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

Proyecto Robótica
Informe Avance Proyecto Nº4

Profesor: Michael Miranda.

Alumnos: Iván Fernández

Claudio Canales D.

Índice

Objetivo

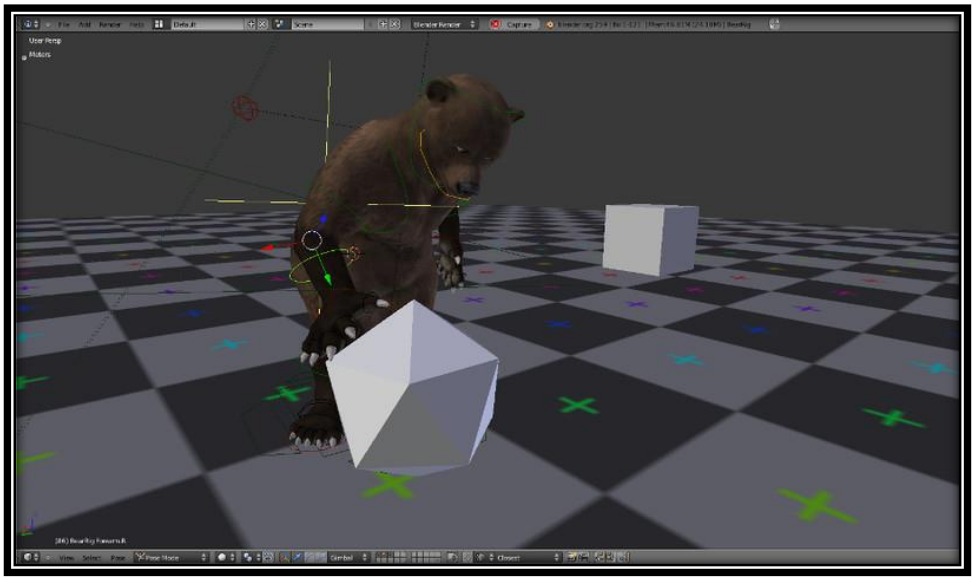
- Aprender a utilizar un programa que permita representar el esqueleto obtenido de Kinect v1 como un personaje de realidad virtual o de video juego en un escenario.

Metas específicas

- Comprender la plataforma del programa Unity3D
- Analizar el comportamiento móvil de un personaje en Unity
- Crear un personaje de videojuego con esqueleto y relaciones entre los huesos en el programa Blender
- Conectar el esqueleto de un personaje en Unity con el de un humano real extraído de Kinect v1 a través de Processing y el protocolo OSC

Procedimiento

- 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.



Introducción

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.

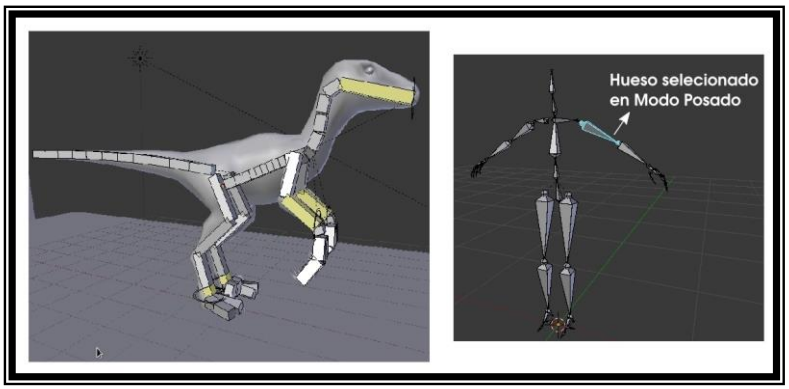
Este motor de Unity incluye además un editor de terrenos, donde se puede esculpir la forma del terreno usando las herramientas visuales que ofrece Unity, se puede pintar, texturizar, añadir hierba, colocar árboles o similares, o inclusive se permite la importación de otros materiales provenientes de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

Blender

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

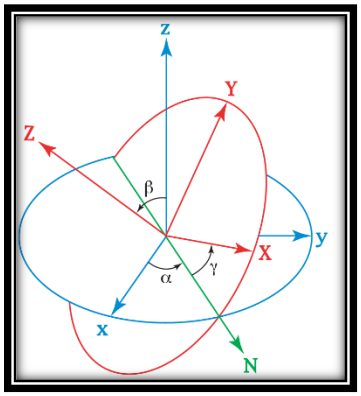
Entre sus características puede:

- 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



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.

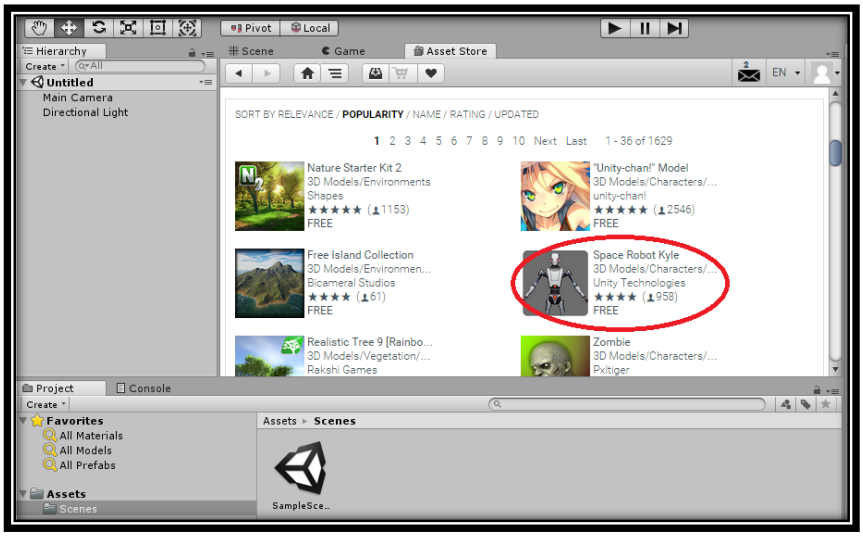


Desarrollo

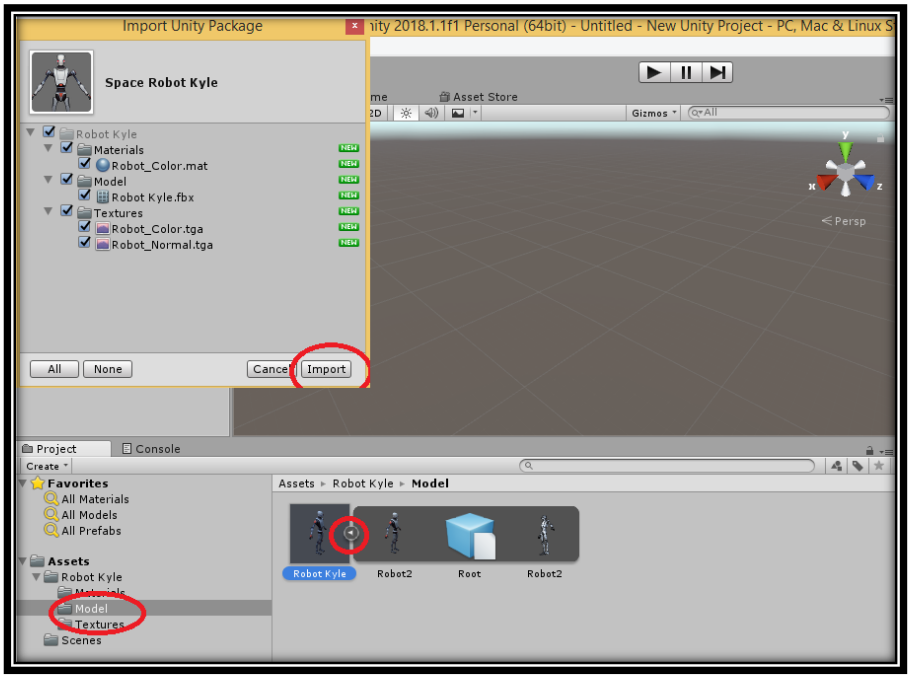
Parte 1: Transformada de posiciones en Unity y Comunicación OSC para recibir mensajes

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.

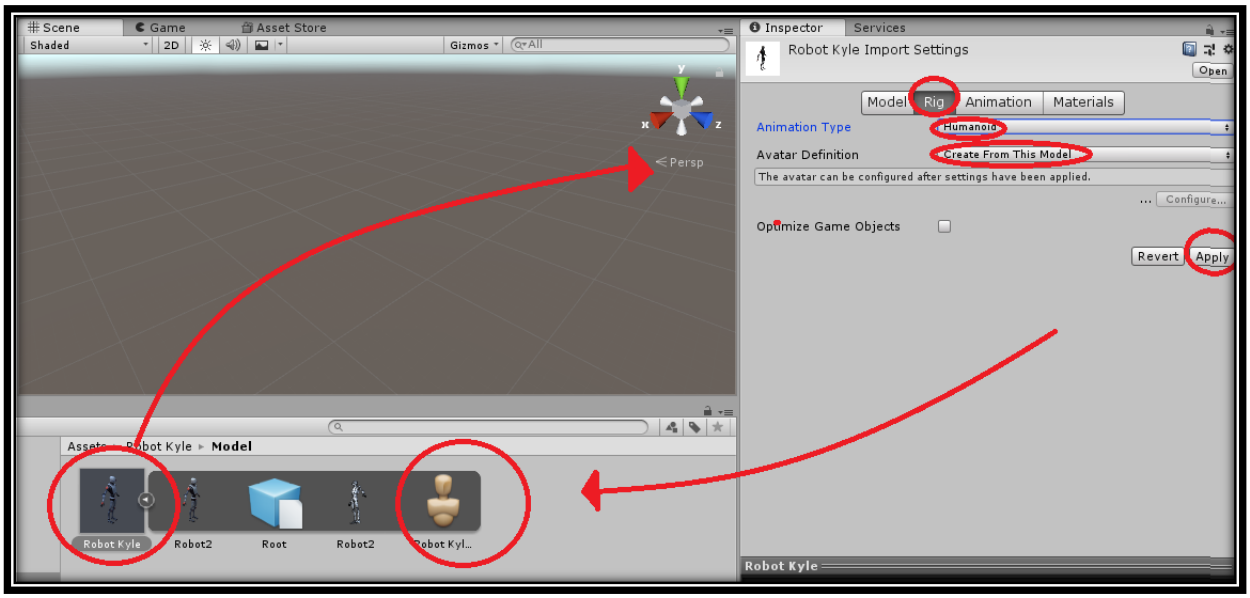
Para descargarlo se debe abrir el programa Unity y hacer clic en la pestaña *Asset Store:*



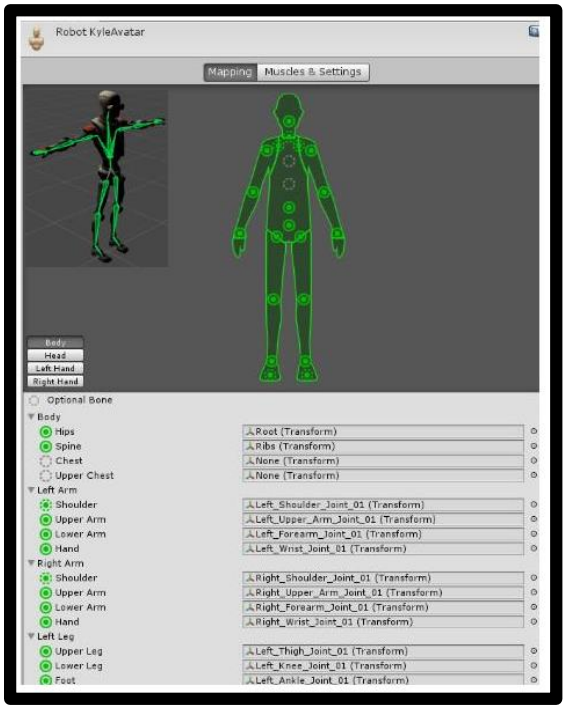
Una vez descargado el robot se importan los archivos en el programa:



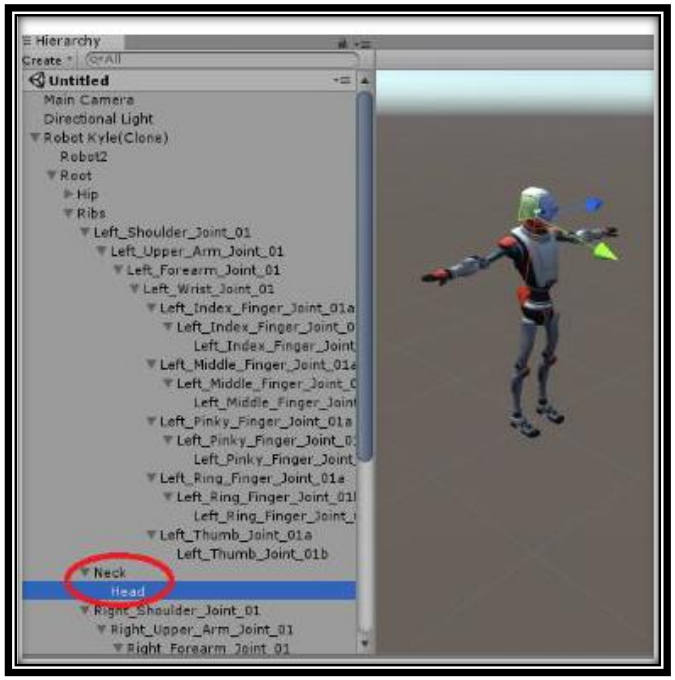
Para obtener un control del robot se debe crear una animación Humanoide de él, es decir, además de las capas superficiales, se crean huesos y nodos para producir el movimiento del personaje. Estos nodos serán conectados posteriormente con las posiciones del esqueleto de una persona capturada por el Kinect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



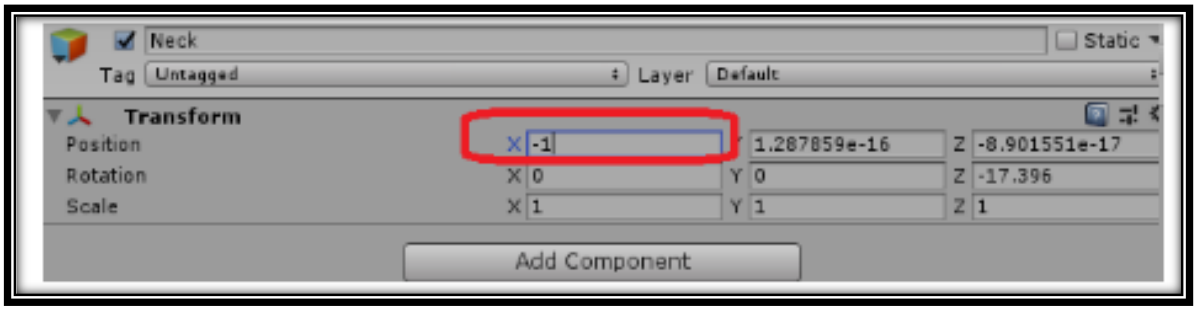
Representación grafica de nodos y huesos del robot:



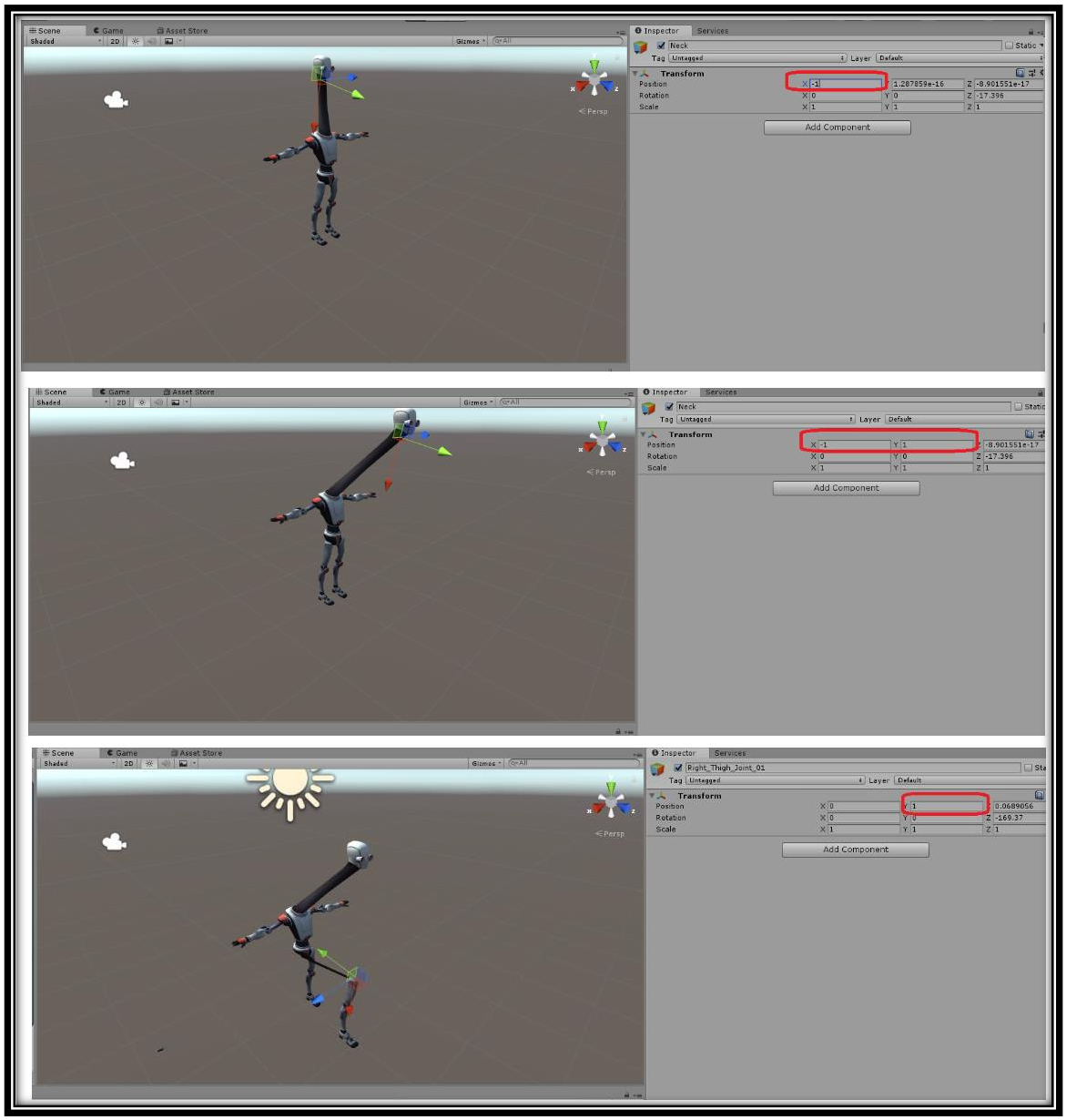
Se aprecia en la parte izquierda del programa un árbol de huesos del robot, con el nombre del hueso y su nivel jerarquía en el movimiento. Por ejemplo en este caso en el estómago está el elemento con más jerarquía, por ende si se cambian las coordenadas de este punto, todas las demás también lo hacen.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



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.



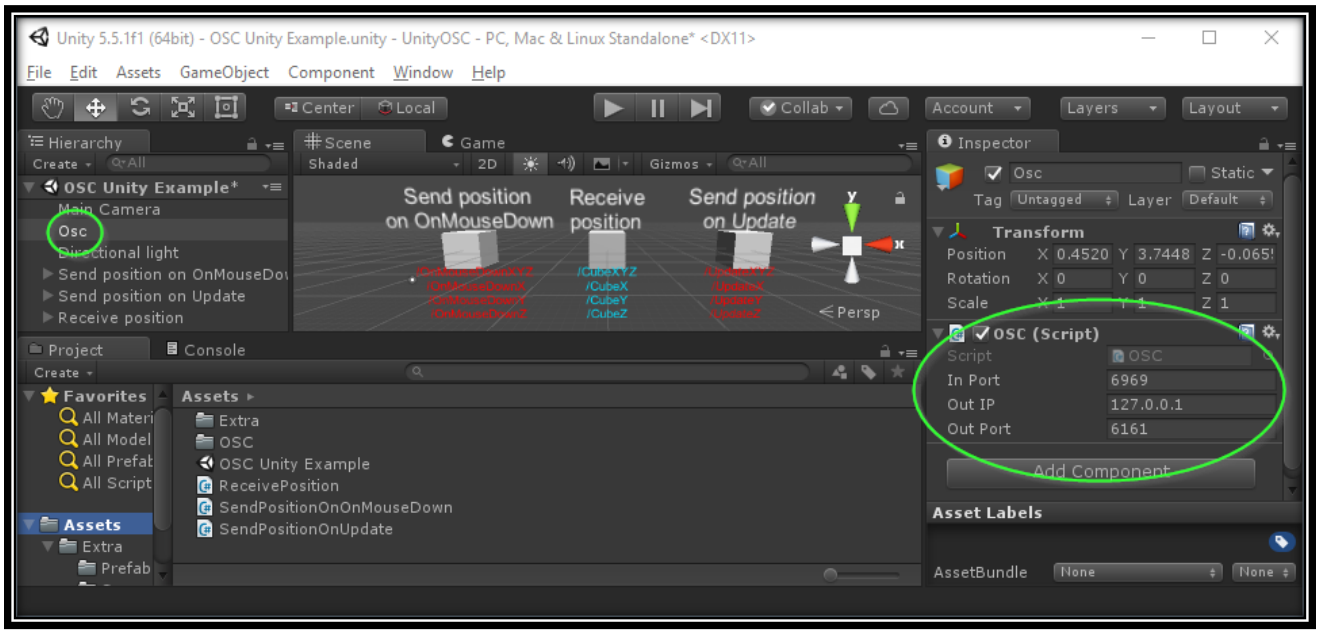
Para entender mejor el funcionamiento del robot en la plataforma se cambian manualmente los datos de las posiciones de los huesos. Por ejemplo para la cabeza del robot (Head) y la rodilla derecha (Neck) se modificaron las posiciones entregando el siguiente resultado:



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.



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*

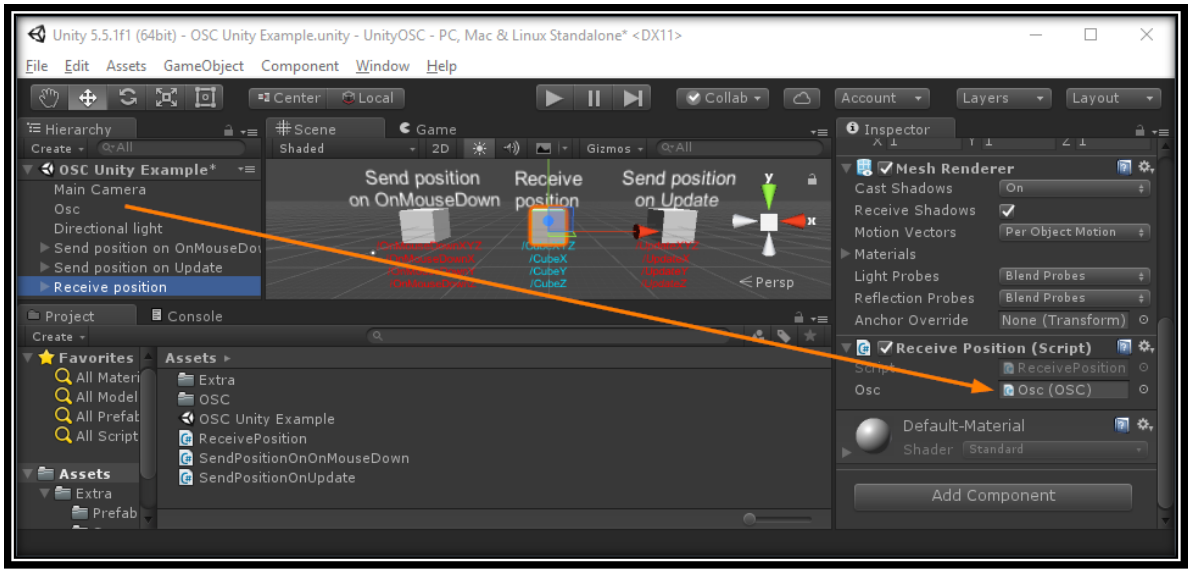
ReceivePosition.cs

```
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
```

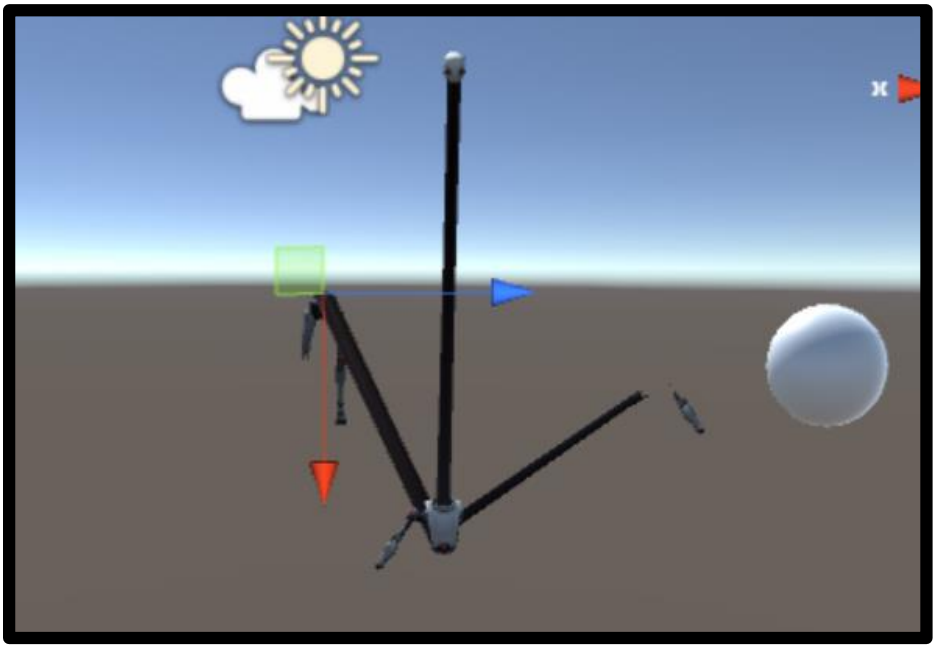
En este ejemplo (ReceivePosition.cs), el script manejará mensajes OSC con las siguientes direcciones (Posición de los puntos del esqueleto de Kinect v1):

“/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:



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:



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

```
67  //Se obtienen las posiciones a traves de la libreria
68  //y se envian por medio de OSC en la funcion "enviar"
69  //Se utiliza el array String para definir las direcciones ejemplo "/Head"
70  //*************************************************************
71    context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_HEAD,jointPos);
72   datos=jointPos.array();
73   vector[0]=jointPos;
74  enviar(datos,Formato[0]);
75    //*************************************************************
76   context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_NECK,jointPos);
77   datos=jointPos.array();
78    vector[1]=jointPos;
79  enviar(datos,Formato[1]);
80    //*************************************************************
81   context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_SHOULDER,jointPos);
82   datos=jointPos.array();
83    vector[2]=jointPos;
84  enviar(datos,Formato[2]);
85    //*************************************************************
86   context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_SHOULDER,jointPos);
87   datos=jointPos.array();
88    vector[3]=jointPos;
89  enviar(datos,Formato[3]);
90    //*************************************************************
91   context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_ELBOW,jointPos);
92   datos=jointPos.array();
93    vector[4]=jointPos;
94  enviar(datos,Formato[4]);
95    //*************************************************************
96   context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_ELBOW,jointPos);
97  datos=jointPos.array();
98   vector[5]=jointPos;
99  enviar(datos,Formato[5]);
100   //*************************************************************
101  context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_RIGHT_HAND,jointPos);
102  datos=jointPos.array();
103   vector[6]=jointPos;
104 enviar(datos,Formato[6]);
105   //*************************************************************
106  context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_LEFT_HAND,jointPos);
107 datos=jointPos.array();
108  vector[7]=jointPos;
109 enviar(datos,Formato[7]);
110   //*************************************************************
111  context.getJointPositionSkeleton(userId,SimpleOpenNI.SKEL_TORSO,jointPos);
112   datos=jointPos.array();
113    vector[8]=jointPos;
114 enviar(datos,Formato[8]);
```

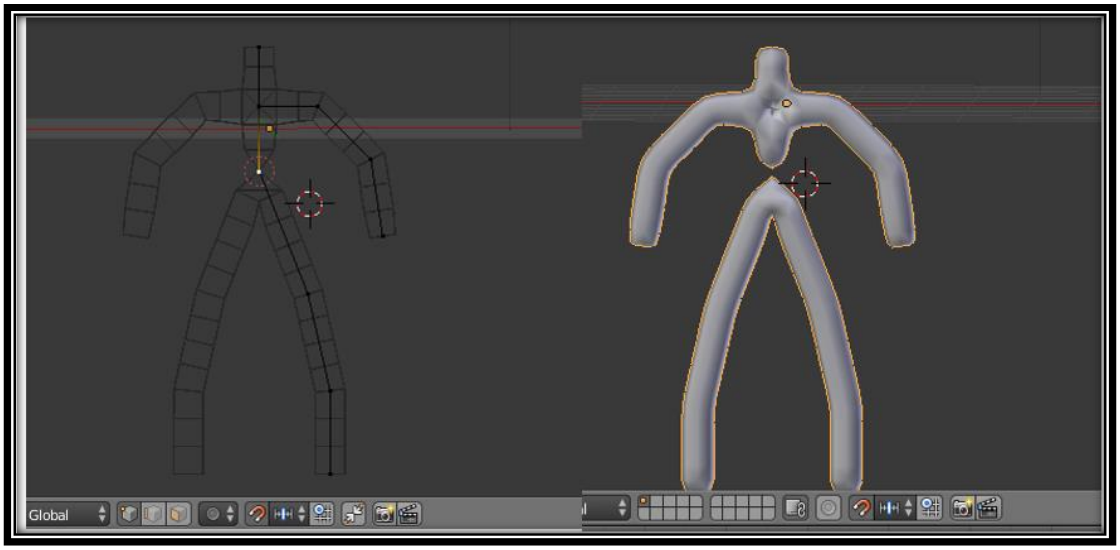
Robotica_Kinect_OSC

```
  //************************************************************
 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());
}
```

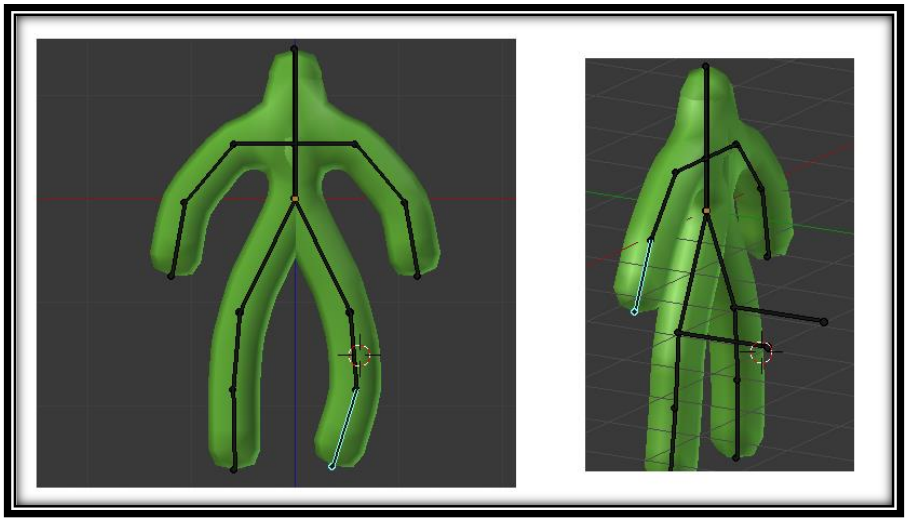
Parte 3: Creación de esqueleto con huesos y superficie en el programa Blender

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.

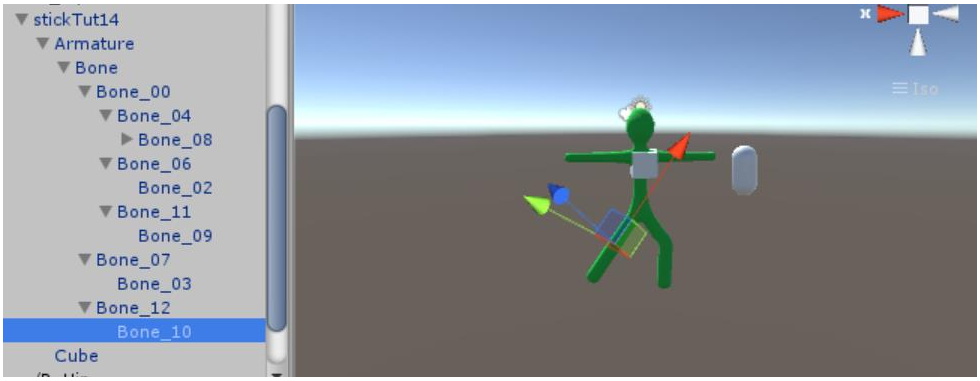
El primer paso es crear la conexión entre los nodos y los huesos del esqueleto que se quiere crear (huesos madres e hijos) y después alrededor del esqueleto crear capas superficiales para dar forma al personaje en Unity:



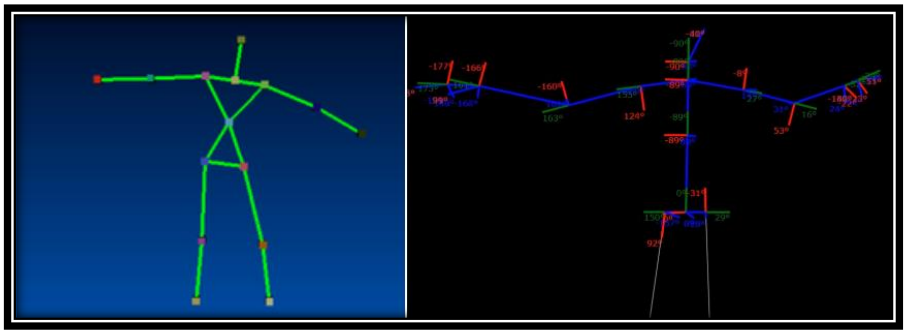
Finalmente se modifican las dimensiones de los huesos a largos definidos además de cambiar de color y tamaño la superficie alrededor del esqueleto:



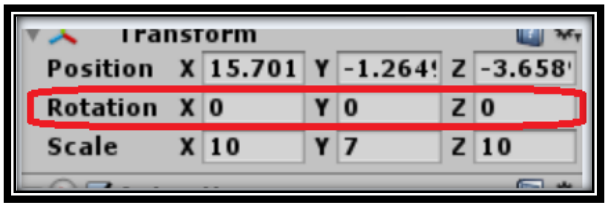
Se importa el esqueleto a Unity y se vuelven a realizar los mismos pasos que anteriormente se le aplicaron al Robot Kyle para establecer la comunicación OSC:



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:



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:

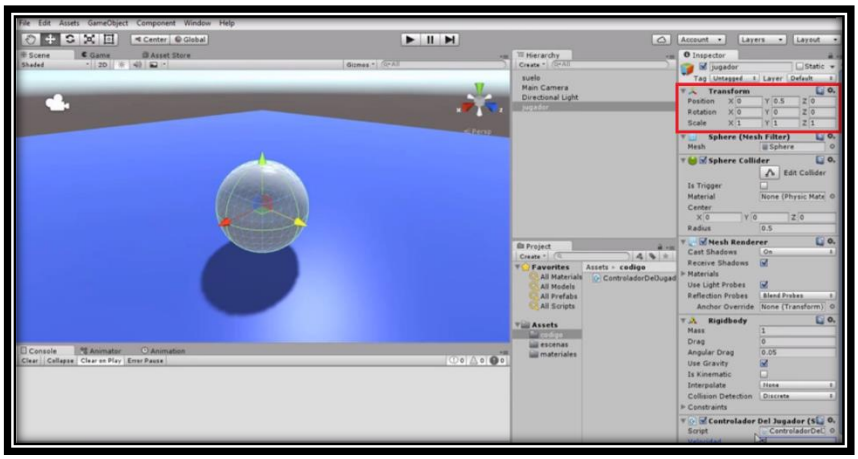


Los datos recolectados del Kinect son muy inestables y se necesitan calibrar y generar nuevos algoritmos complejos, por lo que esta opción la descartamos por el poco tiempo que se da para la realización del proyecto.

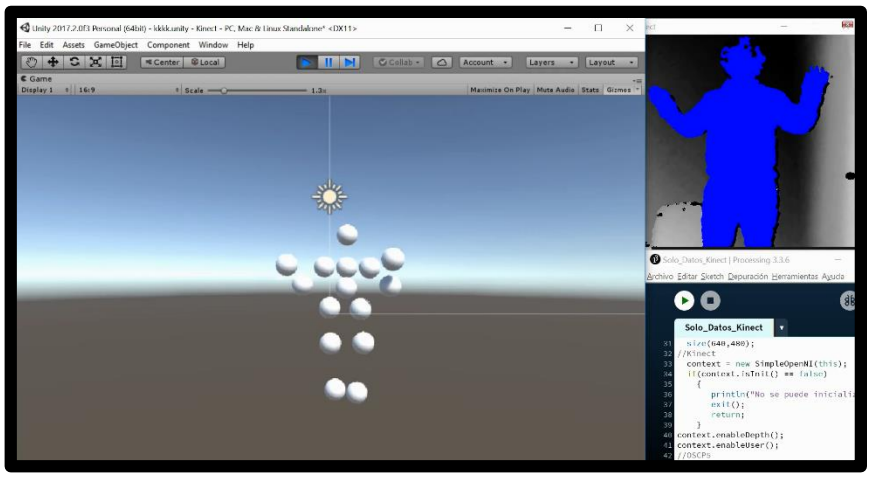
Parte 4: Creación de objetos en Unity3D Protocolo OSC (Objetos)

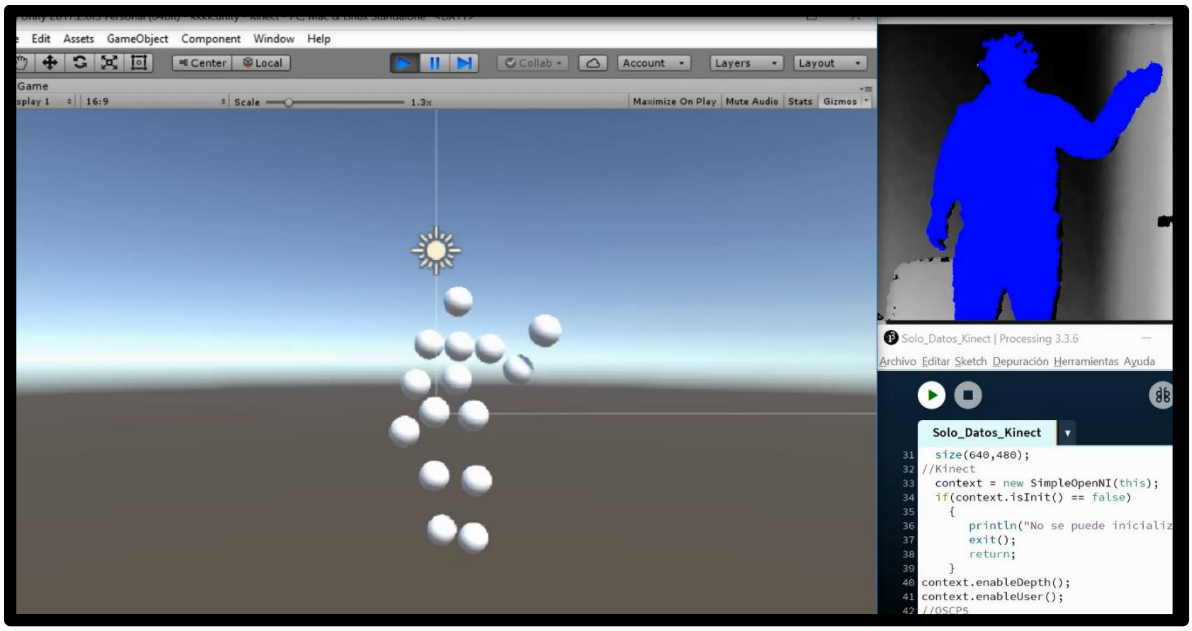
Para representar el movimiento del esqueleto del Kinect v1 en Unity se crean objetos esferas que seguirán el movimiento de los puntos del individuo calculados por la librería SimpleOpenNI en Processing. Esto nos da la posibilidad de visualizar el movimiento naturalmente humano del esqueleto ya que al 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.



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.




Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Game
Scale 1.3x
Maximize On Play Mute Audio Stats Gizmos
Solo_Datos_Kinect | Processing 3.3.6
Archivo Editar Sketch Depuración Herramientas Ayuda
Solo_Datos_Kinect
size(640,480);
//Kinect
context = new SimpleOpenNI(this);
if(context.isInit() == false)
{
println("No se puede inicializ
exit();
return;
}
context.enableDepth();
context.enableUser();

Conclusión

Biografía