{
  //// Habilito el click
  clickEnable = true;
  //// Deshabilito todas las otras acciones
  right_hand = false;
  left_hand = false;
  zoomEnable = false;
  click2 = false;
  dobleClickEnable = false;
  println("ZOOM DISABLE");
  //// Emite un sonido avisando del cambio de gestos
  snare.trigger();
}

/////// 2 MANOS ATRAS DE LA ESPALDA ACTIVA CLICKS DESACTIVA ZOOM /////////
if(   right_hand && left_hand && clickEnable && armLenght > 0 &&
      (handRight.z > (cadera.z + 80.0)) &&
      (handLeft.z > (cadera.z + 80.0)) && handLeft.y < 0 && handRight.y < 0)
  {
    ///// Habilito el Zoom
    zoomEnable = true;
    ///// Deshabilito todas las otras acciones
    clickEnable = false;
    right_hand = false;
    left_hand = false;
    click2 = false;
    dobleClickEnable = false;
    println("ZOOM ENABLE");
    //// Emite un sonido avisando del cambio de gestos
    snare.trigger();
  }

///////////////// MANO DERECHA DELANTE DEL CUERPO ///////////////////
 if(!right_hand && (handRight.z < cadera.z))
 {
   right_hand = true;
 }

//////////////// MANO IZQUIERDA DELANTE DEL CUERPO ///////////////////
 if(!left_hand && (handLeft.z < cadera.z))
 {
   left_hand = true;
 }
```

```
////////////////// MANO IZQUIERDA DETRAS DEL CUERPO ACTIVA CLICK2
//////////////////////
if(   left_hand && right_hand && !click2 && armLenght > 0 &&
      (handLeft.z > (cadera.z + 80.0)) &&
      (handRight.z < (cadera.z + 80.0)) && handLeft.y < 0)
  {
    ///// Habilito click secundario (Boton central del mouse)
    click2 = true;
    ///// Deshabilito todas las otras acciones
    clickEnable = false;
    dobleClickEnable = false;
    zoomEnable = false;
    left_hand = false;
    println("CLICK2 ENABLE");
    //// Emite un sonido avisando del cambio de gestos
    snare.trigger();
  }

////////////////// MANO IZQUIERDA DETRAS DEL CUERPO DESACTIVA CLICK2
//////////////////////
if(   left_hand && right_hand && click2 && armLenght > 0 &&
      (handLeft.z > (cadera.z + 80.0)) &&
      (handRight.z < (cadera.z + 80.0)) && handLeft.y < 0)
  {
    ///// Habilito el Click
    clickEnable = true;
    ///// Deshabilito todas las otras acciones
    left_hand = false;
    click2 = false;
    zoomEnable = false;
    dobleClickEnable = false;
    println("CLICK2 DISABLE");
    //// Emite un sonido avisando del cambio de gestos
    snare.trigger();
  }

////////////////// MANO DERECHA DETRAS DEL CUERPO ACTIVA CLICK DOBLE
//////////////////////
if(   left_hand && right_hand && !dobleClickEnable &&
      armLenght > 0 && (handRight.z > (cadera.z + 80.0)) &&
      ((handLeft.z < (cadera.z + 80.0))) && handRight.y < 0)
  {
    ///// Habilito el Doble Click
    dobleClickEnable = true;
    ///// Deshabilito todas las demas acciones
    right_hand = false;
    click2 = false;
    zoomEnable = false;
    clickEnable = false;
    println("CLICK DOBLE ACTIVADO");
    //// Emite un sonido avisando del cambio de gestos
    snare.trigger();
  }
```

```
////////////////// MANO DERECHA DETRAS DEL CUERPO DESACTIVA CLICK DOBLE
//////////////////
if(   left_hand && right_hand && dobleClickEnable &&
      armLenght > 0 && (handRight.z > (cadera.z + 80.0)) &&
      ((handLeft.z < (cadera.z + 80.0))) && handRight.y < 0)
  {
    ///// Habilita el Click
    clickEnable = true;
    ///// Deshabilita todas las otras acciones
    dobleClickEnable = false;
    right_hand = false;
    click2 = false;
    zoomEnable = false;
    println("CLICK DOBLE DESACTIVADO");
    //// Emite un sonido avisando del cambio de gestos
    snare.trigger();
  }
}
```

Zoom.pde

```
//////////////////// ZOOM ////////////////////////////////
void zoom()
{
//ACTIVA EL ZOOM SI LAS 2 MANOS ESTAN PRESIONADAS CONTRA EL SENSO//////
  if(!clickEnable && lpressed && rpressed && !zoom)
  {
    //// Habilito el zoom
    zoom = true;
    println("ZOOM ACTIVADO");
    //// Calculo la distancia de las manos al momento de hacer el zoom
    zoomDistance = get_Distances(1,
SimpleOpenNI.SKEL_LEFT_HAND,SimpleOpenNI.SKEL_RIGHT_HAND);
  }
  ///// Si alguna de las 2 manos se aleja del sensor se deshabilita el zoom
  else if (zoom && (!lpressed || !rpressed))
  {
    zoom = false;
  println("ZOOM DESACTIVADO");
  }

////////////////////////////// ZOOM IN ///////////////////////////////////
  // Si la distandia de las manos aumenta, se activa el ZOOM IN
  if(zoom && (handDistance > (zoomDistance + 50.0)))
  {
    println("ZOOM IN");

    robot.keyPress(KeyEvent.VK_CONTROL);
    robot.mouseWheel(-1);
    zoomDistance = handDistance;
    robot.keyRelease(KeyEvent.VK_CONTROL);
  }
```

```
//////////////////////////// ZOOM OUT ////////////////////////////////
  //// Si la distancia de las manos disminuye se activa el ZOOM OUT
  else if(zoom && (handDistance < (zoomDistance - 50.0)))
  {
    println("ZOOM OUT");

    robot.keyPress(KeyEvent.VK_CONTROL);
    robot.mouseWheel(1);
    zoomDistance = handDistance;
    robot.keyRelease(KeyEvent.VK_CONTROL);
  }
}
```

Pressed.pde

```
void pressed()
{
  ///// Inicializo los vectores para almacenar los datos
  PVector handRight = new PVector();
  PVector shoulderRight = new PVector();
  PVector handLeft = new PVector();
  PVector shoulderLeft = new PVector();

  kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_HAND,
handRight);
  rhandZ = handRight.z;
  rhandY = handRight.y;
  kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
shoulderRight);
  rshoulderZ = shoulderRight.z;
  rshoulderY = shoulderRight.y;

  kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_HAND, handLeft);
  lhandZ = handLeft.z;
  lhandY = handLeft.y;
  kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_SHOULDER,
shoulderLeft);
  lshoulderZ = shoulderLeft.z;
  lshoulderY = shoulderLeft.y;

////////////////// PRESIONO MANO IZQUIERDA ////////////////////////
  if(!lpressed && armLenght > 0 && lshoulderZ - lhandZ > armLenght)
  {
    // Mano izquierda presionada
    lpressed = true;
    if(click2)
    {
      println("CLICK SECUNDARIO");
      robot.mousePress(java.awt.event.InputEvent.BUTTON2_MASK);
    }
    println("LEFT PRESSED");
  }
```

```
/////////////// SUELTO MANO IZQUIERDA ////////////////////////
  else if(lpressed && lshoulderZ - lhandZ < armLenght)
  {
    lpressed = false;
    if(click2)
    {
      robot.mouseRelease(java.awt.event.InputEvent.BUTTON2_MASK);
      robot.mousePress(java.awt.event.InputEvent.BUTTON1_MASK);
      robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_MASK);
    }
    println("LEFT RELEASED");
  }



////////////////// PRESIONO MANO DERECHA ////////////////////////
  if(!rpressed && armLenght > 0 && rshoulderZ - rhandZ > armLenght)
  {
    rpressed = true;
    if(clickEnable){
      robot.mousePress(java.awt.event.InputEvent.BUTTON1_MASK);
    }
    else if(dobleClickEnable)
    {
      println("DOBLE CLICK");
      robot.mousePress(java.awt.event.InputEvent.BUTTON1_MASK);
      robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_MASK);
      robot.mousePress(java.awt.event.InputEvent.BUTTON1_MASK);
      robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_MASK);
    }
    println("RIGHT PRESSED");
  }
////////////////////// SUELTO MANO DERECHA ////////////////////////////
  else if(rpressed && rshoulderZ - rhandZ < armLenght)
  {
    rpressed = false;
    if(clickEnable){
      robot.mouseRelease(java.awt.event.InputEvent.BUTTON1_MASK);
    }
    println("RIGHT RELEASED");
  }
}
```

SwipeDetector.pde

```
void SwipeDetector()
{
    if(swipe[4].y - swipe[0].y > 800)
    {
      swipe[4].y = swipe[0].y;
      println("SWIPE DOWN");
      robot.mouseWheel(4);
    }
    if(swipe[4].y - swipe[0].y < -800)
    {
      swipe[4].y = swipe[0].y;
      println("SWIPE UP");
      robot.mouseWheel(-4);
    }
    if(swipe[4].x - swipe[0].x < -500)
    {
      swipe[4].x = swipe[0].x;
      println("SWIPE RIGHT");
      robot.keyPress(KeyEvent.VK_RIGHT);
      robot.keyRelease(KeyEvent.VK_RIGHT);
    }
    if(swipe[4].x - swipe[0].x > 500)
    {
      swipe[4].x = swipe[0].x;
      println("SWIPE LEFT");
      robot.keyPress(KeyEvent.VK_LEFT);
      robot.keyRelease(KeyEvent.VK_LEFT);
    }
}
```

B.2 Implementación usando rastreo de manos y dedos

KinecTracking.pde

```
import SimpleOpenNI.*;
import fingertracker.*;
import java.awt.*;
import java.awt.Robot;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
import ddf.minim.*;

//////////////////Inicializacion de Variables/////////////////////
SimpleOpenNI  context;
XnVHandPointContext right=null,left=null;
FingerTracker Rfingers,Lfingers;
XnVSessionManager sessionManager;
XnVFlowRouter flowRouter;
KinectTracking  kinectTracking;
Swipe swipe;
Robot robot = null;
String lastGesture="";
Boolean
zoomstate=false,lgrabstate=false,rgrabstate=false,rightclickstate=false,rpre
ssed=false,lpressed=false,clkstate=false,sound=false;
int
ptime=0,ltime=0,rtime=0,zoomtime=0,rsecure=0,pshtime=0,lsecure=0,zsecure=0,f
inqty=0,rightfinqty=0,leftfinqty=0;
int[] depthMap;
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Minim minim;
AudioSample snare;
int width = (int)screenSize.getWidth();
int height = (int)screenSize.getHeight();

///Configuracion de Variables y Camaras de Kinect///
void setup(){
  minim = new Minim(this);
  snare = minim.loadSample("Button.wav", 512);
  if ( snare == null ) println("Error al cargar el archivo");
  context = new SimpleOpenNI(this);
  context.setMirror(true);
  context.enableDepth();
  context.enableIR();
  context.enableGesture();
  context.enableHands();
  context.addGesture("Click");
  context.setSmoothingHands(.05);
  sessionManager = context.createSessionManager("RaiseHand", "RaiseHand");
  swipe = new Swipe();
  kinectTracking = new KinectTracking();
  flowRouter = new XnVFlowRouter();
  flowRouter.SetActive(kinectTracking);
```

```
  Rfingers = new
FingerTracker(this,context.depthWidth(),context.depthHeight());
  Rfingers.setMeltFactor(80);
  Lfingers = new
FingerTracker(this,context.depthWidth(),context.depthHeight());
  Lfingers.setMeltFactor(80);
  sessionManager.AddListener(flowRouter);
  sessionManager.AddListener(swipe);
  sessionManager.AddListener(kinectTracking);
  stroke(0,0,255);
  strokeWeight(3);
  smooth();
  size(640, context.depthHeight());
}

//////////////////Metodo Principal Aplicacion//////////////////
void draw()
{
  try{
      robot = new Robot();
      }
  catch(Exception e){System.out.println( e.toString() ); }
  context.update();
  context.update(sessionManager);
  depthMap = context.depthMap();
  Rfingers.update(depthMap);
  Lfingers.update(depthMap);
  image(context.irImage(),0,0);
  kinectTracking.draw();
}

/////Metodos para acciones en eventos de inicio de sesion y reconocimiento
de gestos de OpenNI y NITE//////////////////////

void onStartSession(PVector pos)
{
  println("onStartSession: " + pos);
}
void onEndSession()
{context.addGesture("Wave,Click");
  println("onEndSession: ");
}
void onFocusSession(String strFocus,PVector pos,float progress)
{
  println("onFocusSession: focus=" + strFocus + ",pos=" + pos + ",progress="
+ progress);
}
void onRecognizeGesture(String strGesture, PVector idPosition, PVector
endPosition){
  if(strGesture.equals("Click")){
  if(leftfinqty<2){
  robot.mousePress(InputEvent.BUTTON3_MASK);
  robot.mouseRelease(InputEvent.BUTTON3_MASK);}}
  if(leftfinqty>=2){robot.mousePress(InputEvent.BUTTON1_MASK);
  robot.mouseRelease(InputEvent.BUTTON1_MASK);
  robot.mousePress(InputEvent.BUTTON1_MASK);
  robot.mouseRelease(InputEvent.BUTTON1_MASK);}
```

```
  println("onRecognizeGesture - strGesture: " + strGesture +
    ", idPosition: " + idPosition + ", endPosition:" + endPosition);
}

void onProgressGesture(String strGesture, PVector position, float progress)
{
  println("onProgressGesture - strGesture: " + strGesture +
    ", position: " + position + ", progress:" + progress);
}


////////////////Clase Principal de la Aplicacion/////////////////
class KinectTracking extends XnVPointControl
{ float      zoom=0.0,zpush=0.0,psh=0.0,clk=0.0;
  color[]    _colorList = {
color(255,0,0),color(0,255,0),color(0,0,255),color(255,255,0)};

//////////////////// Metodos de eventos al crear puntos de control para las
manos reconocidas//////////////////////////

public void OnPointCreate(XnVHandPointContext cxt){
if(right==null||left==null){
if((right==null&&left==null)||(left==null&&right!=null&&right.getNID()!=cxt.
getNID())){
    left=cxt;
    println("OnPointCreate, lefthandId: " + cxt.getNID()+ " Hand:" +
left.getNID());}
if((right==null&&left==null)||(right==null&&left!=null&&left.getNID()!=cxt.g
etNID())){
    right=cxt;
    println("OnPointCreate, righthandId:" + cxt.getNID()+ " Hand:" +
right.getNID());}}
  }

public void OnPointDestroy(long nID)
  {
    if(right!=null&&0==right.getNID()){right=null;
                        println("OnPointDestroy, handId: " + nID);}
    if(left!=null&&0==left.getNID()){left=null;
       println("OnPointDestroy, handId: " + nID);
     }
  }

//////////////////////////Metodo que permite aumentar el movimiento del
puntero del mouse////////////////////////

  public PVector smoother(PVector current){
    if (current.x>0){
       current.set(current.x*3.5,current.y,current.z);}
    if (current.x<0){
       current.set(current.x*3.5,current.y,current.z);}
    if (current.y>0){
       current.set(current.x,current.y*2.5,current.z);}
    if (current.y<0){
       current.set(current.x,current.y*2.5,current.z);}
  return current;
  }
```

```
////////Método para obtener la distancia entre las manos//////////////
  public float gethandsDistance()
  {
  float d;
  PVector joint1 = new
PVector(right.getPtPosition().getX(),right.getPtPosition().getY(),right.getP
tPosition().getZ());
  PVector joint2 = new
PVector(left.getPtPosition().getX(),left.getPtPosition().getY(),left.getPtPo
sition().getZ());
  d = distance2D(joint1,joint2);
  return d;
  }
/////Metodo para calcular la distancia en 2D de las manos/////////
  public float distance2D(PVector point1, PVector point2)
  {
   float diff_x, diff_y;
   float distance;
   diff_x = point1.x - point2.x;
   diff_y = point1.y - point2.y;
   distance = sqrt(pow(diff_x,2)+pow(diff_y,2));
   return distance;
  }

////Metodos para saber si la distancia esta aumentando o disminuyendo////
  boolean isgrowing(float d){
    if(d>zoom+1.0){zoom=d;
                return true;}
    else{return false;}
  }

  boolean isdecreasing(float d){
    if(d<zoom-3.0){zoom=d;
                return true;}
    else{return false;}
  }

//////Metodo Principal de la Clase////////
  public void draw()
  { finqty=0;
    rightfinqty=0;
    leftfinqty=0;
    pushStyle();
      noFill();

      PVector vec;
      PVector mouse;
      PVector auxHand;
      PVector screenPos = new PVector();
      int colorIndex=0;

        auxHand = null;
        if((zoomstate||lgrabstate||rgrabstate)&&!sound){snare.trigger();
                                                        sound=true;}
        if(!zoomstate&&!lgrabstate&&!rgrabstate){sound=false;}
        if(right!=null){
```

```
        mouse=new
PVector(right.getPtPosition().getX(),right.getPtPosition().getY(),right.getP
tPosition().getZ());
        if(mouse != null){

          context.convertRealWorldToProjective(mouse,screenPos);
          drawFingers(screenPos,Rfingers,right);
          strokeWeight(8);
          stroke(0,255,0);
          point(screenPos.x,screenPos.y);
          movemouse(mouse,screenPos);
          }
        }
        if(left!=null){
        auxHand=new
PVector(left.getPtPosition().getX(),left.getPtPosition().getY(),left.getPtPo
sition().getZ());
        if(auxHand != null){
          strokeWeight(8);
          stroke(0,255,0);
          context.convertRealWorldToProjective(auxHand,screenPos);
          point(screenPos.x,screenPos.y);
          drawFingers(screenPos,Lfingers,left);
        }
      }

    popStyle();
    zoom();
    rightgrab();
    leftgrab();
  }

//////////Metodo que permite mover el mouse/////////

public void movemouse(PVector mouse,PVector screenPos){
    if(!zoomstate){
    mouse=smoother(mouse);
    context.convertRealWorldToProjective(mouse,screenPos);
    robot.mouseMove((((int)screenPos.x-30)*width)/560,(((int)screenPos.y-
30)*height)/400);
    text("Z-Distance: "+(int)(right.getPtPosition().getZ()),10,70);
    }
  }

////////Metodo de la funcionalidad Pan y Seleccion///////

public void leftgrab(){

if(right!=null&&left!=null&&leftfinqty<2&&finqty>=1&&!zoomstate&&!rgrabstate
){
    ltime++;
    if(ltime%25==0)lsecure++;}
    else{lgrabstate=false;
    ltime=0;
    lsecure=0;
    if(!lgrabstate&&lpressed){
    lpressed=false;
```

```
    robot.mouseRelease(InputEvent.BUTTON1_MASK);}
    return;}
    if(ltime>15&&lsecure>0){
    rtime=0;
    zoomtime=0;
    lgrabstate=true;
    if(!lpressed){
      robot.mousePress(InputEvent.BUTTON1_MASK);
      lpressed=true;
    }
    }
}

////////Metodo de la funcionalidad Scroll////////

public void rightgrab(){

if(right!=null&&left!=null&&rightfinqty<2&&finqty>=1&&!zoomstate&&!lgrabstat
e){
    rtime++;
    if(rtime%25==0)rsecure++;}
    else{rgrabstate=false;
    rtime=0;
    rsecure=0;
    if(!rgrabstate&&rpressed){
    rpressed=false;
    robot.mouseRelease(InputEvent.BUTTON2_MASK);}
    return;}
    if(rtime>=15&&rsecure>0){
    ltime=0;
    zoomtime=0;
    rgrabstate=true;
    if(!rpressed){
      robot.mousePress(InputEvent.BUTTON2_MASK);
      rpressed=true;
    }
    }
  }

//////////Metodo de la funcionalidad Zoom/////////////////

public void zoom(){

if(right!=null&&left!=null&&rightfinqty<=2&&leftfinqty<=2&&!rgrabstate&&!lgr
abstate){
    zoomtime++;
    if(zoomtime%25==0)zsecure++;}
    else{zoomtime=0;
    zoomstate=false;
    zsecure=0;}
    if(zoomtime>15&&zsecure>0){
    zoomstate=true;
    rtime=0;
    ltime=0;
    float handDistance = gethandsDistance();
    float d = map(handDistance, 0, 3000, 0, height);
    rect(10, 10, d, 15);
```

```
    text("Distance: "+(int)(d-20)+" centimeters",10,50);
    if(isgrowing(d)){
    robot.keyPress(KeyEvent.VK_CONTROL);
    robot.mouseWheel(-1);
    robot.keyRelease(KeyEvent.VK_CONTROL);}
    if(isdecreasing(d)){
    robot.keyPress(KeyEvent.VK_CONTROL);
    robot.mouseWheel(1);
    robot.keyRelease(KeyEvent.VK_CONTROL);}
    }
  }

// Metodo que dibuja y permite calcular la cantidad de dedos de las manos

public void drawFingers(PVector righthand, FingerTracker
fingers,XnVHandPointContext hand){
    float distance;
    PVector position,prevpos;
    fingers.setThreshold((int)hand.getPtPosition().getZ()+70);
    for(int i = 0; i < fingers.getNumFingers(); i++){
    position = fingers.getFinger(i);
    if (position.x<(righthand.x)+50&&position.x>(righthand.x)-
50&&position.y<(righthand.y)+20&&position.y>(righthand.y)-50){
    stroke(255,0,0);
    strokeWeight(5);
    point(position.x, position.y);
    if(hand==right){rightfinqty++;}
    else{leftfinqty++;}
    finqty++;}
    }
  }
}

//////////Clase del gesto Swipe de NITE///////////
class Swipe extends XnVSwipeDetector {
  Swipe() {
    RegisterSwipeRight(this);
    RegisterSwipeLeft(this);
    RegisterSwipeUp(this);
    RegisterSwipeDown(this);
    this.SetMotionTime(500);
    this.SetSteadyDuration(500);
  }

//////////Acciones a llevar a cabo al reconocer un Swipe//////////
  void onSwipe(float vel, float angle) {
  }
  void onSwipeUp(float vel, float angle) {
    if(!zoomstate&&!lgrabstate&&!pshstate){
    robot.keyPress(KeyEvent.VK_UP);
    robot.keyRelease(KeyEvent.VK_UP);}
  }
    void onSwipeDown(float vel, float angle){
    if(!zoomstate&&!lgrabstate&&!pshstate){
    robot.keyPress(KeyEvent.VK_DOWN);
    robot.keyRelease(KeyEvent.VK_RIGHT);}
  }
```

```
    void onSwipeLeft(float vel, float angle){
    if(!zoomstate&&!lgrabstate&&!pshstate){
    robot.keyPress(KeyEvent.VK_LEFT);
    robot.keyRelease(KeyEvent.VK_LEFT);}
  }
    void onSwipeRight(float vel, float angle){
    if(!zoomstate&&!lgrabstate&&!pshstate){
    robot.keyPress(KeyEvent.VK_RIGHT);
    robot.keyRelease(KeyEvent.VK_RIGHT);}
  }
}
```