Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Mecánica.
Automatización y Robótica.

Proyecto Robótica
” Esqueletización Mediante Kinect”

Profesor: Michael Miranda.

Alumnos: Iván Fernández

Claudio Canales D.

Índice

Ilustraciones

Resumen

El objetivo del presente proyecto del ramo Automatización y Robótica es determinar si la tecnología de Kinect puede aplicarse para el avance de tecnologías y la industria en procesos productivos. Se investigaron las posibilidades que el Kinect puede ofrecer, siendo una de las más novedosas la esqueletizacion de un individuo a través de la cámara RGB y el sensor infrarrojo de distancias. El primer paso para comprender esta tecnología fue determinar si era factible obtener los datos del Kinect en Windows en forma gratuita. El segundo paso es ver la posibilidad si se pueden manipular los datos recopilados del Kinect para calcular con algoritmos de alto nivel la esqueletizacion de un individuo. Para nuestra suerte Greg Borenstein ya había trabajado este problema por medio del programa Processing, por lo que solo se necesitó la librería SimpleOpenNI para obtener las coordenadas del esqueleto. Una vez obtenidas las coordenadas del esqueleto se necesita un medio de conexión y envió de datos entre aplicaciones o programas para ocuparlos en otras tecnologías, por ejemplo un brazo robótico. El protocolo OSC es usado en Processing para mandar datos a un puerto común donde pueden ser leídos por cualquier aplicación o programa. Finalmente una vez realizado con éxito el envió de datos a un puerto común, se buscó una plataforma que creara personajes con esqueletos para representar el movimiento de una persona real en un escenario virtual.

Introducción.

Como humano percibimos la estructura tridimensional del mundo que nos rodea con facilidad. Prueba de ello lo constatamos al mirar a nuestro alrededor. Somos capaces de identificar todos los objetos que nos rodean y distinguir los puntos que pertenecen a un objeto de los que corresponden al entorno.

Actualmente la rama de visión por computadora se encuentra en “State of the Art”, a lo que se refiere, que es un campo muy moderno y que está utilizando los métodos y técnicas más recientes, donde, queda mucho por investigar. En el presente, se puede obtener imágenes 3D de objetos a través de una serie de imágenes solapadas, también reconocer rostros dentro de una base de datos con una gran cantidad de personas y distinguir quien es quien, con un alto grado de confianza. Sin embargo, la capacidad de un ordenador para identificar objetos es bastante limitada y está bastante lejos de ser comparada a la habilidad de una persona para percibir el entorno.

Hoy en día, el análisis de imágenes y la visión por computadora se han vuelto un tópico importante en el desarrollo de diferentes áreas, como lo son la medicina, biomecánica, procesos industriales, entretenimiento digital, entre otros. La obtención y el tratamiento de la información es primordial para realizar análisis de las imágenes, hoy se pueden encontrar dispositivos que además de obtener una imagen 2D RGB, pueden generar una representación 3D, como lo hace el dispositivo fabricado por Microsoft denominado “Kinect for x360”.

Desde el año 2011 a la actualidad, diferentes desarrolladores e investigadores han utilizado el dispositivo Kinect, debido al bajo costo y facilidad de adquisición. Por ejemplo, investigadores en la Universidad de California han utilizado el dispositivo para conocer el comportamiento y la respuesta temporal de las vibraciones motoras de un paciente con Parkinson, esto ha permitido cuantificar la respuesta ante diferentes estímulos, fármacos y conocer una evolución temporal de la enfermedad. Todo lo anteriormente mencionado, fue logrado gracias a un sistema de control que era capaz de obtener los estímulos provenientes de las cámaras y interpretar esa información a través de un modelo de esqueletización.

Estructura.

El desarrollo de este proyecto fue planificado en un proceso de 4 fases, cada fase determina cómo será la siguiente debido a los resultados obtenidos. La fase 4 es el culmino exitoso del proyecto.

Interfaz de Desarrollo.

En el desarrollo de esta experiencia se utilizan diferentes plataformas de trabajo, a continuación se realiza una descripción y contextualización sobre qué consiste cada una de estas.

OpenNI

Tras el gran interés de los estudiantes que utilizan Kinect y la investigación de programas de código abierto, la empresa PrimeSense lanzó su propio software para trabajar con Kinect para que los programadores puedan acceder a la información de Kinect. Este se llama OpenNI (Open Natural Interaction) y es un SDK de código abierto utilizado para el desarrollo de bibliotecas y aplicaciones de middleware de detección 3D principalmente. Cabe destacar que el drive SDK de Microsoft proporciona desarrolladores que trabajan solamente en Windows, por ende OpenNI es la mejor opción para quien quiera trabajar en cualquier plataforma y lenguaje de programación.

NITE

Con el sistema OpenNI el programador puede acceder a los datos de profundidad y cámara de colores de Kinect. OpenNI está bajo una licencia. Sin embargo, una de las características más interesantes de OpenNI es el seguimiento de los usuarios que no está cubierta por la licencia. En su lugar esta proporcionada por un módulo externo llamado NITE. NITE no está disponible bajo una licencia de código abierto. Se trata de un producto comercial que pertenece a PrimeSense. Su código fuente no está disponible pero PrimeSense proporciona una licencia gratuita que puede utilizarse para hacer proyectos que utilizan NITE con OpenNI. Es preciso señalar que algoritmos utilizan la profundidad, el color, el IR y la información de audio recibidos del dispositivo de hardware, lo que les permite realizar funciones tales como la localización y el seguimiento a mano, un analizador de escenas (separación de usuarios de fondo), seguimiento del esqueleto del usuario exacto, reconocimiento de varios gestos y más.

Processing

Processing es una plataforma informática, de lenguaje de programación basado en Java que, entre una de todas las cosas que se puede utilizar, facilita procesos visuales con poco código y la obtención de datos en tiempo real. Mediante Processing podemos dibujar gráficos en dos y tres dimensiones facilitando la manipulación de los datos recolectados de Kinect v1.

El entorno de desarrollo de Processing (PDE) es sencillo y fácil de usar. Los programas se escriben en el editor de texto y se ejecutan pulsando start. Cada programa se escribe en un sketch que por defecto es una subclase de Papplet de Java que implementa la mayor parte de las funciones de Processing. También permite crear clases propias en el sketch pudiendo así usar tipos de datos complejos con argumentos y evita la limitación de utilizar exclusivamente datos tales como enteros, caracteres, números reales y color. Los sketchs se almacenan en sketchbook que no es más que un directorio dentro del ordenador.

Es posible ampliar las capacidades de Processing mediante el uso de extensiones y bibliotecas. Para utilizar Kinect en Processing se importan las librerías *"OpenCV for Processing"* (librería software open-source de visión artificial y machine learning con algoritmos permiten identificar objetos, caras, clasificar acciones humanas en vídeo, etc.) *"Open Kinect for Processing"* (Para usar el sensor de Microsoft Kinect en Processing y permite acceder a los datos de cámara RGB y de profundidad) y *"SimpleOpenNI"* (nos proporciona las capacidades de OpenNI en Processing, como por ejemplo la esqueletizacion e interfaz gráfica 2D y 3d de alto nivel).

Open Sound Control.

Open Sound Control (OSC) es un protocolo para la comunicación entre computadoras, sintetizadores de sonido y otros dispositivos multimedia que está optimizado para la tecnología de red moderna. Trayendo los beneficios de la tecnología de red moderna al mundo de los instrumentos musicales electrónicos, las ventajas de OSC incluyen interoperabilidad, precisión, flexibilidad y una mejor organización y documentación.

La gran mayoría del software y dispositivo OSC utiliza la red Ethernet o WiFI para transmitir datos. Eso significa que los mensajes OSC se pueden enviar fácilmente de una máquina a otra, o en la misma máquina en todas las aplicaciones.

Para que esto funcione, uno necesita un cliente OSC y un servidor OSC. Un cliente OSC es un dispositivo que envía datos OSC a un servidor OSC. Un servidor OSC recibe datos entrantes en un puerto que posee. Un puerto está simplemente representado por un número. La mayoría del software compatible con OSC es tanto un servidor como un cliente, lo que significa que puede enviar datos y recibir datos.

Un mensaje OSC está compuesto de dos partes: la dirección y los argumentos. La dirección es el nombre del mensaje OSC (por ejemplo, "BRAZO DERECHO"), y los argumentos son la lista de valores suministrados con este mensaje (por ejemplo, COORDENADAS DEL BRAZO DERECHO X Y Z).

Características:

- Esquema de nomenclatura simbólica de estilo URL abierto y dinámico
- Datos de argumento numéricos simbólicos y de alta resolución
- Patrón de coincidencia de idioma para especificar múltiples destinatarios de un solo mensaje
- Etiquetas de tiempo de alta resolución
- "Bundles" de mensajes cuyos efectos deben ocurrir simultáneamente
- Sistema de consulta para conocer dinámicamente las capacidades de un servidor OSC y obtener documentación

Áreas de aplicación:

- Instrumentos musicales electrónicos basados en sensores / gestos
- Asignación de datos no musicales al sonido
- Control musical compartido de múltiples usuarios

- Interfaces web
- Rendimiento musical LAN en red
- Rendimiento de WAN y Telepresence
- Realidad virtual
- Envolviendo otros protocolos dentro de OSC

Unity 3D

Unity 3D, es un motor gráfico 3D para PC que se utiliza para desarrollar juegos, aplicaciones interactivas, visualizaciones y animaciones en 3D. El éxito de Unity ha llegado en parte debido al enfoque en las necesidades de los desarrolladores independientes que no pueden crear ni su propio motor del juego ni las herramientas necesarias o adquirir licencias para utilizar plenamente las opciones que aparecen disponibles.

Unity posee un editor visual para poder crear los juegos en él, pues todo el contenido del juego se construye desde este editor y la forma en que los objetos se comportan, se programan usando un lenguaje de script (JavaScrip); esto anterior nos da a entender que no se necesita ser un experto en lenguajes como C++ para poder desarrollar un juego o una animación con Unity 3D.

Unity se estructura mediante el manejo y la creación de escenas para el desarrollo de la aplicación deseada, una escena puede ser cualquier parte del juego o la animación, ya sea un nivel del juego o un área determinada. Se empieza con un espacio en blanco en el cual se puede dar forma a todo lo que se desee crear usando las herramientas de Unity.

Este motor de Unity incluye además un editor de terrenos, donde se puede esculpir la forma del terreno usando las herramientas visuales que ofrece Unity, se puede pintar, texturizar, añadir hierba, colocar árboles o similares, o inclusive se permite la importación de otros materiales provenientes de otros motores de desarrollo.

Ilustración 1:Unity.

Blender

Blender es un programa informático multiplataforma dedicado al modelado, animación y creación de gráficos tridimensionales e interactivos.

Entre sus características puede:

- Modelado
- Texturizado
- Sistema de nodos para las texturas y materiales para mayor complejidad y profesionalismo
- Sistema de Huesos
- Sistema de partículas
- Animación
- Desarrollo de juegos en el sistema
- Tracking de cámara

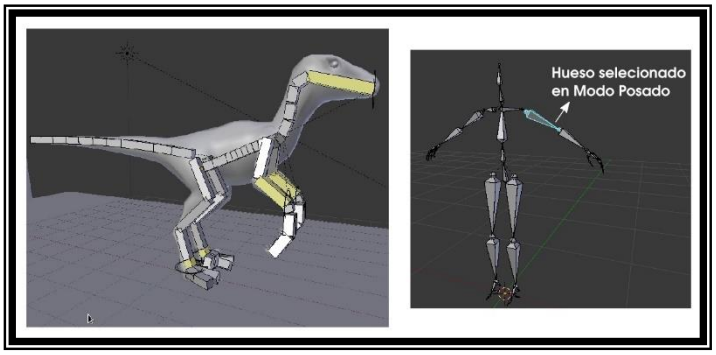


Ilustración 2:Blender.

Coordenadas euclidianas

Los ángulos de Euler constituyen un conjunto de tres coordenadas angulares que sirven para especificar la orientación de un sistema de referencia de ejes ortogonales, normalmente móvil, respecto a otro sistema de referencia de ejes ortogonales normalmente fijos. Dados dos sistemas de coordenadas xyz y XYZ con origen común, es posible especificar la posición de un sistema en términos del otro usando tres ángulos α, β y γ.

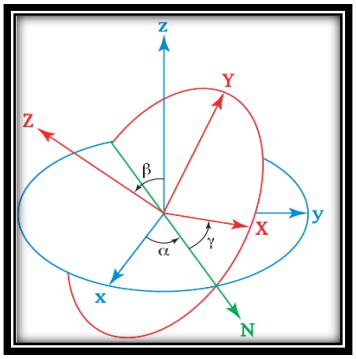


Ilustración 3:Coordenadas Euclidianas.

Objetivos

Fase 1

- Entender el funcionamiento de cada uno los dispositivos interiores de Kinect v1 y sus posibles usos para la ingeniería e industria
- Establecer la conexión y reconocimiento del Kinect v1 y la computadora para Windows
- Encontrar un entorno de desarrollo interactivo de código abierto que permita la manipulación de los datos recolectados del Kinect v1
- Aprender a utilizar y modificar el lenguaje de programación del entorno interactivo encontrado.

Fase 2.

- Encontrar un entorno de desarrollo interactivo de código abierto que permita la manipulación de los datos recolectados del Kinect v1
- Aprender a utilizar y modificar el lenguaje de programación del entorno interactivo encontrado

Fase 3

- Estudiar acerca de los métodos de comunicación entre dispositivos o aplicaciones para en envió y recibo de datos

Fase 4

- Ejecutar una simulación tridimensional en Unity de la esqueletización en Kinect por medio de OSC.
- Implementar el mismo programa diseñado, para manipular una interfaz de producción musical(Ableton).

Metas específicas

Fase 1

- Describir el funcionamiento de cada uno de los dispositivos del Kinect v1
- Instalar en la computadora los drivers apropiados para utilizar en Kinect v1 en la computadora.
- Aprender a utilizar y modificar el lenguaje de programación del entorno interactivo encontrado.

Fase 2.

- Instalar y aprender a utilizar el software libre Processing.
- Conocer los códigos para visualizar la cámara RGB en Processing.
- Conocer los códigos para visualizar el sensor infrarrojo de distancias en Processing.
- Interiorizar en los posibles usos de Kinect en Processing.
- Hallar librerías que obtengan el esqueleto de un individuo partir de algoritmos de alto nivel en Processing.
- Conectar a Matlab el Kinect v1 y evaluar si es óptimo para obtener la esqueletizacion de objetos.

Fase 3.

- Analizar y comprender el protocolo OSC
- Aprender a utilizar el protocolo OSC en conjunto con el programa Processing
- Utilizar el protocolo OSC con Processing para una determinada aplicación (Programa musical Ableton Live)

Fase 4.

- Realizar un acople entre el código diseñado en Processing con Unity.
- Incorporar el protocolo OSC en Unity.
- Diseñar un código en C# para transformar las coordenadas obtenidas en OSC y manipularlas en Unity.

Procedimiento

Fase 1

- Se buscan tesis, trabajos, proyectos y documentos en otras universidades y empresas asociados al desarrollo tecnológico de Kinect v1 para problemas de ingeniería.
- De los mismos documentos encontraros anteriormente y de las paginas oficiales de Kinect se estudia el funcionamiento de los componentes internos.
- Para conectar el Kinect con la computadora se buscan los drivers oficiales abiertos para las aplicaciones no comerciales y se instalan todas las versiones compatibles a los requisitos impuestos por el fabricante hasta que uno funcione correctamente en nuestra computadora.

Ilustración 4:Kinect.

Fase 2.

- De las páginas oficiales de los programas se descargar versiones coincidentes de OpenNI, NITE y Processing hasta encontrar una versión que sea compatible con la computadora.
- La versión compatible de los programas debe ser la correcta para que la librería SimpleOpenNI (Processing: Esqueletizacion) funciones correctamente, ya que los códigos interiores de esta librería son muy sensible a cambios de versión y de bites (32 o 64 bits)
- Una vez tener instalados correctamente los programas y la librería SimpleOpenNI, se procede a ver los ejemplos que trae esta librería y la del programa OpenNI. Esta trae consigo códigos que nos ayudan a entender la plataforma y los algoritmos para obtener la imagen de colores y distancia del Kinect v1.
- Después se busca un ejemplo de un código en la página Github para obtener los puntos de la esqueletizacion de un individuo en Processing por medio de la librería SimpleOpenNI. Se analiza y se comprende el codigo.
- Finalmente se compara él envió de datos de la esqueletizacion entre el programa Processing y Matlab donde se elige cual es mejor para nuestro proyecto.

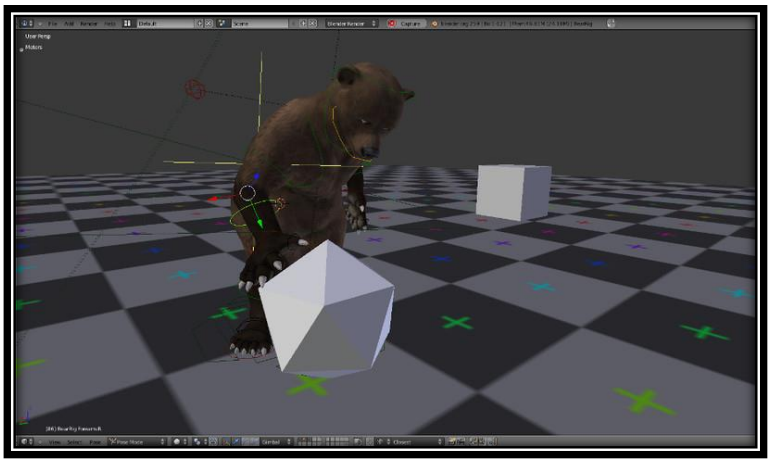
Fase 3.

- Se estudia acerca de la historia del protocolo OSC y su inicial aplicación.
- En la página Github se buscan tutoriales y códigos ya hechos por estudiantes en Processing y se analiza el código modificándolo continuamente para obtener un mayor aprendizaje.

Ilustración 5:OSC.

Fase 4.

- Instalar el programa Unity en la computadora.
- Buscar un personaje o esqueleto gratuito que Unity nos ofrezca y entender el comportamiento móvil de la posición de sus partes.
- Crear nuestro propio personaje con esqueleto en el programa Blender, lo más básico posible y parecido al esqueleto recolectado de Kinect v1.
- Representar el movimiento del esqueleto de Kinect v1 por medio de esferas en cada punto que nos ofrezca la librería SimpleOpenNI de Processing.



Desarrollo.

Fase 1.

Parte 1: Kinect v1 como desarrollo tecnológico

Para interiorizar y conocer el potencial de Kinect en el futuro para el crecimiento e innovación tecnología se buscan en la red proyectos ya comenzados a realizarse. Se encuentra tesis de algunos de universidades en España y algunas en chile (Universidad de concepción y Universidad de Chile). Algunos ejemplos recolectados de estas tesis son:

- Reconocimiento de esqueletos de distinta forma con métodos matemáticos vectoriales y de reconcomiendo en base a machine learning.
- Algoritmo de identificación de las posiciones para la poda de las parras mediante técnicas de procesamiento de imágenes.
- Implemento de algoritmos de procesamiento de imágenes para reconocer posición y color de objetos específicos en 3D (Esferas, cuadrados, triángulos)
- Movimiento de brazo robótico por medio de la posición de brazo humano reconocida por Kinect
- Sistema de control de televisión mediante Kinect y redes neuronales
- Generación de maniquís 3D a partir de imágenes del sensor Kinect
- Procesamiento de imágenes adquiridas por medio del sensor Kinect para determinar la posibilidad de una víctima en situaciones de peligro determinadas
- Moldeado 3D de cabeza y rasgos faciales mediante Kinect
- Variables cinemáticas de determinados puntos de control de un individuo mediante procesamiento de imágenes en Kinect
- Reconocimiento de gestos y actitudes corporales seminconscientes e inconscientes con visualizados con sensor Kinect

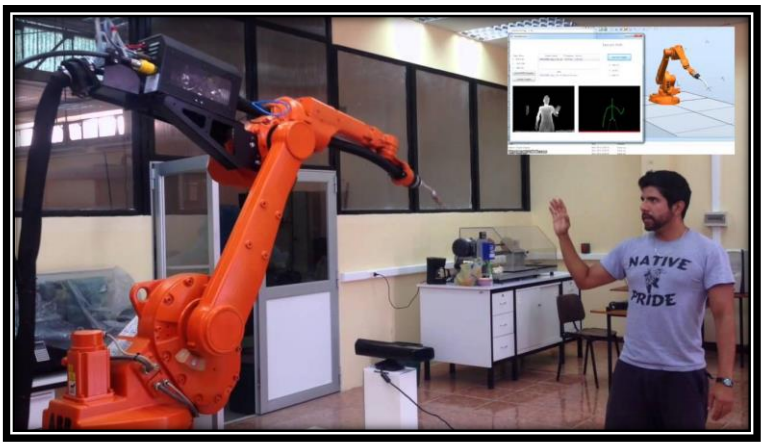


Ilustración 6:Implementación Industrial Kinect.

Parte 2: Descripción de los dispositivos interiores de Kinect v1

Ilustración 7:Arquitectura Kinect.

Los componentes que utiliza Kinect para la interacción con el individuo, la adquisición de imágenes y de distancia a objetos son:

- Sensores de profundidad infrarrojo: Compuesto por un proyector de rayos infrarrojos donde se refleja la luz en los objetos y son capturados por un sensor monocromático. El sensor detecta los segmentos de puntos reflejados y estima la profundidad a partir de la intensidad y la distorsión de los mismos. Captura 30 cuadros por segundo con una resolución de 640x480 píxeles.

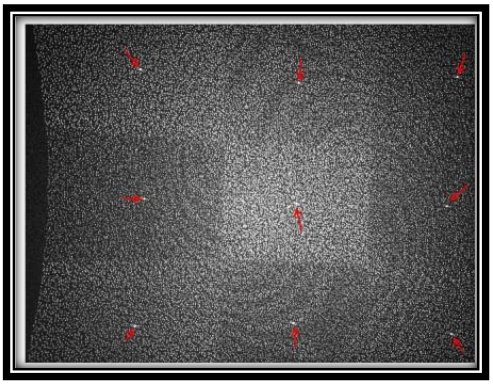


Ilustración 8:Imagen infrarroja.

- Cámara RGB: La cámara RGB (rojo, verde, azul) es del tipo CMOS, trabaja por defecto de 640x480 a 30 cuadros por segundo. Para transmitir los datos de los pixeles utiliza comunicaciones serial.

Ilustración 9: Imagen RGB.

- Micrófono Multi Matriz: Conjunto de cuatro micrófonos para el reconocimiento de órdenes y charla con el individuo.
- Inclinación motorizada: Permite ajustar la cámara hacia arriba o hacia abajo hasta 27º utilizada principalmente para la calibración de la cámara para utilizar un algoritmo que impone condiciones iniciales.

Parte 3: Drivers asociados a Windows

Kinect para Windows SDK v1.8:

El Software de Desarrollo de Kinect para Windows (SDK) es un conjunto de herramientas que permiten al programador crear aplicaciones capaces de interactuar con el dispositivo. Destinada únicamente destinada a la investigación y al desarrollo de aplicaciones no comerciales. Soporta lenguaje de programación C+ y C++ con Visual estudio.

Tiene los siguientes elementos:

Controladores de Microsoft para Kinect:

- Captura y procesado de sonido a través de la API de audio estándar de Windows.
- Captura y transmisión de datos de imagen y profundidad.
- Funciones que permiten a una aplicación utilizar más de un sensor Kinect conectado a la computadora.

NUI API

- Se trata de un conjunto de APIs que obtienen datos de los sensores de imagen y controlan los dispositivos.

AUDIO API

- API que soporta y gestiona el sistema de micrófonos.

Requisitos del Sistema:

Sistema operativo compatible:

- Windows 7, Windows 8, Windows 8.1, Windows Embedded Standard 7

Requisitos de hardware

- Su computadora debe tener las siguientes capacidades mínimas:
- Procesador de 32 bits (x86) o 64 bits (x64)
- Procesador de doble núcleo a 2.66 GHz o más rápido
- Bus USB 2.0 dedicado
- 2 GB de RAM
- Un sensor de Microsoft Kinect para Windows

Requisitos de Software

- Visual Studio 2010 o Visual Studio 2012. Las ediciones Express gratuitas se pueden descargar de Microsoft Visual Studio 2010 Express o Microsoft Visual Studio 2012 Express.
- NET Framework 4 (instalado con Visual Studio 2010) o .NET Framwork 4.5 (instalado con Visual Studio 2012)
- Para desarrollar Kinect habilitado para voz para aplicaciones de Windows, debe instalar el Microsoft Tech Platform SDK v11

Para instalar el SDK:

- Asegúrese de que el sensor Kinect no esté enchufado a ninguno de los puertos USB de la computadora.
- Si tiene instalada una versión anterior de Kinect para Windows SDK, cierre las muestras abiertas, el Sample Browser, etc. y salte al paso 5. Kinect para Windows v1.8 actualizará la versión anterior.
- Retire cualquier otro controlador para el sensor Kinect.
- Si tiene instalado Microsoft Server Speech Platform 10.2, desinstale los componentes de Microsoft Server Speech Platform Runtime y SDK, incluidas las versiones de bit x86 y x64, más el lenguaje de reconocimiento de voz de Microsoft Server - Kinect Language Pack.
- Cierre Visual Studio. Debe cerrar Visual Studio antes de instalar el SDK y luego reiniciarlo después de la instalación para recoger las variables de entorno que requiere el SDK.
- Una vez que el SDK haya terminado de instalarse correctamente, asegúrese de que el sensor Kinect esté enchufado a una fuente de alimentación externa y luego enchufe el sensor Kinect en el puerto USB de la PC. Los controladores se cargarán automáticamente.
- El sensor Kinect ahora debería estar funcionando correctamente.

Reconocimiento del dispositivo:

Para verificar la instalación se conecta el kinect por el Puerto USB y en ventana de administración de dispositivos aparecerá de la siguiente forma:

- Microsoft Kinect Audio Array Control
- Microsoft Kinect Camera
- Microsoft Kinect Device

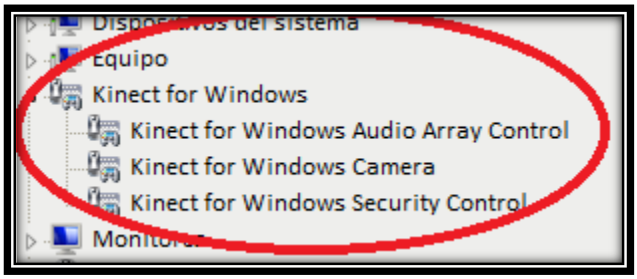


Ilustración 10:Driver Kinect.

Verifique antes que no exista ningún drivers de Kinect instalado en el computador ya que al conectar el Kinect, ya que Windows busca el drivers anteriormente instalado. Para eliminar drivers de forma más rápida y eficiente se ocupa el programa DriverStoreExplorer.v0.9.10.

Fase 2.

Parte 1: Instalación de entornos de desarrollo

OpenNI

Para descargar OpenNI SDK se accede a la página oficial http://openni.ru/openni-sdk/index.html y se descarga la versión para Windows *"OpenNI 2.2.0.33 Beta (x86) (32 bits)"*. Se abre el archivo descargado y se siguen las instrucciones que indica el setup:

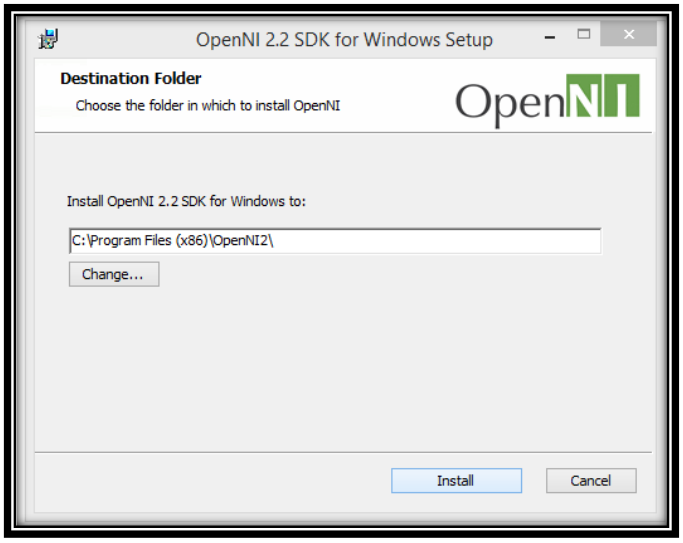


Ilustración 11:OpenNI.

Nota: El código fuente OpenNI está disponible en GitHub

Para verificar la instalación correcta, se abre Viewer OpenNI (aplicación de OpenNI) con el Kinect conectado. Debe aparecer en la interfaz la visualización de la cámara infrarroja o RGB.

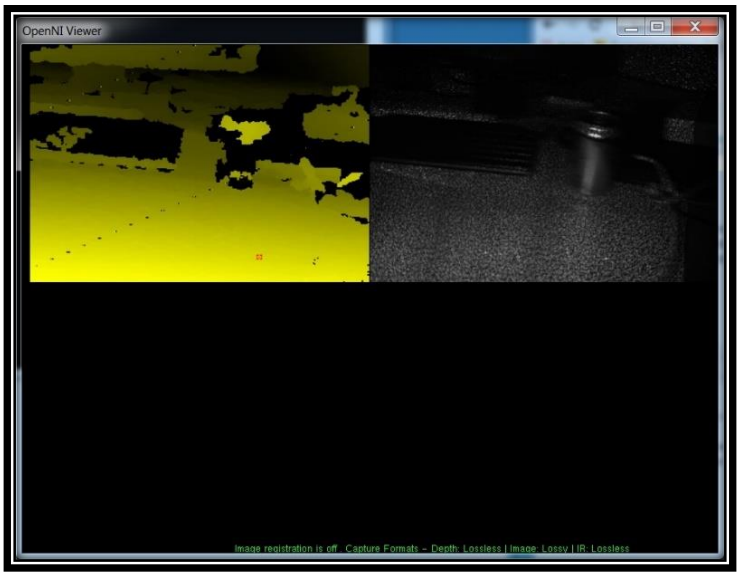


Ilustración 12: Viewer OpenNI.

NITE

El middleware 3D PrimeSense NITE también se descarga de la página oficial http://openni.ru/files/nite/index.html donde se descarga la versión "*NITE 2.2.0.11*". El único prerrequisito que pide es tener instalado OpenNI 2.2. Se aceptan las condiciones de termino y se apretar el botón instalar.

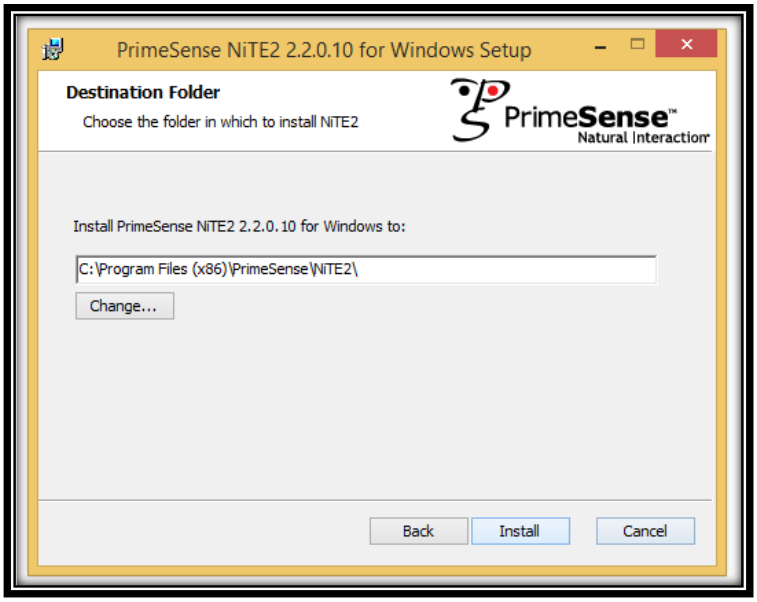


Ilustración 13:PrimeSense.

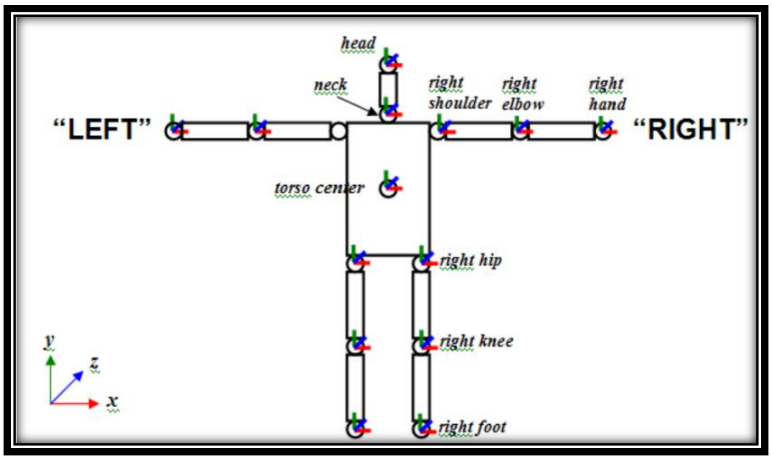


Ilustración 14:Esqueletizacion.

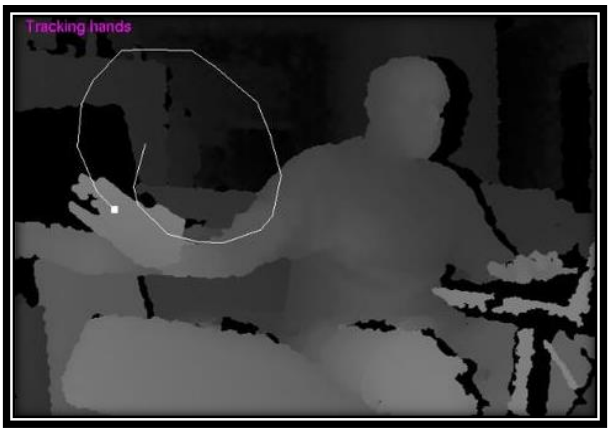


Ilustración 15:Imagen de Profundidad.

Processing

Para instalar Processing en la computadora hay que ingresar a la página oficial https://processing.org/. Se descargar la versión con sistema operativo Windows de 32 bits: "*Processing - 3.3.6 – Windows 32".* Para abrir Processing solo se debe abrir el archivo *processing.exe* de la carpeta comprimida descargada anteriormente.

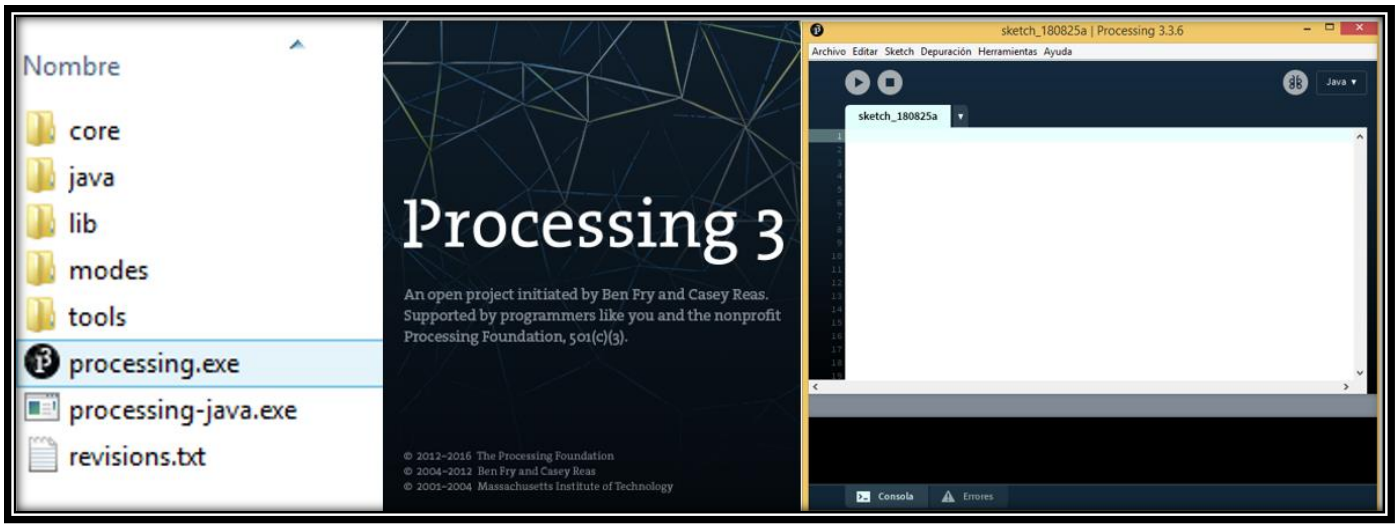


Ilustración 16:Processing.

Librerías Processing

Para buscar e importar librerías, desde la ventada de Processing se hace clic en *Sketch > Importar librerías > Añadir a la biblioteca*. Esto abrirá la ventana *Contribution Manager* donde se podrá buscar todo tipo de librerías de diferentes autores gratuitamente. Otra forma de importar una librerías es descargarla directamente de la red y el archivo de la librería copiarlo dentro de la carpeta de Processing llamada *"Libraries".* Para verificar una importación correcta se debe observar en *Sketch > Importar librerías* que abajo este el nombre de la librería requerida.

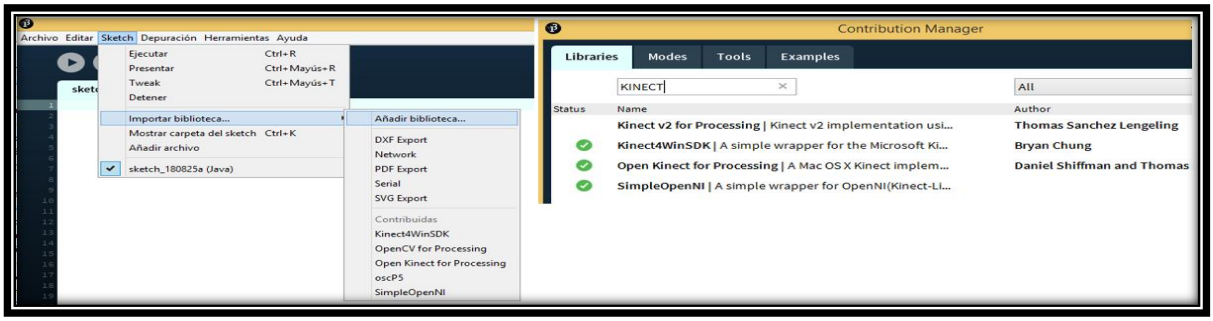


Ilustración 17:Librerias Processing.

Las librerías que se importaran para el reconocimiento de un esqueleto y el envió de los datos a cualquier aplicación son:

- OpenCV for Processing: con algoritmos permiten identificar objetos, caras, etc.
- Open Kinect for Processing: Para usar el sensor de Microsoft Kinect en Processing
- OscP5: Implementación Open Sound Control para la transmisión de datos.
- SimpleOpenNI: OpenNI en Processing

Nota: GitHub es una plataforma de desarrollo inspirada en la forma en que trabajas. Desde el código abierto hasta el negocio, puede alojar y revisar el código, administrar proyectos y crear software junto con 30 millones de desarrolladores. De esta página se debe descargar librería *SimpleOpenNI* y para encontrarla en el buscador se escribe: *simple-openni-master* o solo se ingresa a la página: https://github.com/wexstorm/simple-openni

Se pueden encontrar más versiones y archivos compatibles con sistemas operativos de la librería *SimpleOpenNI* de la página https://code.google.com/archive/p/simple-openni/downloads ya que está en constante mejora.

Parte 2: Códigos para obtener la cámara RGB y distancias a partir de sensor infrarrojo en Processing

sketch_1

```
//Lo primero es incluir las declaraciones de importación adecuadas en la parte superior de su código:
import org.openkinect.processing.*;

//Además de una referencia a un objeto Kinect, es decir
Kinect kinect;

//Luego setup(), puedes inicializar ese objeto kinect:
void setup() {
  kinect = new Kinect(this);
  kinect.initDevice();
  kinect.initDepth();
}

//Acceder a los datos del sensor de kinect
//Actualmente, la biblioteca pone los datos a su disposición de cinco maneras:
//PImage (RGB) de la cámara de video kinect.
//PImage (grayscale) de la cámara IR de Kinect.
//PImage (grayscale) con el brillo de cada píxel asignado a la profundidad (más brillante = más cerca).
//PImage (RGB) con el matiz de cada píxel asignado a la profundidad.
//int[] array con datos de profundidad sin formato (números de 11 bits entre 0 y 2048).


//Dibujar RGB y profundidad
void drag(){
  // Tamaño de la ventana en pixeles para la visualizacion de RGB o profundidad
  size(512,424);

  //Utilizar el Kinect como una cámara web vieja y normal
  //Con kinect v1 no se puede obtener tanto la imagen de video como la imagen de IR. Ambos son devueltos a través de getVideoImage ()
  PImage img = kinect.getVideoImage();
  image(img, 0, 0);

  //Imagen de profundidad en escala de grises:
  PImage img = kinect.getDepthImage();
  image(img, 0, 0);
}

//Datos de profundidad sin procesar:
//Los valores de profundidad sin procesar oscilan entre 0 y 2048
int[] depth = kinect.getRawDepth();

//Imagen de profundidad de color
kinect.enableColorDepth(true);

//FUNCIONES UTILES
//initDevice() - comienza todo (video, profundidad, IR)
//activateDevice(int) - activar un dispositivo específico cuando se conectan varios dispositivos
//initVideo() - solo iniciar video
//enableIR(boolean) - encender o apagar la imagen de la cámara IR (v1 solamente)
//initDepth() - solo comienza la profundidad
//enableColorDepth(boolean) - activar o desactivar los valores de profundidad como imagen en color
//enableMirror(boolean) - duplicar la imagen y los datos de profundidad
//PImage getVideoImage() - tomar la imagen de video RGB
//PImage getDepthImage() - agarra la imagen del mapa de profundidad
//int[] getRawDepth() - agarrar los datos de profundidad sin procesar
//float getTilt() - obtener el ángulo del sensor actual (entre 0 y 30 grados)
//setTilt(float) - ajustar el ángulo del sensor (entre 0 y 30 grados)
```

Ilustración 18: Código para representar las Imágenes de Kinect.

Parte 3: Posibles usos del Kinect en Processing

- Rastreo promedio de puntos (Centroide)

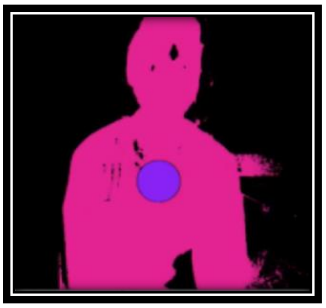


Ilustración 19:Centroide.

- Detección punto más cercano, más alto, más bajo, más alejado

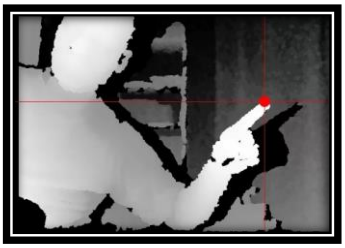


Ilustración 20:Punto más cercano.

- Detección de bordes

Ilustración 21:Detección de bordes.

- Mapeo 3D de superficies

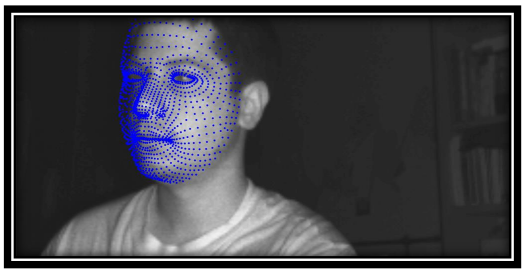


Ilustración 22:Mapeo 3D de superficies.

Parte 4: Esqueletizacion en Processing (SimpleOpenNI)

Se obtiene el modelo del esqueleto a través de la utilización de la librería SimpleOpenNI. Para una descripción más detallada del código se encuentra información en el libro *"Making.Things.See - Greg.Borenstein".*

```
1.  //Se importa la libreria SimpleOpenNI
2.  import SimpleOpenNI.*;
3.  //Se inicializa la instancia context.
4.  SimpleOpenNI  context;
5.  //se utiliza este Array para los  colores de las personas detectadas.
6.  color[]       userClr = new color[]{ color(255,0,0),
7.                                       color(0,255,0),
8.                                       color(0,0,255),
9.                                       color(255,255,0),
10.                                      color(255,0,255),
11.                                      color(0,255,255)
12.                                    };
13. PVector com = new PVector();
14.
15. PVector com2d = new PVector();
16.
17. void setup()
18. {
19.   //Se inicializa una ventana de 640*480 donde aparecerá el esqueleto
20.   size(640,480);
21.   //Se crea la instancia context
22.   context = new SimpleOpenNI(this);
23.   if(context.isInit() == false)
24.   {
25.      println("No se puede inicializar SimpleopenAI, no esta conectada la camara");
26.      exit();
27.      return;
28.   }
29.   // Permite la obtención de la profundidad
30.   context.enableDepth();
31.   // Crea el esqueleto para todas las uniones
32.   context.enableUser();
33.  //Fondo de la pantalla
34.   background(200,0,0);
35.   stroke(0,0,255);
36.   strokeWeight(3);
37.   smooth();
38. }
39.
40. void draw()
41. {
42.   // Actualiza la cámara
43.   context.update();
44.
45.   //Dibuja la imagen de profundidad
46.
47.   //image(context.depthImage(),0,0);
48.
49.   //Presenta la imagen RGB en la ventana, en la posición (0,0)
50.
51.   image(context.userImage(),0,0);
52.
```

```
53.
54.
55.  // Dibuja el esqueleto si esa disponible
56.
57.  //Declara un Array de los usuarios en la pantalla
58.   int[] userList = context.getUsers();
59.   for(int i=0;i<userList.length;i++)
60.   {
61.     //Prueba para saber si está realizando Esqueletizacion
62.
63.     if(context.isTrackingSkeleton(userList[i]))
64.     {
65.       stroke(userClr[ (userList[i] - 1) % userClr.length ] );
66.
67.       //LLAMA A LA LA FUNCION DRAW , QUE ESTA AL INFERIOR
68.
69.       drawSkeleton(userList[i]);
70.     }
71.     // Dibuja el centro de masa
72.
73.     if(context.getCoM(userList[i],com))
74.     {
75.       context.convertRealWorldToProjective(com,com2d);
76.       stroke(100,255,0);
77.       strokeWeight(1);
78.       beginShape(LINES);
79.         vertex(com2d.x,com2d.y - 5);
80.         vertex(com2d.x,com2d.y + 5);
81.
82.         vertex(com2d.x - 5,com2d.y);
83.         vertex(com2d.x + 5,com2d.y);
84.       endShape();
85.
86.       fill(0,255,100);
87.       text(Integer.toString(userList[i]),com2d.x,com2d.y);
88.     }
89.   }
90. }
91.
92.       // Dibuja el esqueleto con las uniones seleccionadas
```

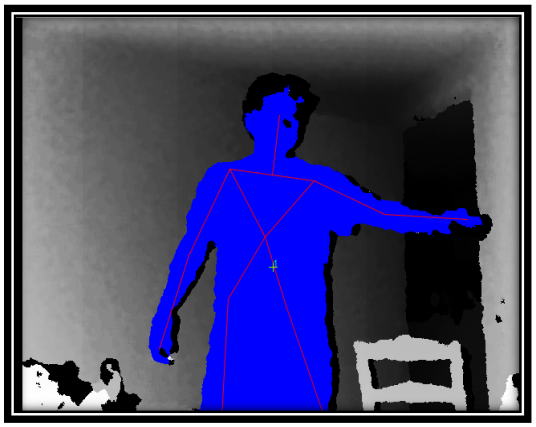


Ilustración 23: Esqueletización Desarrollada.

Parte 5: Conectar Kinect en Matlab y obtener esqueleto

Para utilizar Kinect en el programa Matlab se debe descargar un paquete llamado *"Image Acquisition Toolbox"* que permite adquirir datos del sensor de imagen directamente en MATLAB y Simulink. El código para visualizar el esqueleto es el siguiente:

```
clc; clear all ; close all
% INICIALIZACION DE KINECT, CAMARA RGB Y SENSOR DE DISTANCIA
imaqreset;
depthVid= videoinput('kinect',2);
triggerconfig (depthVid, 'manual');
depthVid.FramesPerTrigger=1;
depthVid.TriggerRepeat=inf;
set(getselectedsource(depthVid),'TrackingMode','Skeleton');
viewer=vision.DeployableVideoPlayer();
start(depthVid);
himg=figure;
pause(5)
% SELECCIONAR PUNTOS SELECCIONADOS Y LOS DIBUJA
while ishandle(himg)

    trigger(depthVid);
    [depthMap,~,depthMetaData]=getdata(depthVid);
    %[imgColor, ts_color, metaData_Color] = getdata(vid2);
    %imshow(depthMap,[0 4096]);
    %imshow(vid2)
    %size(depthMetaData)
    if sum(depthMetaData.IsSkeletonTracked)>0
        skeletonJoints = depthMetaData.JointDepthIndices(:,:,depthMetaData.IsSkeletonTracked);
        %hold on;
        %plot(skeletonJoints(:,1), skeletonJoints(:,2),'*');
        %for ii=1:20
        %    t(ii)=depthMap(skeletonJoints(ii,1), skeletonJoints(ii,2))
        %end
        %plot3(t,skeletonJoints(:,1), skeletonJoints(:,2),'*');
        %cont=0;

        % SELECCIONA PUNTOS DEL ESQUELETO DENTRO DE 640X480
        iii=0;
        for ii=1:20
            n=skeletonJoints(ii,1);
            m=skeletonJoints(ii,2);
            if (((n<=0)||(m<=0))||((n>640)||(m>480)))
                %cont=1
                %print ('algo malo punto')
                %pause(1)
            else
                %cont=2
```

```
                iii=iii+1;
                nn(iii)=skeletonJoints(ii,1);
                mm(iii)=(skeletonJoints(ii,2));
                mmm(iii)=floor(-(skeletonJoints(ii,2))+481) ;
                t(iii)=depthMap(mm(iii),nn(iii));
            end
        end
        %DA VUELTA LA IMAGEN O PUNTOS
        cont=0
        for ii=1:iii
            if ((abs(t(3)-t(ii)))<1000)
               cont=cont+1
               nnx(cont)=nn(ii);
               mmx(cont)=mm(ii);
               mmmx(cont)=mmm(ii) ;
               tx(cont)=depthMap(mmx(ii),nnx(ii));
            end
        end

        % DIBUJAR PUNTOS DE DISTANCIA SELECCIONADOS
        plot3(tx,nnx,mmmx,'*');
        hold on
        plot3(t(3),nn(3),mmm(3),'r*');
        hold off
        axis([0,4096,0,640,0,480]);
        grid on
    end
end
```

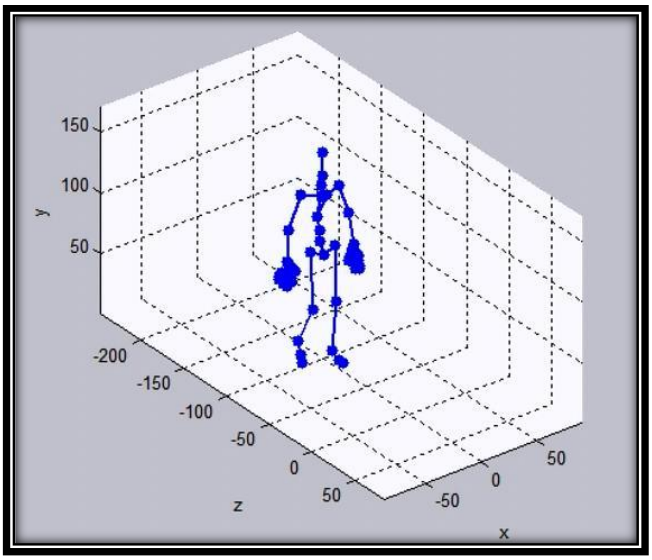


Ilustración 24:Esqueletización Matlab 3D.

Ilustración 25:Esqueletizacion 2D Matlab.

Fase 3.

Parte 1: Mensajes OSC solo con Processing

Para enviar un mensaje desde aplicaciones o dispositivitos se deben realizar los siguientes pasos:

- Conocer la IP del servidor y el puerto por donde se enviara el mensaje
- Crear el mensaje con su Nombre y Valor (array, entero, string, etc)
- Enviar el mensaje creado a la localización del servidor
- Para que el servidor lo lea debe conocer la IP del Cliente y el puerto por donde envía el mensaje

A continuación se presenta un esquema representativo del protocolo OSC para enviar mensajes desde un computador a una Tablet:

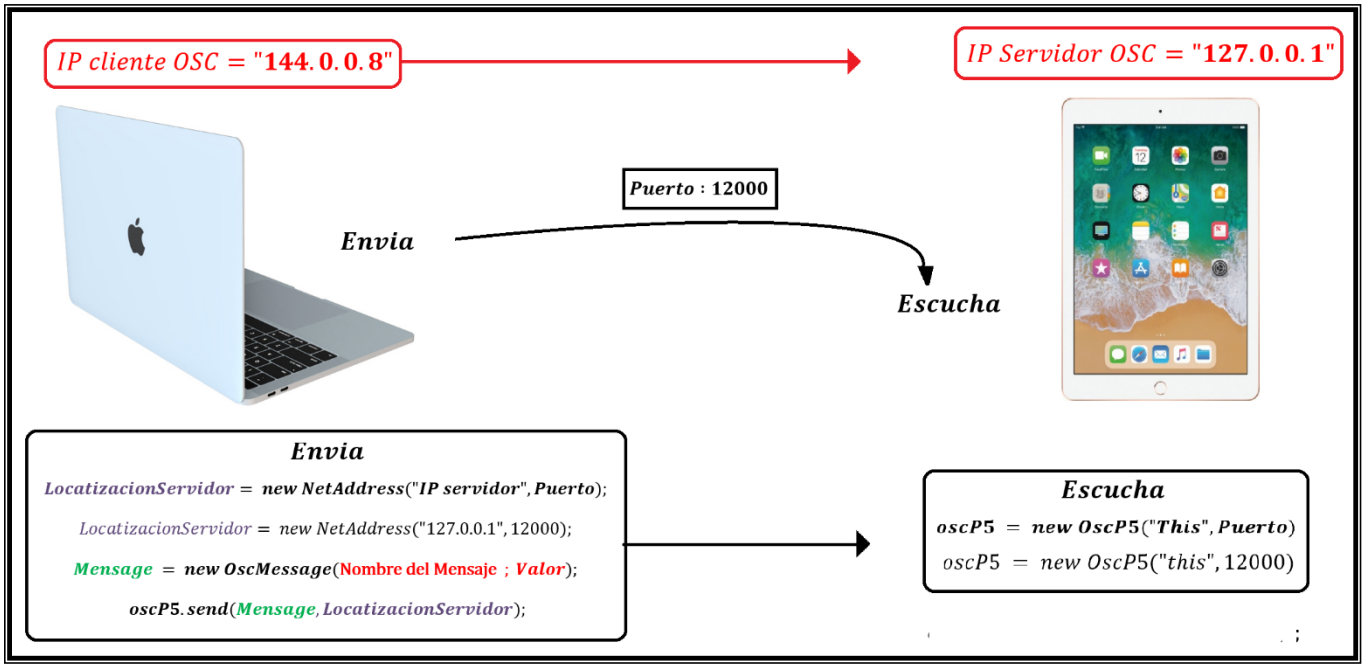


Ilustración 26:Protocolo OSC.

Código ejemplo para enviar y recibir mensajes OSC al mismo computador al hacer clic con el mouse en una interfaz gráfica en Processing:

```
/**
* El ejemplo muestra cómo enviar y recibir mensajes de osc.
 */
import oscP5.*;
import netP5.*;
OscP5 oscP5;
NetAddress myRemoteLocation;

void setup() {
  size(400,400);
  frameRate(25);
  // * inicia oscP5, escuchando los mensajes entrantes en el puerto 12000 * /
  oscP5 = new OscP5(this,12000);

   /* myRemoteLocation es una dirección de red. una NetAddress toma 2 parámetros,
    * una dirección IP y un número de puerto. myRemoteLocation se usa como parámetro en
    * oscP5.send () cuando se envían paquetes de osc a otra computadora, dispositivo,
    * solicitud. uso ver a continuación. para fines de prueba, el puerto de escucha
    * y el puerto de la dirección de ubicación remota son iguales, por lo tanto,
    * enviar mensajes de vuelta a este boceto.
    */
     myRemoteLocation = new NetAddress("127.0.0.1",12000);
}

void draw() {
  background(0);
}

void mousePressed() {
 /* en las siguientes formas diferentes de crear mensajes osc se muestran con el ejemplo */
  OscMessage myMessage = new OscMessage("/test");
  myMessage.add(123); /* agrega un int al mensaje osc */

  /* Envia el mensaje */
  oscP5.send(myMessage, myRemoteLocation);
}

/* incoming osc message se reenvía al método oscEvent. */
void oscEvent(OscMessage theOscMessage) {
/* imprime el patrón de dirección y la etiqueta de tipo del OscMessage */
  print("### received an osc message.");
  print(" addrpattern: "+theOscMessage.addrPattern());
  println(" typetag: "+theOscMessage.typetag());
}
```

Parte 2: Mensajes OSC aplicados a programa Ableton Live más plugin Live Grabbel

Ableton Live:

Ableton Live es un secuenciador de audio y MIDI, aplicación también conocida como DAW (Digital Audio Workstation) para el sistema operativo Windows y macOS. Live es un software rápido, fluido y flexible para la creación y el rendimiento musical. Viene con efectos, instrumentos, sonidos y todo tipo de características creativas, todo lo que necesitas para hacer cualquier tipo de música.

Live Grabber:

Los plugin de Live Grabber son un conjunto de complementos gratuitos de Max For Live que envían acciones desde Ableton Live a cualquier dispositivo de la red que admita Open Sound Control (OSC). A partir de la versión 3, Live Grabber también permite lo contrario: controlar Ableton Live con OSC.

Metodología:

En esta segunda parte del laboratorio se controlar el volumen de una pista musical reproducida por Ableton Live. Para controlar el volumen se envían datos de números enteros de 0 a 100, donde con 0 no se escucha la pista y con 100 esta al máximo volumen. Estos datos son enviados por el protocolo OSC a un puerto en específico que lo lee Ableton a través de su plugin Live Grabber. Del programa Processing se envían los datos (0 a 100) a el puerto correspondiente, donde estos son recolectados de la posición de la mano derecha del esqueleto de un individuo. Este esqueleto es obtenido gracias al dispositivo Kinect y la librería SimpleOpenNI utilizada en Processing. El punto más alto donde pueda estar la mano se mandara por OSC el valor 100 y el más bajo será 0.

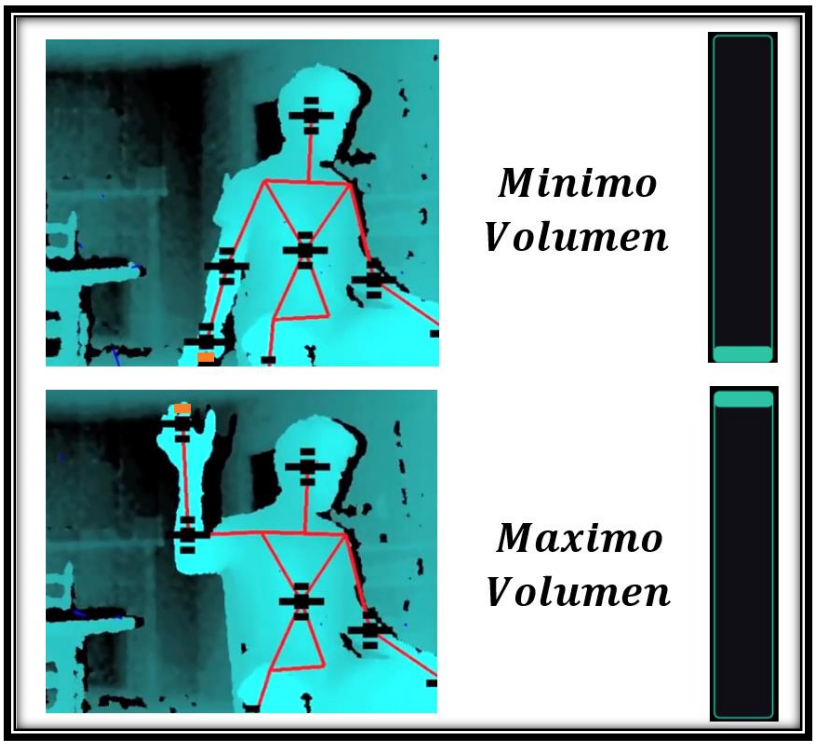


Ilustración 27:Efectos Ableton Kinect.

Ableton Live recibiendo los datos:

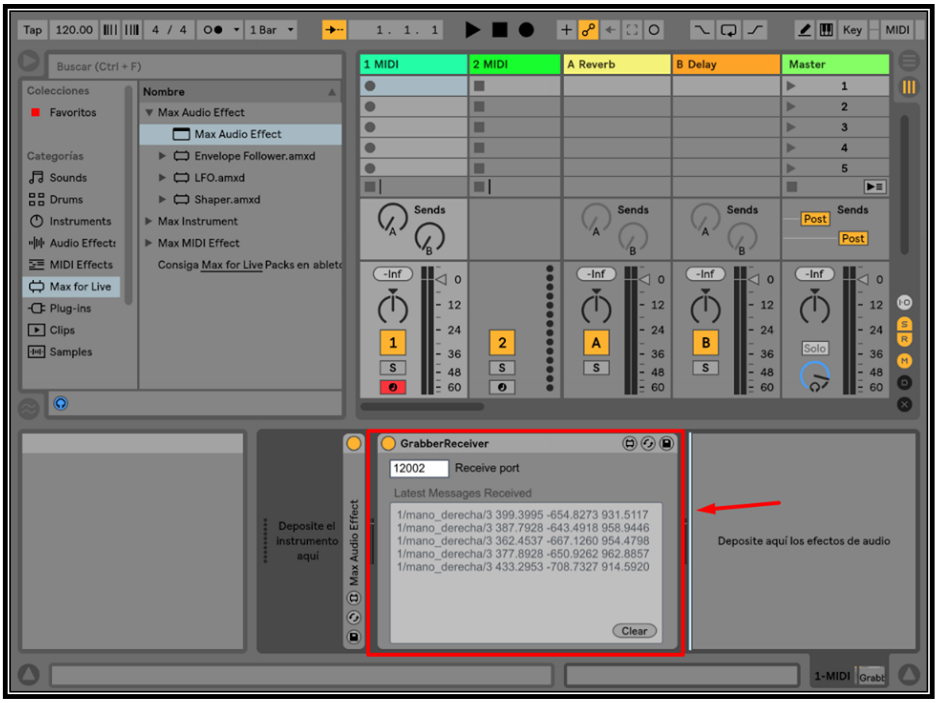
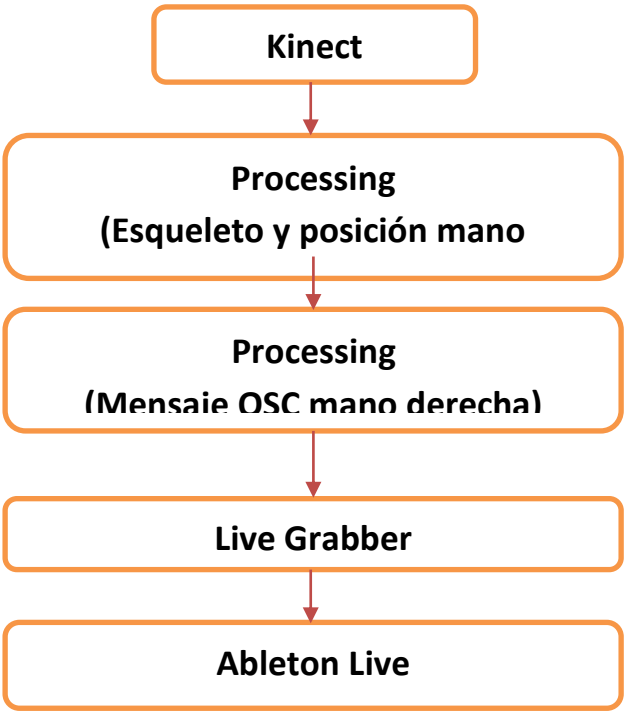


Ilustración 28:Ableton Recibiendo OSC.

Diagrama de bloques del procesamiento:



Fase 4

Parte 1: Transformada de posiciones en Unity y Comunicación OSC para recibir mensajes

Para conocer la plataforma del programa primero se descarga un robot gratuito de la página oficial de Unity llamado "*Space Robot Kyle".* Este robot es un personaje que camina en un terreno vacío.

Para descargarlo se debe abrir el programa Unity y hacer clic en la pestaña *Asset Store:*

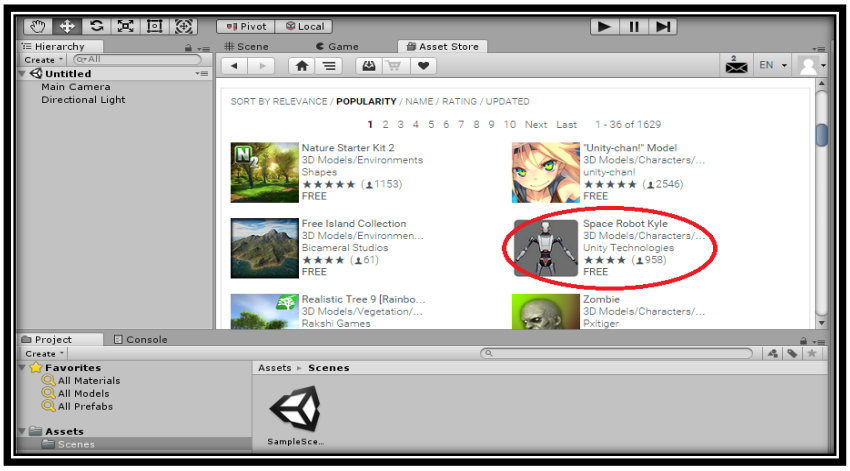


Ilustración 29:Asset.

Una vez descargado el robot se importan los archivos en el programa:

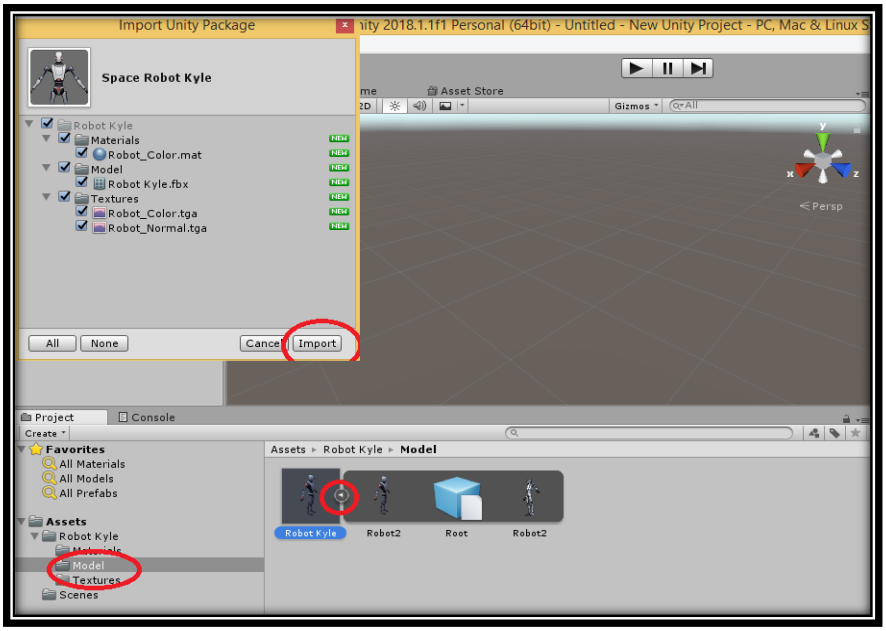


Ilustración 30:Modelo 3D.

Para obtener un control del robot se debe crear una animación Humanoide de él, es decir, además de las capas superficiales, se crean huesos y nodos para producir el movimiento del personaje. Estos nodos serán conectados posteriormente con las posiciones del esqueleto de una persona capturada por el Kinect v1.

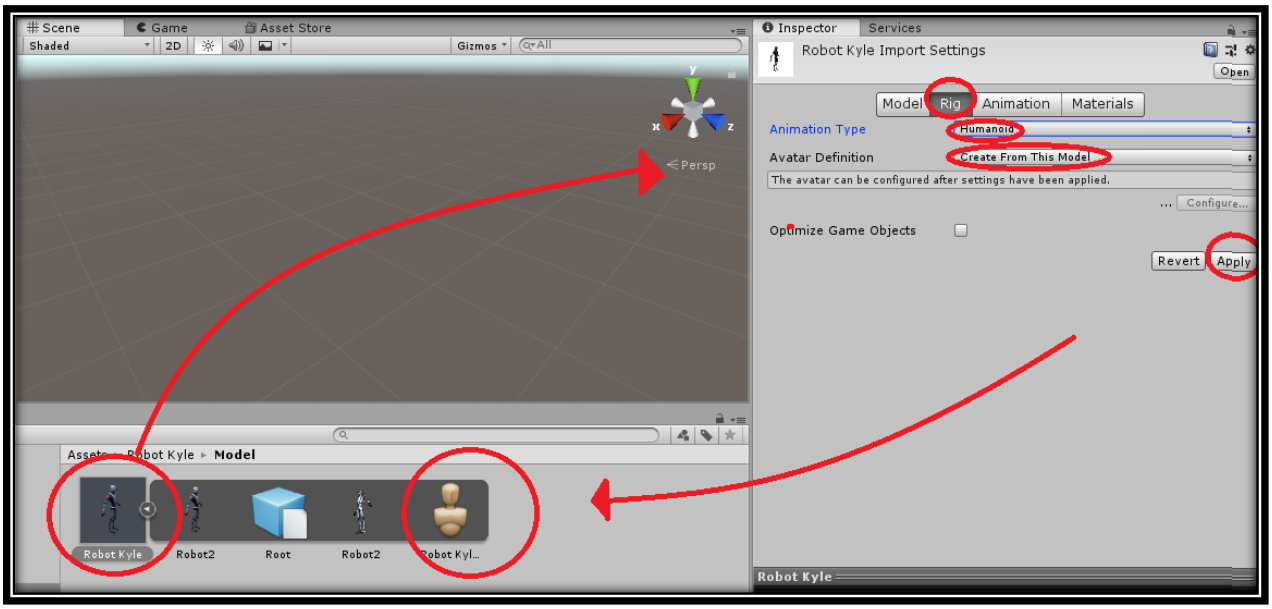


Ilustración 31:Propiedades Modelo.

Representación grafica de nodos y huesos del robot:

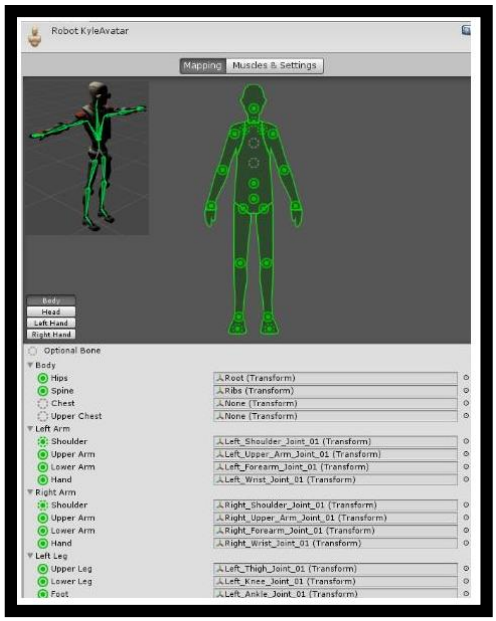


Ilustración 32:Modelo Esqueleto.

Se aprecia en la parte izquierda del programa un árbol de huesos del robot, con el nombre del hueso y su nivel jerarquía en el movimiento. Por ejemplo en este caso en el estómago está el elemento con más jerarquía, por ende si se cambian las coordenadas de este punto, todas las demás también lo hacen. A este grado jerárquico se les llama comúnmente como huesos padres y huesos hijos.

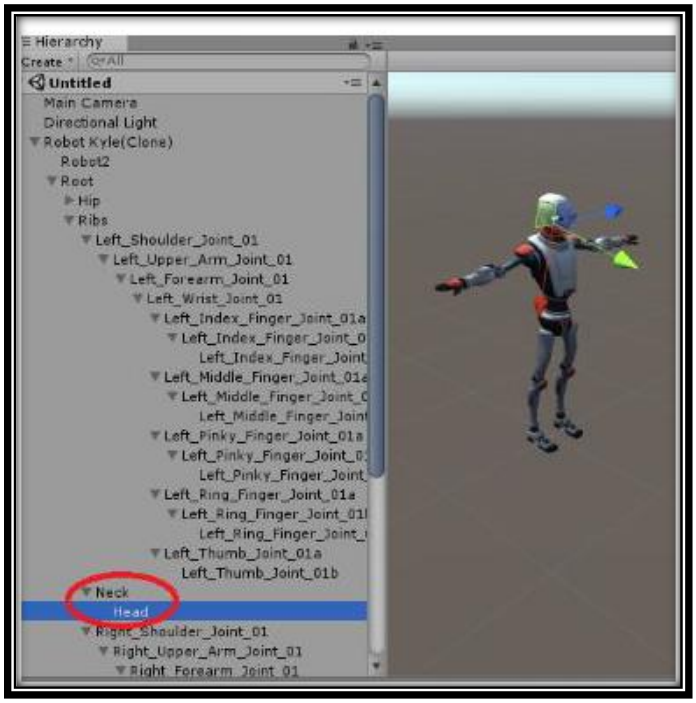


Ilustración 33:Huesos modelo 3D.

Para modificar la posición de un hueso en específico se hace clic en el nombre del hueso y aparece al lado derecho de la ventana del programa las posiciones en las coordenadas globales(XYZ) , la rotación de los huesos (Coordenadas euclidianas) y la escala.

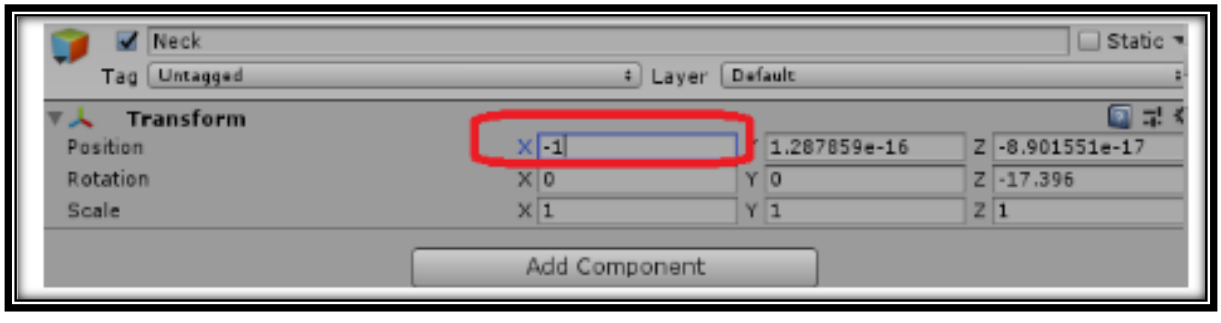


Ilustración 34:Posicion de Nodos.

Para entender mejor el funcionamiento del robot en la plataforma se cambian manualmente los datos de las posiciones de los huesos. Por ejemplo para la cabeza del robot (Head) y la rodilla derecha (Neck) se modificaron las posiciones entregando el siguiente resultado:

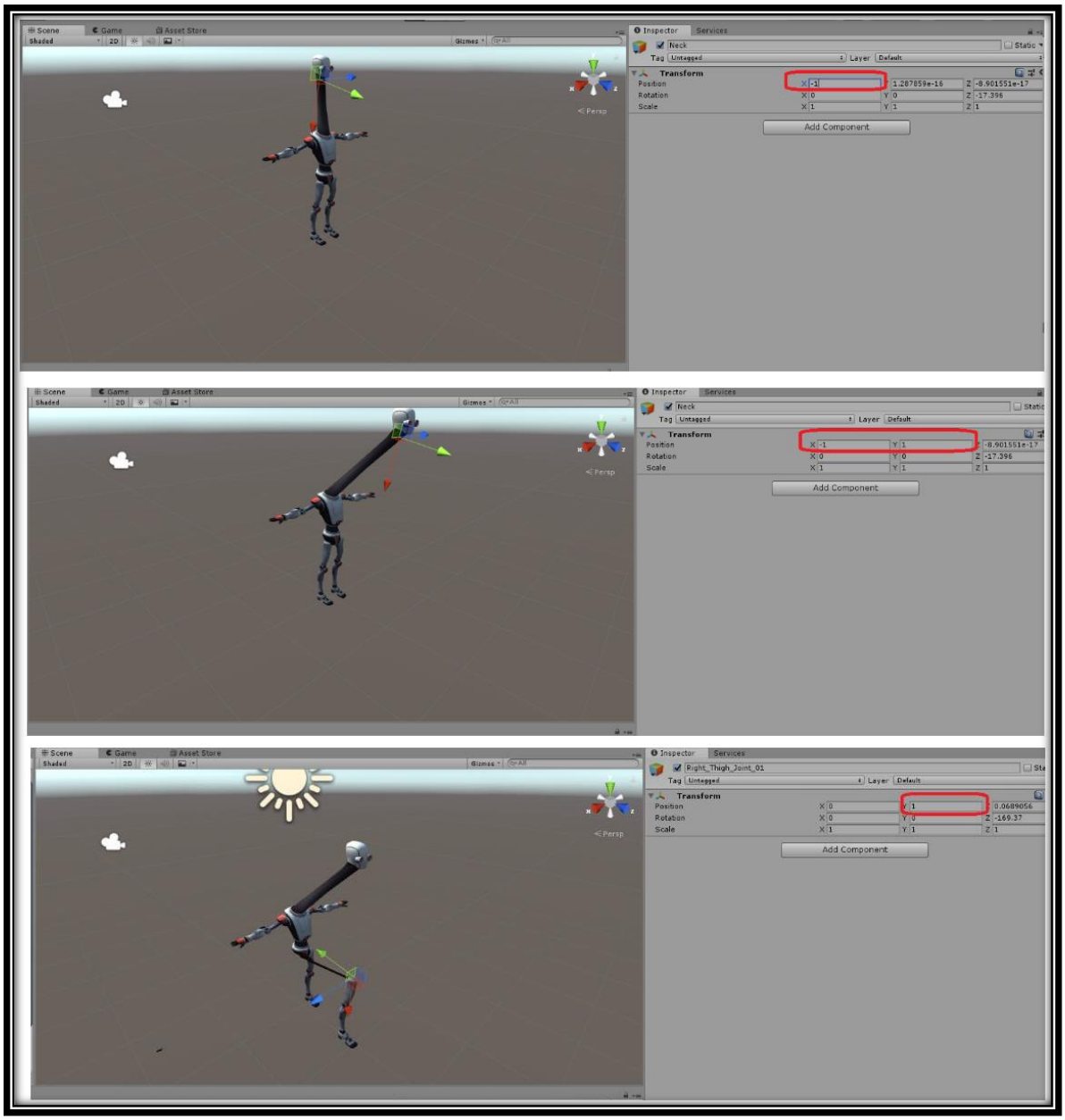


Ilustración 35:Experimentación Modelo.

Para conectar las posiciones del robot con los datos del esqueleto calculados en Processing por la librería SimpleOpenNI (donde está librería obtiene las posiciones del esqueleto de una persona a través de Kinect V1), se crea un canal de comunicación por medio del protocolo OSC.

Para utilizar la comunicación OSC primero se tiene que importar el código a Unity de la siguiente manera:

- Importe el script OSC.cs en el proyecto arrastrándolo a sus activos (el script se puede encontrar en la carpeta "Activos" del proyecto Unity).
- Crea un GameObject vacío y arrastra el OSC.cs importado sobre él.
- Configure el puerto OSC y la configuración de IP para que coincida con sus necesidades.

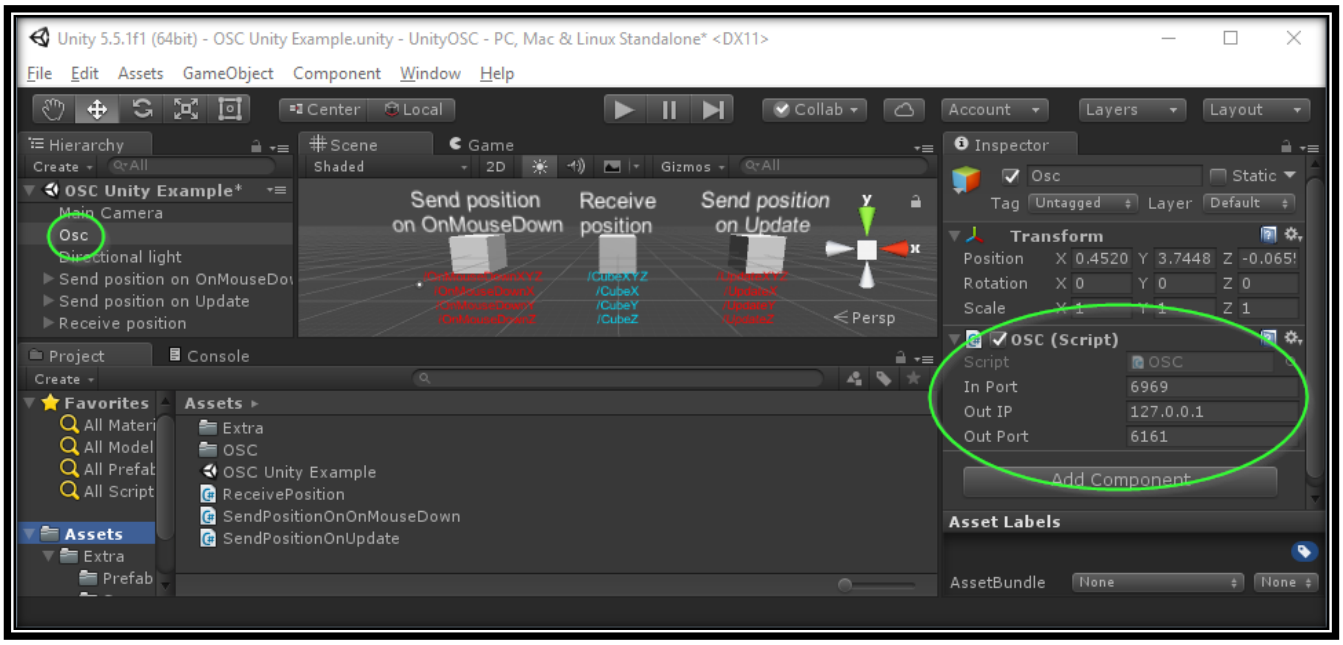


Ilustración 36: OSC Unity.

Para recibir mensajes OSC se deben realizar los siguientes pasos:

- Crea una nueva secuencia de comandos.
- Cree una referencia al script OSC en el *gameObject* vacío que ha creado: OSC (osc público).
- En *Start ()* establezca el nombre de una función a llamar cuando se recibe un mensaje OSC específico: *osc.SetAddressHandler (adress_nombre ;OnReceiveXYZ).*
- Crea la función que recibirá el mensaje OSC: *OnReceiveXYZ(OscMessage message)*
- Obtenga los datos del mensaje con *GetFloat*
- Modifique las posiciones nuevas con la función *transform.position*

ReceivePosition.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class ReceivePosition : MonoBehaviour {
5
6      public OSC osc;
7      public string Adress_nombre;
8       float factor_escala=200.0f;
9      // Use this for initialization
10     void Start () {
11        osc.SetAddressHandler(Adress_nombre, OnReceiveXYZ );
12
13     }
14
15     // Update is called once per frame
16     void Update () {
17
18     }
19
20     void OnReceiveXYZ(OscMessage message){
21         float x = message.GetFloat(0)/factor_escala;
22          float y = message.GetFloat(1)/factor_escala;
23         float z = message.GetFloat(2)/factor_escala;
24
25         transform.position = new Vector3(x,y,z);
26     }
27 }
28
```

Ilustración 37:Codigo Desarrollado en C#.

En este ejemplo (ReceivePosition.cs), el script manejará mensajes OSC con las siguientes direcciones (Posición de los puntos del esqueleto de Kinect v1):

“/Head"
"/Neck"
"/L_Shoulder"
"/R_Shoulder"
"/L_Elbow"
"/R_Elbow"
"/R_Hand"
"/L_Hand"
"/Torso"
"/L_Knee"
"/R_Knee",
"/L_Hip",
"/R_Hip",
"/L_Foot",
"/R_Foot"

Finalmente se establece la referencia al script OSC.cs para cada hueso que se desee cambiar la posición:

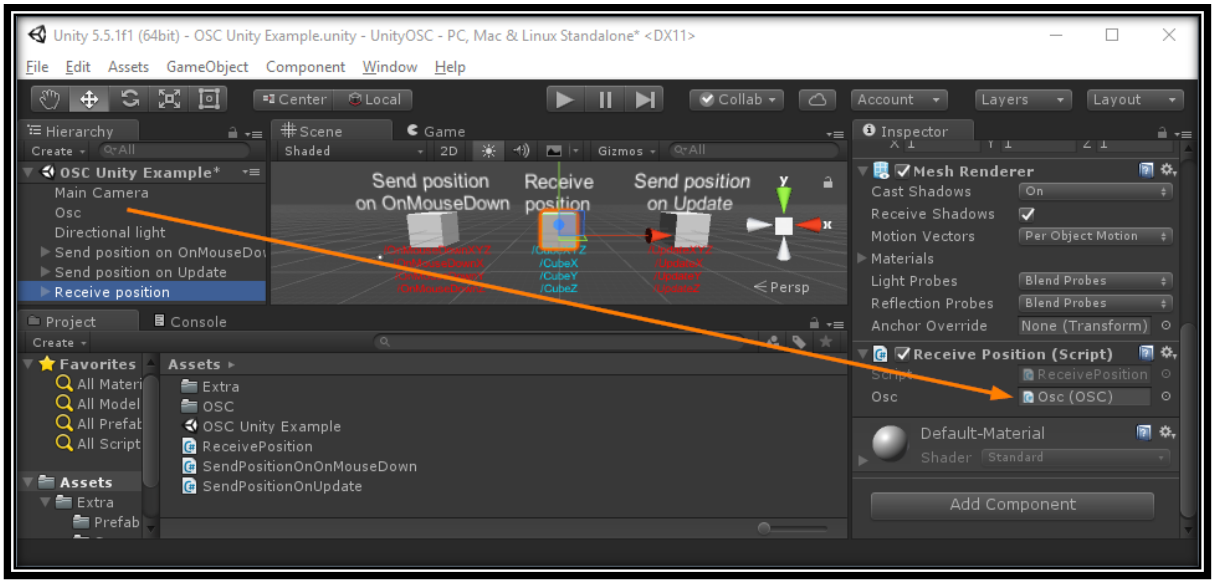


Ilustración 38:Incorporación OSC Unity.

En la siguiente imagen se presenta el resultado de la conexión. Fue el esperado ya que las dimensiones del robot están a una escala menor que las del esqueleto obtenido por el Kinect v1:

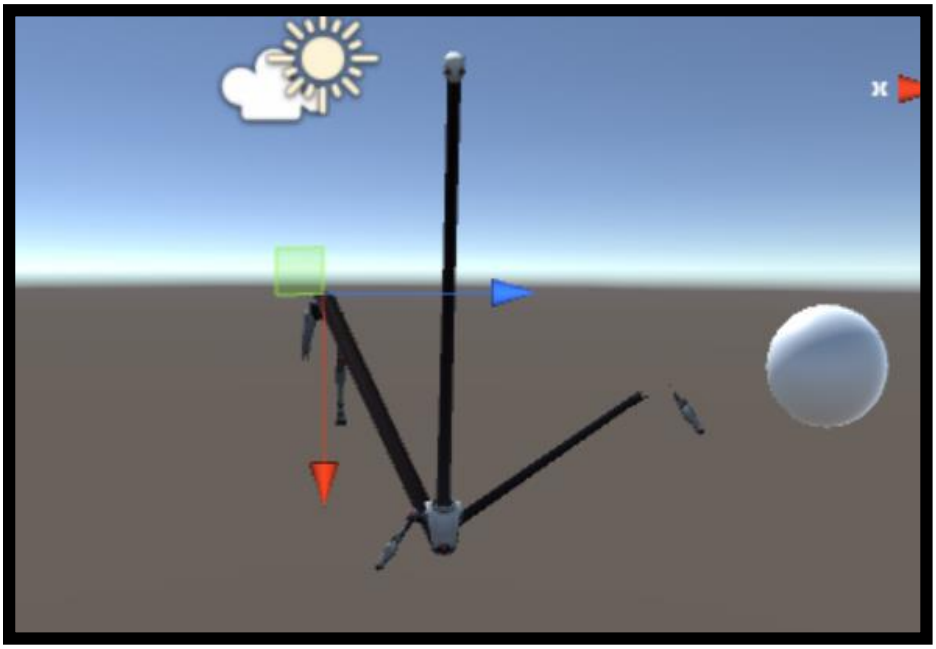


Ilustración 39:Kinect con Modelo 3D.

Parte 2: Modificación del código en Processing que recolecta los puntos de un esqueleto (de Kinect v1 gracias a la librería SimpleOpenNI) para enviar los datos de las posiciones a través del protocolo OSC a Unity

Robotica_Kinect_OSC

```
import oscP5.*;
import netP5.*;
import SimpleOpenNI.*;
//Son las direcciones en OSC, que se ocupan despues
String[] Formato={"/Head",//0
                  "/Neck",//1
                  "/L_Shoulder",//2
                  "/R_Shoulder",//3
                  "/L_Elbow",//4
                  "/R_Elbow",//5
                  "/R_Hand",//6
                  "/L_Hand",//7
                  "/Torso",//8
                  "/L_Knee",//9
                  "/R_Knee",//10
                  "/L_Hip",//11
                  "/R_Hip",//12
                  "/L_Foot",//13
                  "/R_Foot"};//14

//Se inician variables,datos es para enviar el array x,y,z
float[] datos;
PVector[] vector = new PVector[30];
OscP5 oscP5;
NetAddress remote;
OscMessage mensaje;
SimpleOpenNI  context;

void setup()
{
  size(640,480);
//Kinect
  context = new SimpleOpenNI(this);
  if(context.isInit() == false)
    {
       println("No se puede inicializar SimpleopenAI, no esta conectada la camara");
       exit();
       return;
    }
context.enableDepth();
context.enableUser();
//OSCP5
//Escucha loos mensajes del puerto 12001
 oscP5 = new OscP5(this,12010);
 //envia a la red local y al puerto 12001
 remote = new NetAddress("127.0.0.1",5000);

}
void draw()
{
  context.update();
  image(context.userImage(),0,0);
  int[] userList = context.getUsers();
    for(int i=0;i<userList.length;i++)
    {
      //Se realiza el ciclo for por cada ciclo DRAW
      if(context.isTrackingSkeleton(userList[i]))
      {
        //LLAMA A LA LA FUNCION DRAW , QUE ESTA AL INFERIOR
        drawSkeleton(userList[i]);}
    }
}
float[] source;
void drawSkeleton(int userId){
PVector jointPos = new PVector();
```

Robotica_Kinect_OSC

```
//Se obtienen las posiciones a traves de la libreria
//y se envian por medio de OSC en la funcion "enviar"
//Se utiliza el array String para definir las direcciones ejemplo "/Head"
//************************************************************
  context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_HEAD,jointPos);
 datos=jointPos.array();
 vector[0]=jointPos;
enviar(datos,Formato[0]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_NECK,jointPos);
 datos=jointPos.array();
  vector[1]=jointPos;
enviar(datos,Formato[1]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_SHOULDER,jointPos);
 datos=jointPos.array();
  vector[2]=jointPos;
enviar(datos,Formato[2]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_SHOULDER,jointPos);
 datos=jointPos.array();
  vector[3]=jointPos;
enviar(datos,Formato[3]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_ELBOW,jointPos);
 datos=jointPos.array();
  vector[4]=jointPos;
enviar(datos,Formato[4]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_ELBOW,jointPos);
datos=jointPos.array();
 vector[5]=jointPos;
enviar(datos,Formato[5]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_HAND,jointPos);
 datos=jointPos.array();
  vector[6]=jointPos;
enviar(datos,Formato[6]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_HAND,jointPos);
datos=jointPos.array();
 vector[7]=jointPos;
enviar(datos,Formato[7]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_TORSO,jointPos);
  datos=jointPos.array();
   vector[8]=jointPos;
enviar(datos,Formato[8]);
```

Robotica_Kinect_OSC

```
  //************************************************************
 context.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_KNEE,jointPos);
  datos=jointPos.array();
enviar(datos,Formato[9]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_KNEE,jointPos);
 datos=jointPos.array();
enviar(datos,Formato[10]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_HIP,jointPos);
 datos=jointPos.array();
enviar(datos,Formato[11]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_HIP,jointPos);
datos=jointPos.array();
enviar(datos,Formato[12]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_FOOT,jointPos);
  datos=jointPos.array();
enviar(datos,Formato[13]);
  //************************************************************
 context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_FOOT,jointPos);
datos=jointPos.array();
enviar(datos,Formato[14]);
  //************************************************************


}

//Envia los datos, los crea y los envia.
void enviar(float[] datos,String formato ){
  mensaje=new OscMessage(formato);
  mensaje.add(datos);
  oscP5.send(mensaje, remote);
}

void onNewUser(SimpleOpenNI curContext, int userId)
{
  println("Nuevo Usuario - userId: " + userId);
  println("\tEmpieza a Buscar el esqueleto.");
  curContext.startTrackingSkeleton(userId);
}
void oscEvent(OscMessage theOscMessage) {
  /* print the address pattern and the typetag of the received OscMessage */
  print("### received an osc message.");
  print(" addrpattern: "+theOscMessage.addrPattern());
  println(" typetag: "+theOscMessage.typetag());
}
```

Parte 3: Creación de esqueleto con huesos y superficie en el programa Blender

Como Robot Kyle tiene muchos huesos y es un sistema complejo para el movimiento en tiempo real de sus partes, se procede a crear un nuevo personaje con el programa Blender.

El primer paso es crear la conexión entre los nodos y los huesos del esqueleto que se quiere crear (huesos madres e hijos) y después alrededor del esqueleto crear capas superficiales para dar forma al personaje en Unity:

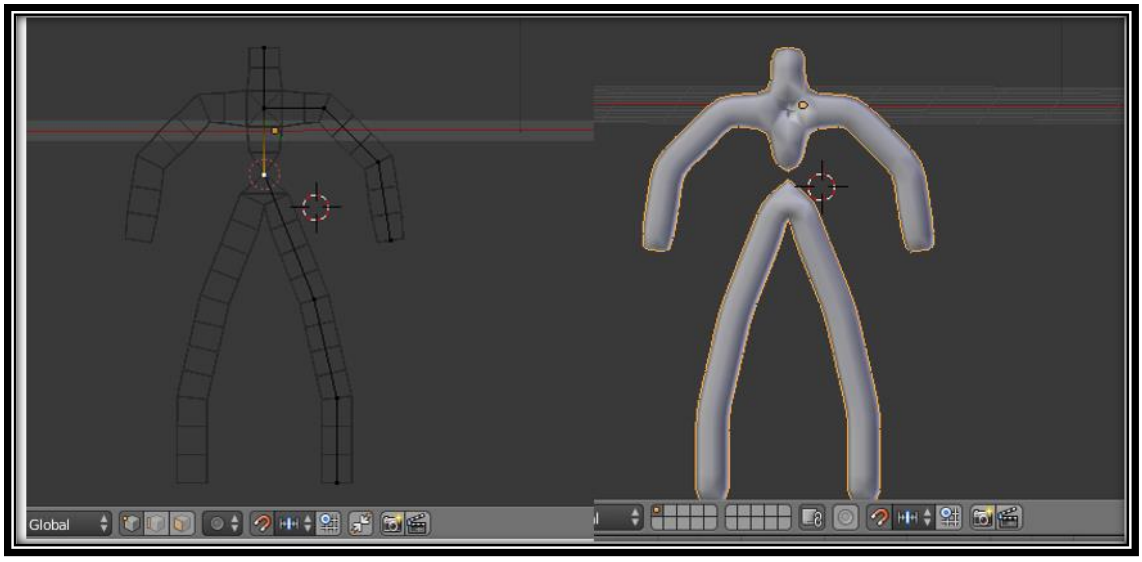


Ilustración 40:Esqueleto Blender.

Finalmente se modifican las dimensiones de los huesos a largos definidos además de cambiar de color y tamaño la superficie alrededor del esqueleto:

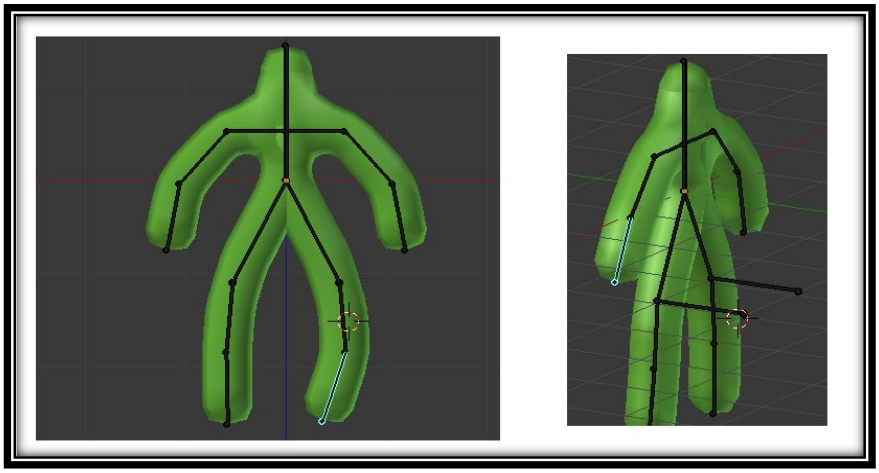


Ilustración 41:Modelo Mejorado Blender.

Se importa el esqueleto a Unity y se vuelven a realizar los mismos pasos que anteriormente se le aplicaron al Robot Kyle para establecer la comunicación OSC:

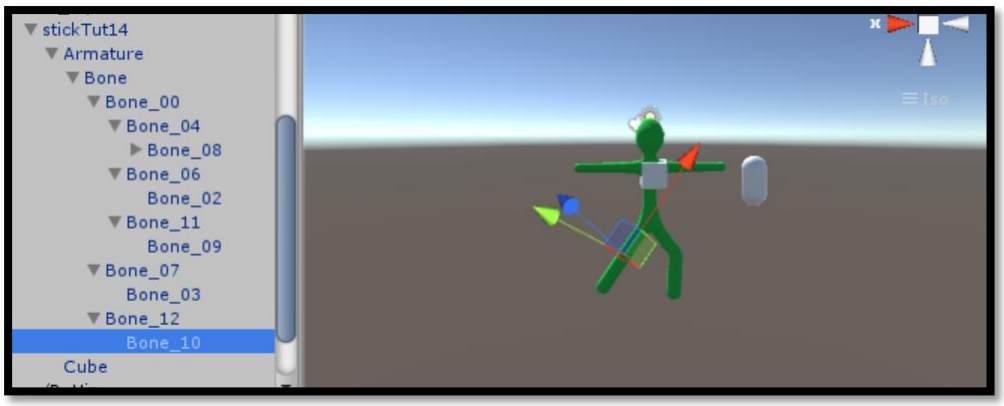


Ilustración 42:Modelo Blender Exportado a Unity.

El resultado de esta simulación fue mucho mejor que con el robot Kyle ya que las dimensiones de los huesos están escaladas con la del esqueleto del Kinect v1. Sin embargo el movimiento del personaje fue erróneo, no natural al del ser humano ya que solo cambiamos las posiciones de los nodos y no las rotaciones de los huesos en coordenadas euclidianas. Para modificar las rotaciones de los huesos debemos obtener esos datos de Kinect v1 por medio de la librería SimpleOpenNI. Esta nos da las matrices de rotación en cada hueso:

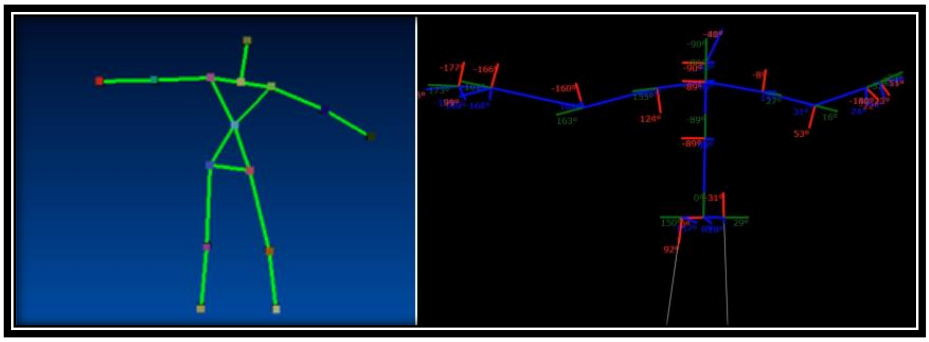
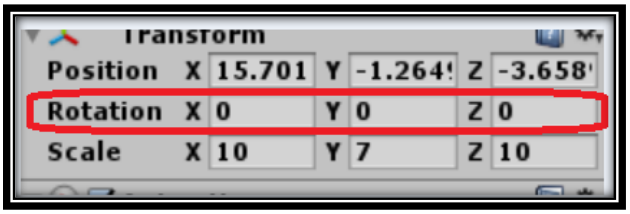


Ilustración 43:Rotación de Huesos.

Estos datos de la matriz de rotación se deben enviar a través del protocolo OSC a Unity y modificar la rotación al igual como se modifica la posición anteriormente:



Los datos recolectados del Kinect son muy inestables y se necesitan calibrar y generar nuevos algoritmos complejos, por lo que esta opción la descartamos por el poco tiempo que se da para la realización del proyecto.

Parte 4: Creación de objetos en Unity3D Protocolo OSC (Objetos)

Para representar el movimiento del esqueleto del Kinect v1 en Unity se crean objetos esferas que seguirán el movimiento de los puntos del individuo calculados por la librería SimpleOpenNI en Processing. Esto nos da la posibilidad de visualizar el movimiento naturalmente humano del esqueleto ya que al no tener huesos no será necesario ocupar las matrices de rotación.

Las esferas son objetos primitivos en Unity por lo que se crean fácilmente imponiendo a cada una un radio específico. Cada objeto esfera puede cambiar su posición al igual como en el caso de los nodos de los huesos del robot Kyle. Para ello importamos el script OSC.cs para la comunicación OSC y ReceivePosition.cs para recibir los mensajes de los datos del esqueleto del Kinect v1 enviados desde Processing.

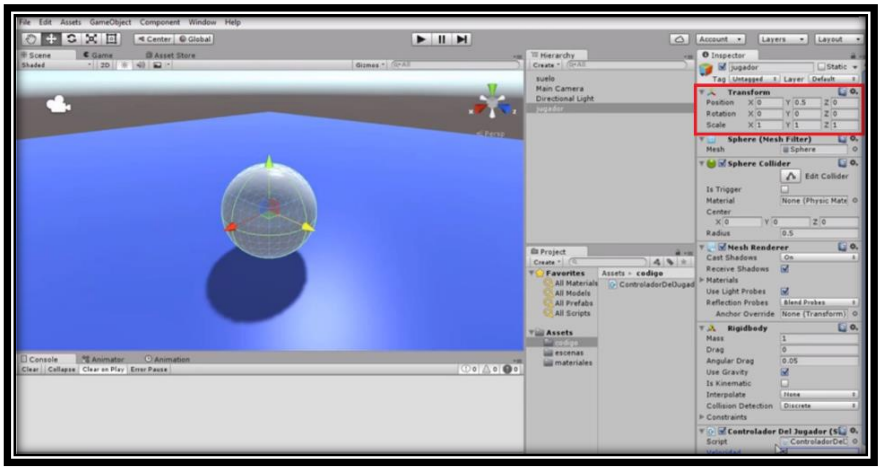


Ilustración 44:Esfera en Unity.

Los resultados son los esperados, se aprecia un correcto envió de datos por protocolo OSC entre Processing y Unity, con una rapidez en tiempo real moderada y una visualización del movimiento del individuo natural.

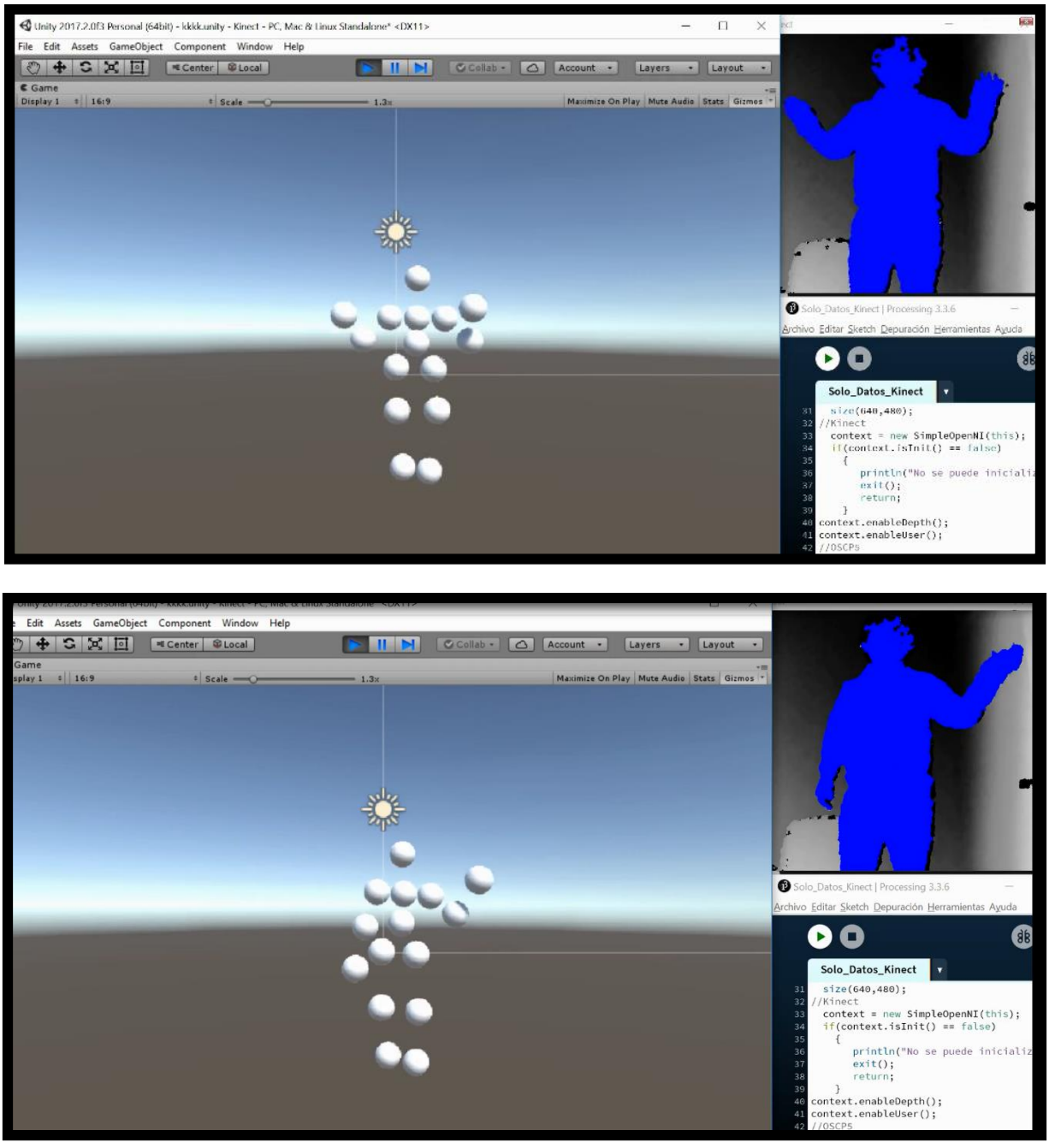


Ilustración 45:Esqueletización en tiempo real Unity.

Conclusión

La conexión del dispositivo al computador fue realizada con éxito, sin embargo existieron inconvenientes ya que los drivers de Microsoft para Kinect ahora son limitados para el uso en programas abiertos, en vista que Windows compro la marca “Kinect Xbox 360” y empezaron a vender su propio modelo “Kinect for Windows”. En el mismo plano, los drivers PrimeSensor fueron vendidos a la empresa Apple, dificultando la instalación actualmente en los ordenadores de gente común y corriente. Cabe decir que los programas instalados deben ser de la misma versión y de 32 bits para que el Kinect v1 pueda utilizarse, dificultando notoriamente la instalación.

Para procesar la información extraída del Kinect y visualizarla en animaciones 2D o 3D se necesita una tarjeta de video apropiada y con los drivers actualizados, aumentando los costos de cualquier proyecto de kinect que tenga este objetivo.

Se puede apreciar de las imágenes obtenidas, que el dispositivo Kinect funciona de forma correcta y puede graficar las imágenes de profundidad y RGB de forma correcta, sin visualizar ningún tipo de retraso o problema alguno, esto es debido a la correcta implementación de las librerías.

El proceso de esqueletización presenta una velocidad de respuesta alta, ya que el código se ejecuta de forma continua y presenta una mínima latencia (5ms).

Se puede observar, que la esqueletización se ejecuta de forma exitosa y que el modelo funciona bien, independiente de las características físicas del individuo, ropa, tamaño, entre otros. El algoritmo utilizado para la esqueletización, que se basa en “Random Forest”, funciona según lo esperado, representando una opción fiable, rápida y económica. Incluso con varios usuarios, el proceso se lleva a cabo sin mayores problemas.

El proceso de comunicación a través de los protocolos OSC y UDP, funcionan exitosamente. Aun así, falta desarrollo del código debido a que fueron probados con el mismo programa y lo ideal es que la comunicación se utilizado entre dos programas diferentes, para realizar la animación.

La comunicación a por medio del protocolo OSC en el inicio de su creación solo se usó para conectar instrumentos musicales principalmente pero en realidad para nuestro uso de desarrollo tecnología fue un éxito. Se creó un código en el programa Processing que pueda enviar datos a cualquier puerto donde puedan ser leídos por cualquier dispositivo y aplicaciones. Este programa sirve para cualquier dispositivo que se pueda leer datos a través del lenguaje de programación en Processing, por lo que este protocolo es muy versátil para todo tipo de problemas informáticos de transmisión de datos.

La plataforma del programa Unity es una gran solución para representar el movimiento de un esqueleto extraído de Kinect, ya que se pueden crear personajes u objetos. Estos personajes u objetos se pueden mover en un escenario al igual que una persona camina en una habitación. Es posible ya que cada objeto o hueso de la persona tiene coordenadas XYZ, de escala y de rotación, creando un movimiento continuo y real del esqueleto humano captado por Kinect. Unity tiene funciones que hacen posible la lectura de datos por medio de la comunicación OSC de cualquier puerto. La conexión entre Unity y Processing por medio del protocolo OSC fue un éxito ya que la velocidad de respuesta y envió de datos de las coordenadas de los esqueletos en tiempo real es casi simultánea, con un rango de error aceptable. Como recomendación para personajes con más huesos y restricciones de movimiento se recomienda obtener de Kinect las matrices de rotación de cada parte del esqueleto.

Links de Interés

- Kinect para Windows SDK v1.8 : https://www.microsoft.com
- DriverStoreExplorer: https://archive.codeplex.com/?p=driverstoreexplorer
- Página oficial OSC: http://opensoundcontrol.org/introduction-osc
- Código de OSC ejemplo: https://github.com/truthlabs/unity-osc-processing/blob/master/Hello%20World/OSC_Processing/OSC_Processing.pde
- Ejemplo aplicaciones OSC: http://www.sojamo.de/libraries/oscP5/#examples
- Live Grabber: https://sonicbloom.net/en/livegrabber-to-sendreceive-osc-in-ableton-live/
- Ableton: https://www.ableton.com
- Códigos de informáticos en Processing: https://github.com/
- Paquete para Kinect v1 y esqueletizacion en Matlab: https://www.mathworks.com/help/imaq/acquiring-image-and-skeletal-data-using-the-kinect.html

Bibliografía

Borestein, G. (2012). *Making Things See.* San Francisco.

Microsoft Research Cambridge & Xbox Incubation. (2011). *Real-Time Human Pose Recognition in Parts from Single Depth Images.* New York: Microsoft.

Rheiner, M. (24 de Abril de 2018). *Github*. Obtenido de https://github.com/wexstorm/simple-openni

Rouse, M. (24 de Abril de 2018). *Search networking Tech Target*. Obtenido de https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol

WRIGHT, M. (1998). *Open Sound Control: an enabling technology.* California: Standford.