PROYECTO FINAL DE CARRERA

Guiado gestual de un robot humanoide mediante un sensor Kinect.

Estudios: Ingeniería técnica en informática de sistemas.

Autor: Sammy Pfeiffer

Director: Joan Aranda López

Año: 2011

Índice de contenido

1 Introducción

1.1 Kinect

Ilustración 1: Sensor Kinect

Recientemente ha salido al mercado un interesante sensor, comercialmente llamado Kinect.Parafraseando a la Wikipedia encontramos que Kinect se puede definir como: “un controlador de juego libre y entretenimiento desarrollado por Microsoft para la videoconsola Xbox 360. Kinect permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos, comandos de voz, y objetos e imágenes “.

Microsoft Research invirtió veinte años de desarrollo en la tecnología de Kinect. Fue anunciado por primera vez el 1 de junio de 2009 en la Electronic Entertainment Expo 2009 como "Project Natal".

El sensor de Kinect es una barra horizontal de aproximadamente 23 cm conectada a una pequeña base circular con un eje de articulación de rótula, y está diseñado para ser colocado longitudinalmente por encima o por debajo del televisor.

Kinect se compone principalmente de:

-Una cámara tradicional (Resolución 640x480 RGB 30fps VGA).
-Un emisor de infrarrojos.
-Una cámara de infrarrojos.
-4 Micrófonos (16bit sampling rate: 16Hz).
-Un motor.

Ilustración 2: Kinect desmontado

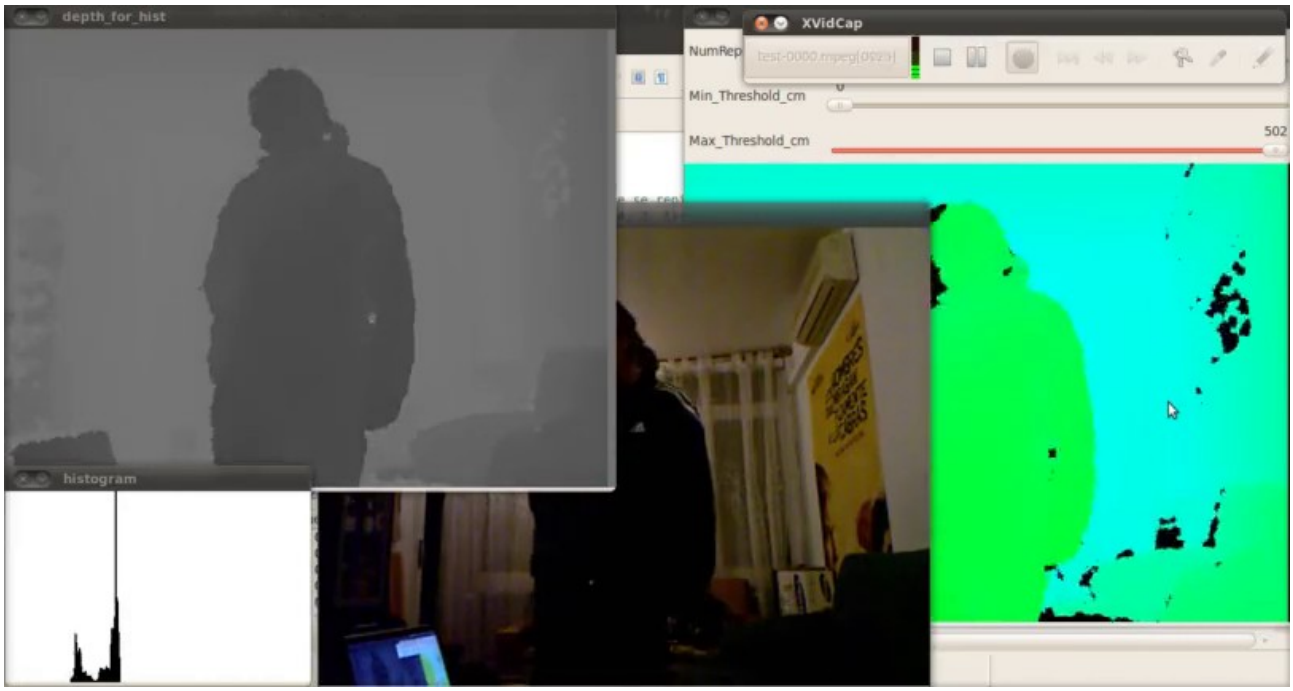


Ilustración 3: Imagen de profundidad en escala de grises, imagen webcam, profundidad en escala de colores

1.2 ¿Cómo funciona Kinect?

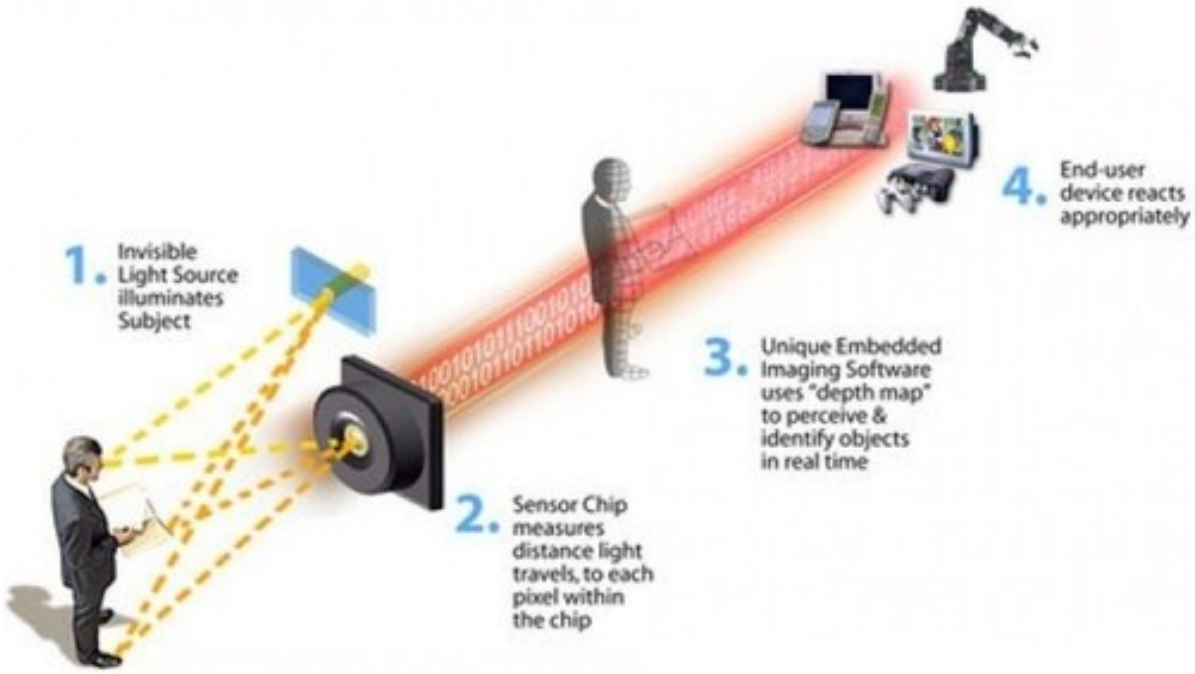


Ilustración 4: Esquema funcionamiento

Para conocer la distancia a la que se encuentra cada píxel de la imagen de profundidad se emite un a constelación de puntos con el emisor infrarrojo:

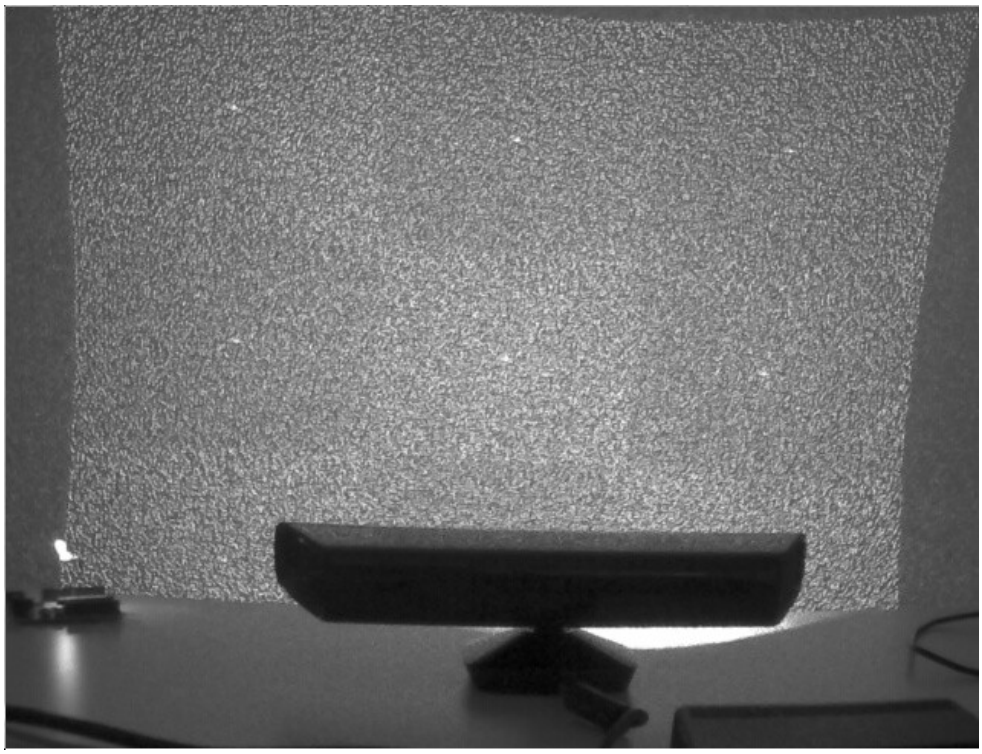


Ilustración 5: Constelación infrarroja Kinect

Entonces la cámara infrarroja detecta esta constelación y Kinect calcula la disparidad para cada píxel (la diferencia entre donde estaba el punto al proyectarlo a donde está en la proyección). A esto se le llama cámara de luz estructurada.

1.3 Utilidad directa del sensor: Skeleton tracking.

Ilustración 6: Ejemplo de Skeleton Tracking

Skeleton Tracking, o lo que es lo mismo, hacer seguimiento del cuerpo (esqueleto) de un usuario en tiempo real.
La situación actual es que Microsoft ha anunciado la salida de un SDK propio para usar el Kinect incluyendo Skeleton Tracking. Con lo cual ha surgido un movimiento para crear drivers y aplicaciones para Kinect mayormente opensource.

A partir de lo cual se ha creado OpenNI, una organización para promover la compatibilidad e interoperabilidad de dispositivos, aplicaciones y middleware de interacción natural (Natural Interaction).

En OpenNI han creado una librería para realizar Skeleton Tracking incluida en un paquete llamado NITE. Esta librería nos brinda seguimiento del cuerpo del usuario aportando las coordenadas en el espacio de cada uno de los puntos de interés (extremidades y articulaciones).

1.4 Robot humanoide: Bioloid

Un robot humanoide es robot antropomorfo que, además de imitar la apariencia humana, imita algunos aspectos de su conducta.

Ilustración 7: Bioloid en forma humanoide

Para este proyecto se ha usado el kit comercial Bioloid Premium Kit, de Robotis. Este robot está diseñado para el aprendizaje y hobby de la robótica. Está formado por varios sensores, una batería, el controlador principal llamado CM-510, los componentes para unir las diferentes piezas y servomecanismos modulares llamados Dynamixels. Estos últimos pueden usarse conectados en Daisy-chain para construir robots de diferentes formas (con ruedas, piernas, humanoides...).

Este kit se programa en un lenguaje C propio, como es costumbre en otros paquetes similares destinados a la robótica como hobby.

Existen otros kits como por ejemplo:

-Lego Mindstorms.
-iRobot Create.
-Vex Robotics Design System.
-KHR-1

En este proyecto se ha usado el Bioloid Premium Kit dado que satisface nuestras necesidades y además existía uno a nuestra disponibilidad.

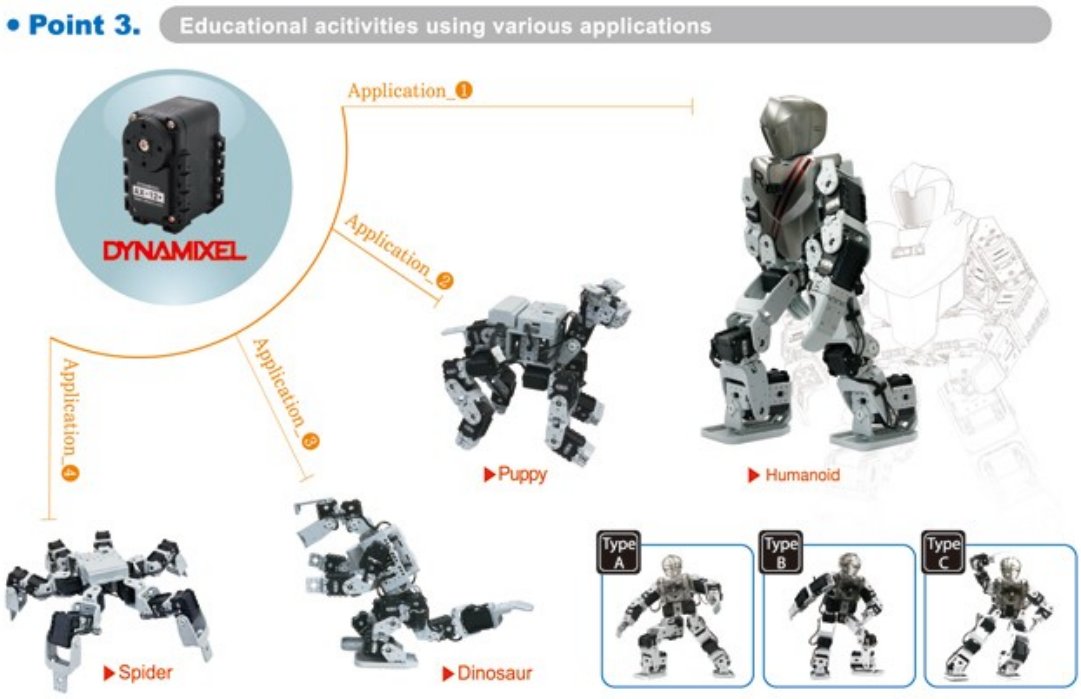


Ilustración 8: Diferentes configuraciones del kit Bioloid

1.4.1 Servomecanismos: Dynamixels

Ilustración 9: Dynamixel

Ilustración 10: Adaptador USB - Dynamixel / Serie

Los Dynamixel tienen una similitud a los tradicionales servomotores, ambos son motores basados en posición, se le puede indicar un ángulo al que girar y se posicionará en ese lugar. Pero no acaba ahí, se puede jugar con unas 50 variables distintas en relación a su movimiento: máximo ángulo de movimiento, velocidad de movimiento, fuerza, carga actual, temperatura, etc.
Estos motores pueden ser manipulados directamente gracias al adaptador suministrado con el Kit estando conectados por Daisy-chain unos con otros, con lo cual se forma una pequeña red por la que se pueden mandar mensajes para ejecutar movimientos, leer variables de los distintos motores, etc.
También se puede conectar por serie al controlador CM-510 y este controlador se ocupará de mandar estos mensajes.

2 Objetivos

El objetivo principal consiste en controlar los brazos de un robot humanoide mediante los datos proporcionados por un sensor Kinect. Este objetivo se compone de los siguientes subobjetivos o tareas:

-Estudio de las características de los actuadores: será necesario saber como y que son capaces de hacer.

-Implementación de un firmware de control de los actuadores del robot.

-Implementación de la cinemática inversa de los brazos: se tendrá que poder dar un punto en el espacio donde se quiera posicionar la mano del robot y que este posicione las articulaciones donde sea necesario.

-Estudio de la resolución y precisión de los datos ofrecidos por Kinect.

-Incorporación de la librería de Skeleton Tracking: será necesario conocer como funciona y que datos ofrece sobre el esqueleto para trabajar con él.

-Simulador para evaluación de los datos del esqueleto: será necesario representar los datos del esqueleto para poder controlar su precisión.

-Simulador para la simulación de los movimientos del robot: será necesario simular primero los movimientos para no poner en peligro el robot y comprobar que los datos de los movimientos son correctos.

-Unión de los simuladores en una aplicación final que cumpla el objetivo principal.

-Probar y evaluar la aplicación final de integración de los simuladores. Para ello se diseñarán las pruebas necesarias una vez logrado el anterior subobjetivo.

3 Trabajos previos

3.1 OSCeleton, skeleton tracking.

OSCeleton es un pequeño programa que hace uso de los datos de esqueleto proporcionados por el OpenNI framework para enviar por protocolo OSC estas coordenadas.

3.1.1 Protocolo OSC

El protocolo OSC (Open Sound Control) es un protocolo pensado para comunicar instrumentos musicales, dispositivos multimedia y ordenadores en tiempo real. Comúnmente es transportado por UDP.

Las características de este protocolo son:

-Ampliable, dinámico. Esquema de nombres simbólicos tipo URL.
-Datos numéricos simbólicos y de alta resolución
-Lenguaje de coincidencia de patrones (pattern matching) para especificar múltiples receptores de un único mensaje.
-Marcas de tiempo (time tags) de alta resolución.
-Mensajes “empaquetados” para aquellos eventos que deben ocurrir simultáneamente
-Sistema de interrogación para encontrar dinámicamente las capacidades de un servidor OSC y obtener documentación.

Un ejemplo de un paquete OSC de OSCeleton podría ser:

Address pattern: "/joint"

Type tag: "sifff"

s: Joint name, check out the full list of joints below.

i: The ID of the user.

f: X coordinate of joint in interval [0.0, 1.0]

f: Y coordinate of joint in interval [0.0, 1.0]

f: Z coordinate of joint in interval [0.0, 7.0]

3.1.2 OpenNI framework

OpenNI (Open Natural Interaction) es un framework multilenguaje, multiplataforma, que define APIs para escribir aplicaciones que utilicen “interacción natural”.

La interacción natural se refiere al concepto donde la interacción entre humano y dispositivo esta basada en los sentidos humanos, normalmente oído y vista. Con esta premisa el framework incluye:

-Reconocimiento de voz y comandos.
-Gestos con las manos, gestos predeterminados para realizar acciones configurables.
-Body Motion Tracking, donde todo el cuerpo es analizado e interpretado para conseguir un tracking.

3.2 Características Kinect

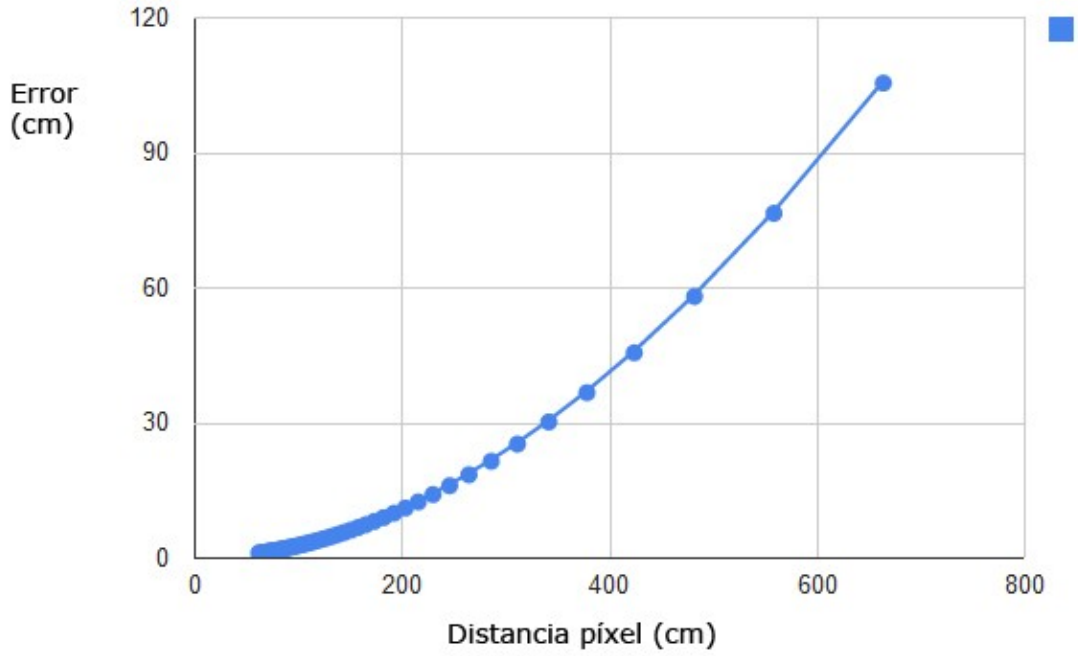


Ilustración 11: Error en centímetros de un píxel en función de la distancia de este

Las pruebas realizadas dieron lugar a esta gráfica donde podemos ver la precisión de los datos otorgados por el Kinect. La información de un píxel concreto será cada vez más lejana a la realidad según nos alejemos del sensor. Nos encontramos que el Kinect opera correctamente entre los 60cm y los 5 metros.

En el caso del Skeleton Tracking (según los desarrolladores de OpenNI que coincide con los consejos de Microsoft sobre el uso de Kinect) la distancia ideal se sitúa entre los 2 y 2,5 metros.

4 Diseño

4.1 Planificación temporal

Tareas \ Semanas	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
Estudio de las características de los actuadores.															
Implementación de un firmware del robot para el control de los actuadores.															
Implementación de la cinemática inversa de los brazos.															
Estudio de la resolución y precisión de los datos ofrecidos por Kinect.															
Incorporación de la librería de Skeleton Tracking.															
Simulador para evaluación de los datos del esqueleto.															
Simulador para simulación de los movimientos del robot.															
Unión de simuladores.															
Evaluación y test.															

Hay tareas que se encadenan unas con otras ya que resulta necesario trabajar conjuntamente.

4.2 Firmware de Bioloid

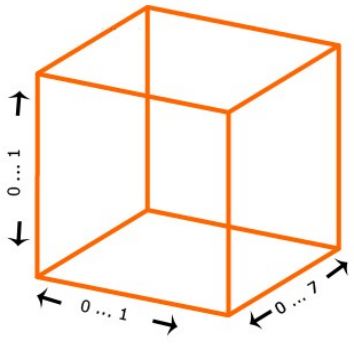
Será necesario desarrollar un firmware para comunicarse con los servomotores del Bioloid de la manera más simple y eficiente posible.

4.3 Adaptación de datos del esqueleto

Será necesario convertir los datos de las coordenadas del esqueleto de tal manera que consigamos las coordenadas de cada una de las manos en referencia al sistema de coordenadas de nuestro Bioloid.

Las coordenadas de los puntos del esqueleto nos vienen dados en una referencia al punto de vista del Kinect:

-El eje X y el eje Y en un rango de [0..1].
-El eje Z en un rango de [0..7].



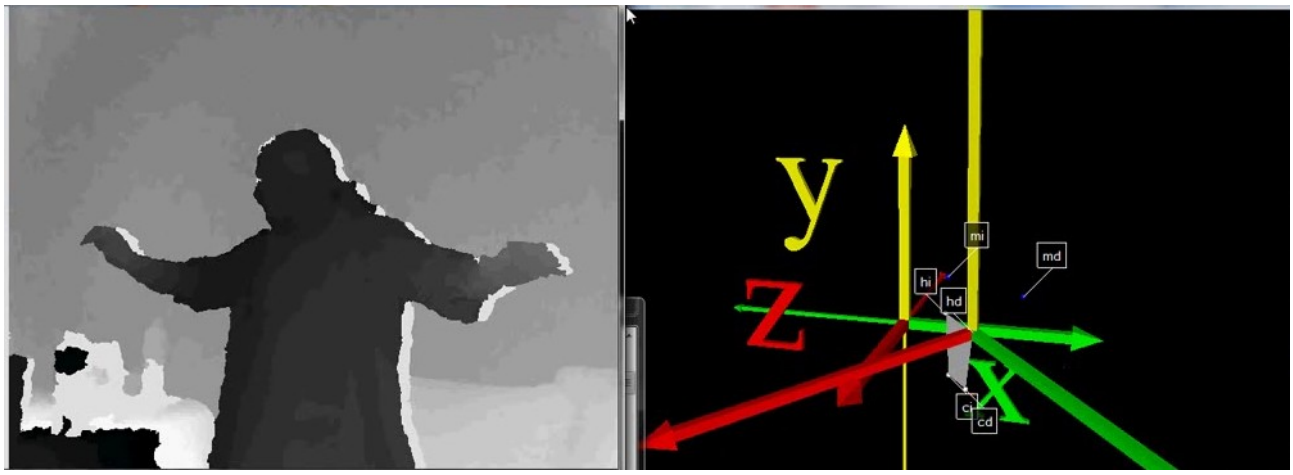


Ilustración 12: Posición relativa al usuario de sus manos

Necesitamos trasladar las coordenadas de las manos del sistema de coordenadas proporcionado por OSCeleton al sistema de coordenadas en referencia al usuario. El plano formado por el pecho del usuario debe tener el origen de coordenadas.

4.4 Simulación del robot y cinemática inversa

La cinemática inversa consiste en hallar los ángulos de cada articulación dada la configuración deseada para la figura, en nuestro caso, la posición de la mano.

La decisión de mover los brazos por cinemática inversa hace que crear un simulador sea muy necesario (hacer las pruebas directamente en el robot entraña cierto peligro).

La simulación ha de servir para poder probar la aplicación sin tener que tener conectado el robot.

La simulación tendrá que tener cierto control sobre:

-No tratar de introducir dentro del cuerpo la mano.

-No tratar de posicionar la mano en posiciones no alcanzables.

5 Implementación

La implementación está compuesta por los siguientes pasos ordenados cronológicamente según han sido implementados.

5.1 Programación del firmware del Bioloid para el acceso a los Dynamixel

El código para el acceso ha de ser lo más sencillo y rápido posible.

La comunicación entre el ordenador y el Bioloid se realiza por serie (usando un adaptador USB-serie en este caso). Con lo cual cuanto más cortos y sencillos sean los mensajes mejor reaccionarán las articulaciones.

La estructura de los mensajes implementada es la siguiente:

XX YYYY

Donde XX es el número identificador del Dynamixel que debe recibir la orden e YYYY es la "cantidad de movimiento" o lo que es lo mismo, el ángulo al que queremos que vaya el motor. Esta cantidad de movimiento está codificada en una escala de 0 a 1023 representando los 360 grados.
El diagrama de flujo es el siguiente:

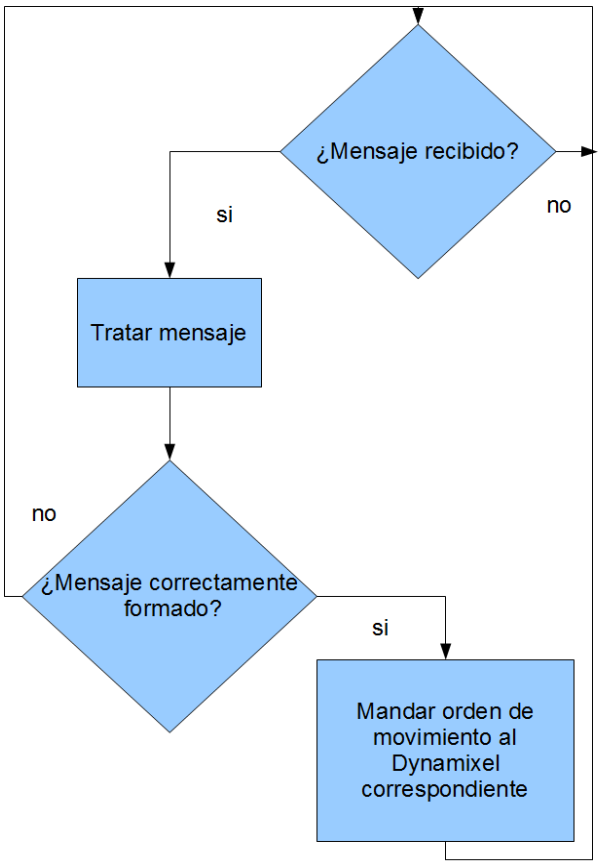


Ilustración 13: Diagrama de flujo firmware Bioloid

El código fuente consta en el anexo.

5.2 Programación del simulador con cinemática inversa del Bioloid

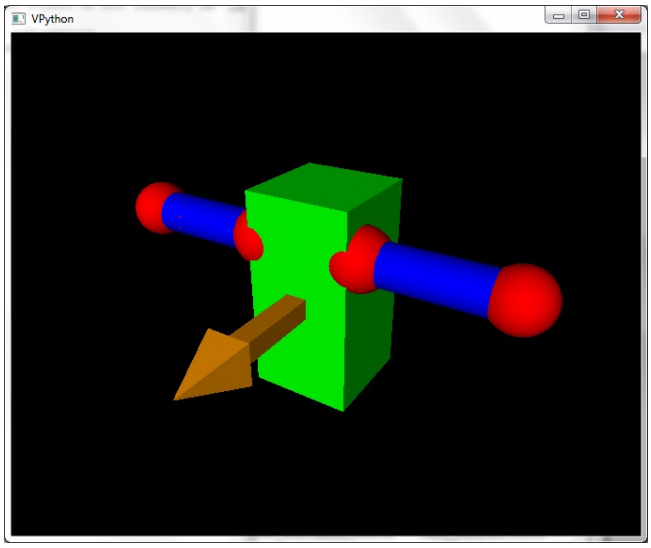


Ilustración 14: Estructura tronco superior bioloid en simulación

Se midió el robot y se consiguieron las siguientes medidas aproximadas:

-Distancia hombro a hombro (ancho) 10cm.
-Distancia hombro a codo 9cm.
-Distancia codo a mano 4,3cm.
-Distancia hombro a "cadera" 20cm.
-Largo robot 10cm.

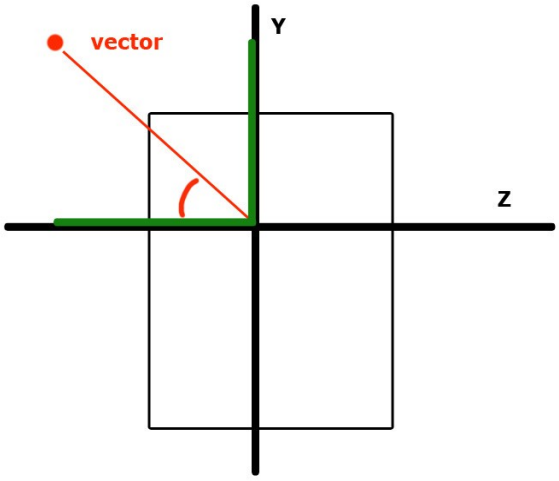
El simulador se implementó a base de formas simples conectando un cubo central, representando el torso, a los brazos. Estos están representados por unos cilindros (uno para el brazo y otro para el antebrazo) con unas esferas que marcan las articulaciones y las manos.

Gracias a Vpython, crear esta simulación es muy sencillo. Basta con declarar el tipo de estructura (box, cylinder, sphere) sus coordenadas, su tamaño y su vector direccional y ya está representado.

El código de la declaración se encuentra en el anexo.

5.2.1 Cinemática inversa

Rotación del hombro:

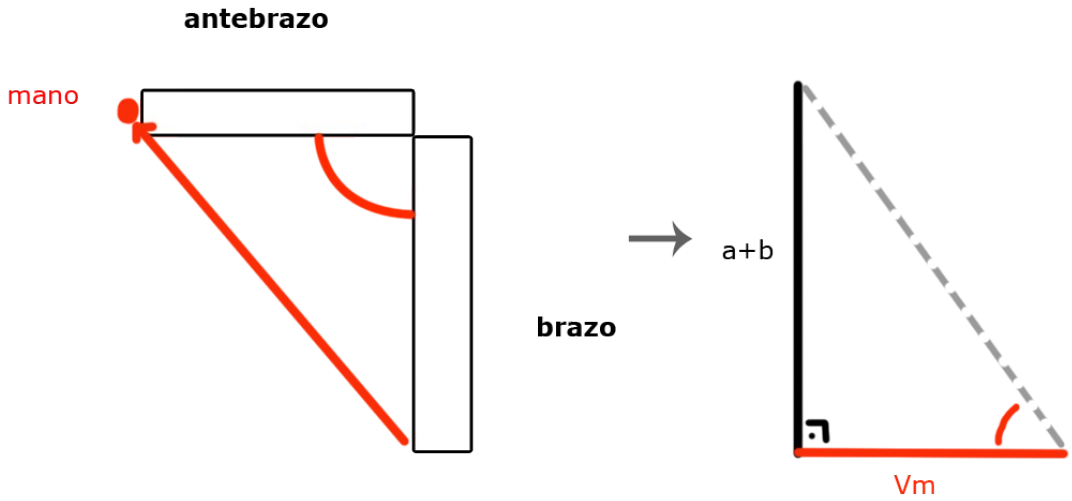


Tenemos el vector V que es el que va desde el eje (hombro) al punto de la mano. Para sacar el ángulo de rotación operaremos:

$$\alpha = \arctan(Vy, Vz)$$

Y esto lo negaremos dado que nos da el ángulo en negativo.

Ángulo del codo:



Dado que no podemos depender de en que ejes están situadas las partes del brazo la forma más sencilla de proceder es tratar directamente con las distancias en un plano imaginario.

Primero sacamos la distancia de un cateto de nuestro triángulo imaginario: Cateto adyacente:

(Vm = Vector mano, a = antebrazo, b = brazo)

$$|Vm| \leqslant b + a$$
$$|Vm|^2 \leqslant b^2 + a^2 + 2ab$$
$$|Vm|^2 - a^2 - b^2 \leqslant 2ab$$
$$\frac{|Vm|^2 - a^2 - b^2}{2ab} \leqslant 1$$

$$C = \frac{|Vm|^2 - a^2 - b^2}{2ab}$$

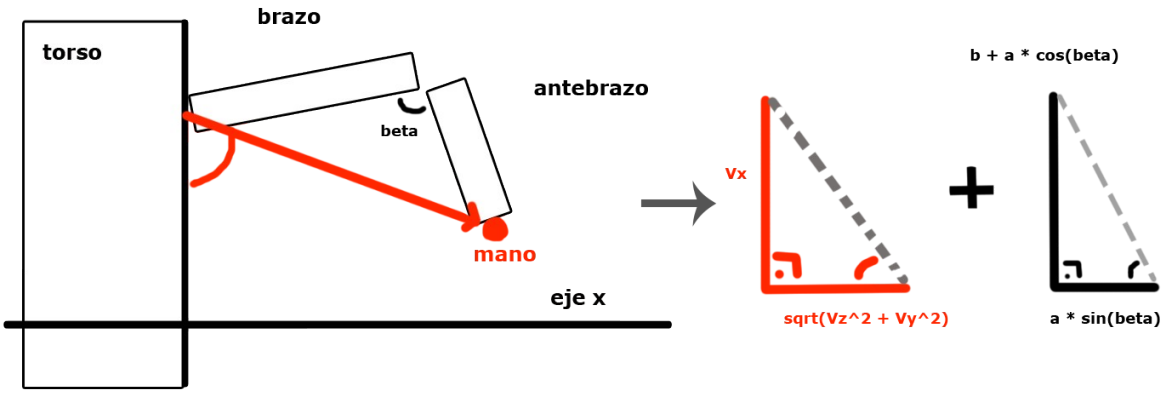
Luego sacamos el otro cateto, que estará en relación a este:

$$D = \sqrt{1 - C^2}$$

Y sacamos el ángulo calculando la arcotangente como en la articulación anterior:

$$\alpha = \arctan(C, D)$$

Ángulo del hombro:



Solo podemos depender del eje X dado que es sobre el cual el ángulo se moverá (imaginariamente partiendo siempre de ahí), con lo cual tendremos que trabajar como en el caso anterior con las distancias para conseguir el cateto adyacente pero teniendo en cuenta el ángulo del codo.

$$Si\ Vx > 0:$$
$$\alpha = -\arctan\left(\frac{Vx}{\sqrt{Vz^2 + Vy^2}}\right) + \arctan\left(\frac{a * \sin(\beta)}{b + a * \cos(\beta)}\right) + \frac{\pi}{2}$$

$$Si\ Vx \leq 0:$$
$$\alpha = \arctan\left(\frac{Vx}{\sqrt{Vz^2 + Vy^2}}\right) - \arctan\left(\frac{a * \sin(\beta)}{b + a * \cos(\beta)}\right) + \frac{\pi}{2}$$

Todo esto habrá que multiplicarlo por su factor de escala para pasarle la información en el formato apropiado a los Dynamixel.

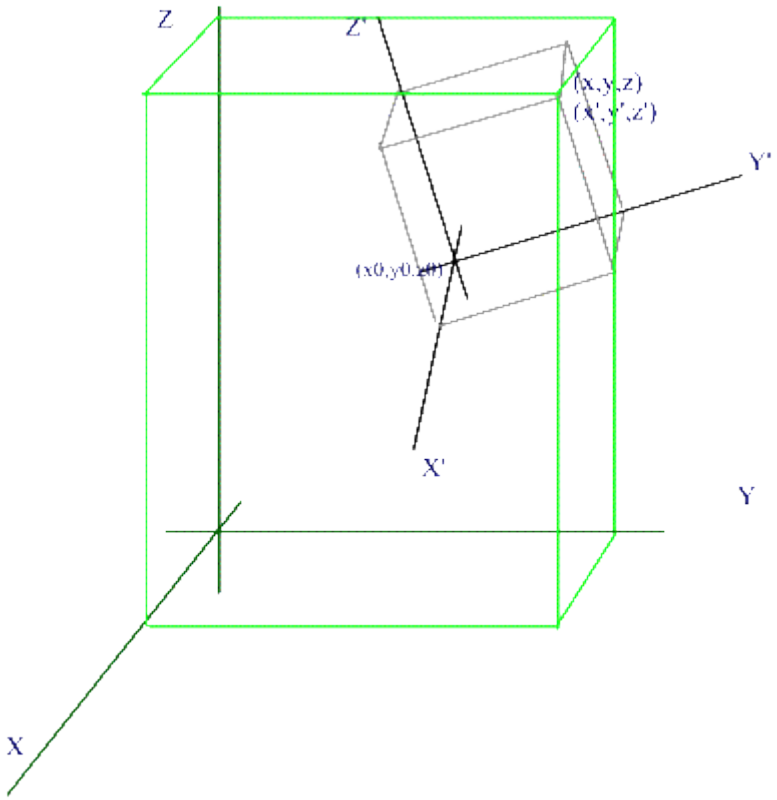
Además los ángulos imaginarios de los que partimos no coinciden con los de los Dynamixel, así que tendremos que hacer una pequeña calibración.
Esta calibración consiste en sumar 146º de offset a todos los ángulos.

5.3 Programación del simulador de coordenadas de Kinect (3D) con el cambio sistema de coordenadas

Para dar uso a nuestra cinemática inversa necesitamos las coordenadas de las manos en el mismo sistema de coordenadas en el que se encuentra nuestra simulación. Para ello el primer paso es conseguir las coordenadas de las manos en referencia al pecho del usuario del Kinect. En este caso he tomado el origen de coordenadas en el hombro izquierdo por comodidad. Luego solo hace falta hacer una traslación de coordenadas para situarlo en el origen del simulador (el centro del cubo que representa el torso).

Procedimiento:
Sean (x,y,z) las coordenadas del punto respecto a los ejes de coordenadas X-Y-Z.Sean (x0,y0,z0) las coordenadas del origen de coordenadas de los ejes X-Y-Z respecto al nuevo sistema de coordenadas X'-Y'-Z'.

Sean a,b,c el ángulo que se gira el eje X' respecto a los ejes X-Y-Z, sean d,e,f el ángulo que se gira el eje Y' respecto a los ejes X-Y-Z, y sean g,h,i el ángulo que se gira el eje Z' respecto a los ejes X-Y-Z.



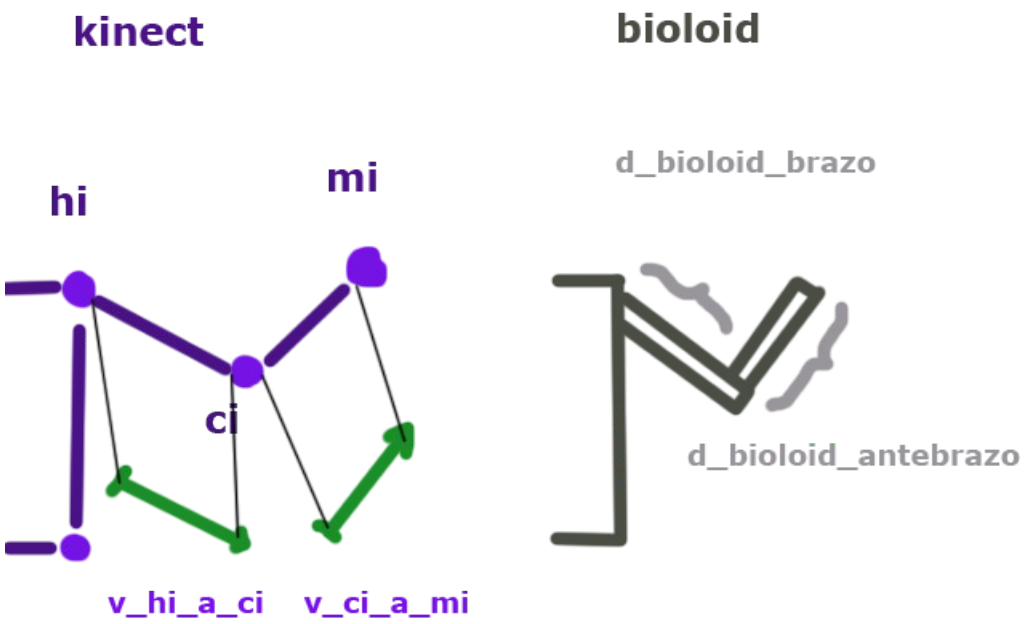
$$x = x_0 + x'\cos(a) + y'\cos(d) + z'\cos(g)$$
$$y = y_0 + x'\cos(b) + y'\cos(e) + z'\cos(h)$$
$$z = z_0 + x'\cos(c) + y'\cos(f) + z'\cos(i)$$

Será el sistema de ecuaciones que nos dará las coordenadas de nuestro punto en referencia al nuevo sistema de coordenadas.

5.4 Programación del escalado de las coordenadas conseguidas tras el cambio de sistema de referencia

Dado que el cuerpo humano es distinto al cuerpo del Bioloid (principalmente nuestro antebrazo sumado a nuestra mano es mas largo que nuestro brazo, mientras en el Bioloid pasa al contrario) necesitamos hacer un escalado de las coordenadas del punto teniendo esto en cuenta.

Para el brazo izquierdo (el proceso es muy similar para el derecho):



Hombro izquierdo, codo izquierdo, mano izquierda se refiere a las coordenadas de estos puntos.

d_brazo = distancia(hombro izquierdo, codo izquierdo)
d_antebrazo = distancia(codo izquierdo, mano izquierda)
d_bioloid_brazo = 0.09
d_bioloid_antebrazo = 0.043
d_todo_kinect = d_brazo + d_antebrazo
d_todo_bioloid = d_bioloid_brazo + d_bioloid_antebrazo

v_hi_a_ci = (codo izquierdo – hombro izquierdo)
v_hi_a_ci = (codo izquierdo / modulo(codo izquierdo)) * d_bioloid_brazo
v_ci_a_mi = (mano izquierda – codo izquierdo)
v_ci_a_mi = (v_ci_a_mi /modulo(v_ci_a_mi)) * d_bioloid_antebrazo

punto mano izquierda = v_hi_a_ci + v_ci_a_mi + (0.05, 0.05, 0.025)
Donde (0.05, 0.05, 0.025) es la traslación al origen de coordenadas.

Lo que hacemos es usar los vectores proporcionales al tamaño del brazo y del antebrazo para situar correctamente y en escala la posición de la mano.

5.5 Combinación de ambos simuladores para la aplicación final

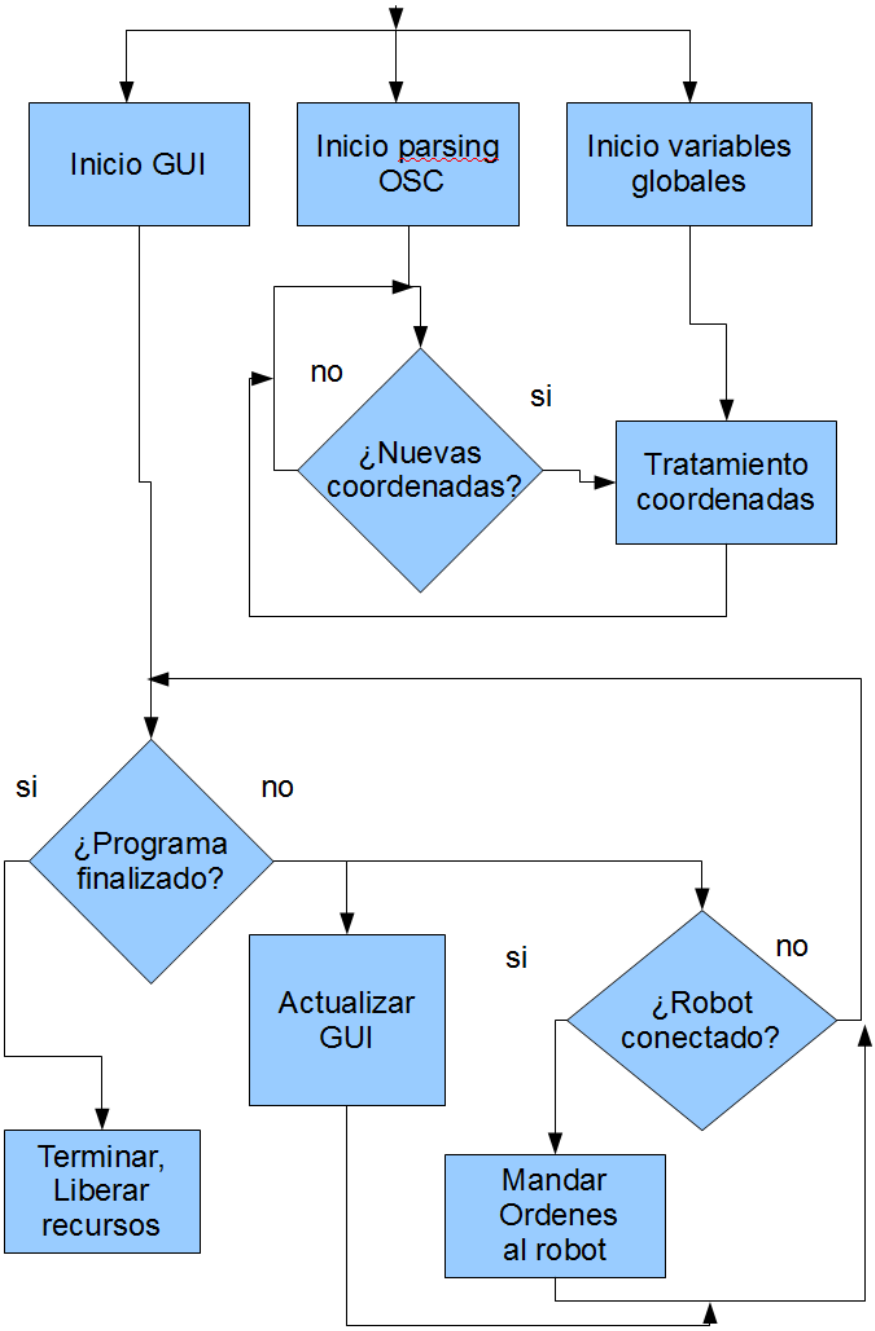


Ilustración 15: Diagrama de flujo de la aplicación

Inicialmente se inicia el GUI (Graphical User Interface) en el cual se presenta la simulación del Bioloid. A su vez se inicia el hilo de ejecución que se encargará de mantener una estructura de variables con las coordenadas de los puntos que nos interesan a tiempo real. Mientras el programa siga corriendo se actualizará la GUI continuamente y en caso de estar conectado el robot se le enviarán las ordenes correspondientes a cada Dynamixel.

6 Resultados

Se ha demostrado la viabilidad del uso de un sensor Kinect como interficie útil y en tiempo real para el control gestual de los brazos de un robot humanoide. Vídeo en youtube: http://www.youtube.com/watch?v=wi3l8W-CCHM

Ilustración 16: Capturas de un video de uso de la aplicación

En cuanto a la **fiabilidad de los movimientos** durante el proyecto se han realizado pruebas de evaluación de los resultados en los pasos intermedios tal como podemos encontrar en el diagrama de Gantt. Concretamente se han realizado las siguientes pruebas para cada tarea:

-Implementación de un firmware para el control de los actuadores:

Para cada movimiento de cada Dynamixel se leía el valor de la posición objetivo, se le mandaba como orden y luego se leía la posición actual.

-Implementación de la cinemática inversa de los brazos:

Se probaron casos simples (en los ejes X,Y,Z) y casos no triviales de posiciones de la mano. En estos casos se comprobaba, primero visualmente en el simulador (a grandes rasgos si tenía sentido), luego analizando los resultados de las coordenadas y ángulos obtenidos con un modelo hecho a mano, si estos datos eran correctos.

-Incorporación de la librería de Skeleton Tracking:

Se comprobó que los datos sobre el esqueleto fueran un flujo constante contando el número de actualizaciones de cada articulación por segundo. También se comprobó que las coordenadas no sufrieran saltos bruscos de posición analizando las coordenadas de varias escenas manualmente.

-Simulador para evaluación de los datos del esqueleto:

Se comprobó que el cambio de sistema de coordenadas para los puntos del esqueleto eran correctos, primero visualmente en el simulador (a grandes rasgos para saber si tenía sentido), luego analizando los resultados de las coordenadas haciendo los cálculos manualmente.

-Simulador para simulación de los movimientos del robot:

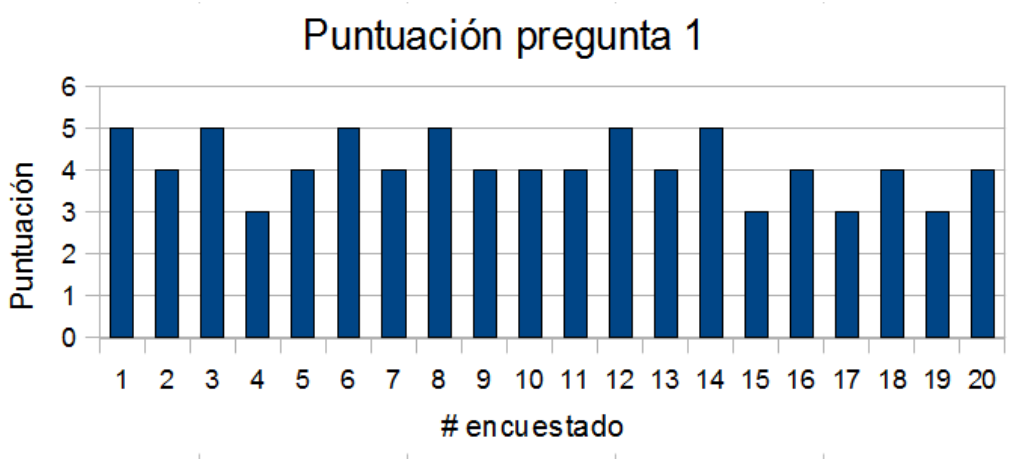
La comprobación se realizó al implementar la cinemática inversa de los brazos.

En cuanto al **tiempo de respuesta** se han analizado varios videos sobre el uso de la aplicación donde se ha observado (observando cuantos frames pasan desde que comienza un movimiento hasta que el brazo del robot empieza a moverse imitándolo) que el retraso es menor a 15ms en condiciones de reposo a movimiento.
En caso de que el robot este moviendo el brazo y se le fuerce a cambiar de dirección, por la dinámica de los Dynamixel este tiempo aumenta.

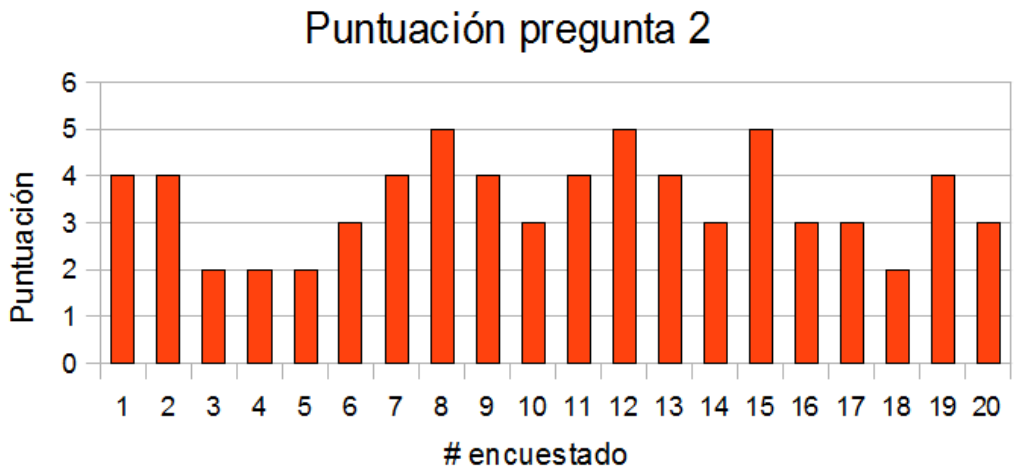
Se ha realizado una pequeña encuesta entre 20 personas que han probado el simulador. La prueba ha consistido en jugar durante unos minutos con el sistema ejecutando movimientos sencillos al principio, para habituarse, y luego moverse con libertad.
En esta encuesta se le pedía al usuario que puntuara del 1 al 5 las siguientes preguntas:

1. Al mover solo un brazo, ¿en que grado te ha parecido que el robot respondía bien?
2. Al mover los dos brazos, ¿en que grado te ha parecido que el robot respondía bien?
3. En general, ¿piensas que la posición de las manos del robot son fieles a las del usuario?
4. En general, ¿piensas que el robot responde rápidamente a los movimientos del usuario?

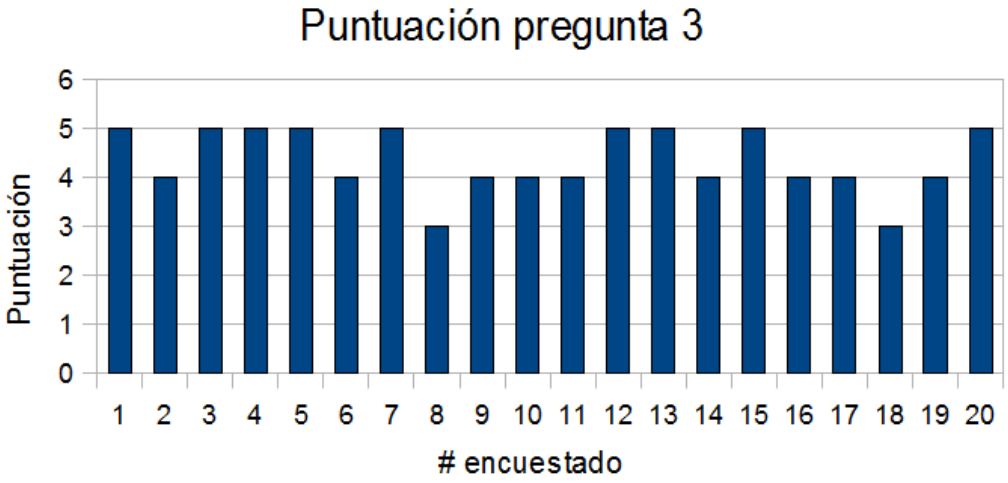
De donde sacamos las siguientes respuestas:



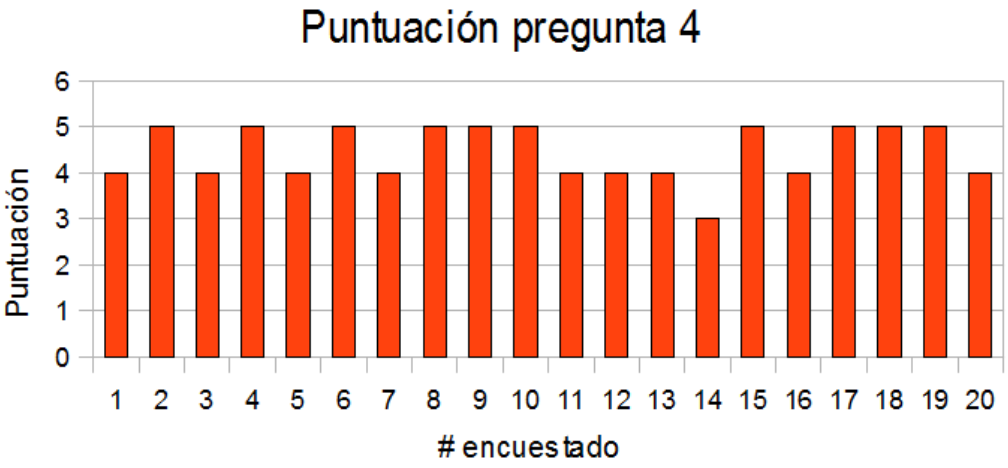
La cual nos da una media de 4,1.



La cual nos da una media de 3,45.



La cual nos da una media de 4,35.



La cual nos da una media de 4,45.

7 Análisis económico

Concepto	Horas	Precio hora	Total
Diseño	45	70,00 €	3.150,00 €
Mecánica	15	25,00 €	375,00 €
Programación	240	25,00 €	6.000,00 €
Total	300		9.525,00 €

Material	Precio	Precio Final
Bioloid Premium Kit	1.200,00$*	850,00 €
Kinect	150,00 €	150,00 €
Estación de trabajo	1.000,00€**	167,00 €
Total		1.167,00 €

* Cambio actual 1€ = 1,41$
** Amortización a 24 meses, uso para el proyecto: 4 meses.

Coste total del proyecto: 10.692,00€

7.1 Argumentos económicos del proyecto:

¿Que aporta al sensor Kinect este proyecto?
Da promoción dado que se le da uso para un producto que puede llamar la atención.

¿Que aporta al robot este proyecto?
Principalmente es un proyecto reproducible que añade interés didáctico. A partir de él se pueden montar otras configuraciones del Bioloid y manejarlas a través del Kinect.

Se podría crear algún tipo de juego de recreativa donde los brazos podrían coger una pelota y tirarla a algún objetivo.

Podría construirse un pequeño show de promoción, al pasar por delante de la configuración del proyecto uno podría mover el robot. Al atraer la atención del posible consumidor/cliente se afianzaría el producto en promoción.

8 Conclusiones

Para conseguir realizar este proyecto se han tenido que integrar muchos elementos. Al ser punta de lanza de este tema no había demasiado material del que partir con lo cual nos hemos visto obligados a probar diferentes opciones en cuanto a enfoques del problema y herramientas utilizadas.
Dado el carácter de investigación y desarrollo del proyecto se encuentra fuera de nuestro alcance la evaluación precisa del sistema de guiado ya que esto sería origen de otro proyecto.

El objetivo de este proyecto ha sido posible con relativo poco esfuerzo gracias a que la tecnología de Kinect nos permite conseguir un esqueleto de un usuario fácil y fielmente. Aún así el tratamiento de los datos para que fuera útiles requirió un esfuerzo considerable por no tener trabajos anteriores en los cuales basarse.

Ha resultado ser un ejercicio muy interesante y una gran oportunidad para aprender conceptos importantes como cinemática inversa, programación con gráficos en 3D y repasar conceptos aplicando geometría y trigonometría.

Python ha resultado ser un lenguaje muy sencillo de aprender teniendo una base de programación y que permite obtener grandes resultados en poco tiempo.

El Bioloid Premium Kit es un paquete increíble para adentrarse en el mundo de la robótica. Hay bastantes recursos en la web para conseguir comenzar cualquier proyecto que uno se plantee.

9 Lineas futuras de trabajo

Sería muy posible, y probablemente se realizará, la obtención del movimiento del resto del cuerpo (especialmente las piernas) con un método similar. En principio lo más sencillo sería simplemente mover las piernas, sin caminar. Esto es así ya que caminar implica ser consciente de la dinámica del cuerpo para que este no caiga, para ello necesita de sensores para poder facilitar esta tarea y un sistema que los monitorice para actuar como sea debido en cada caso.

10 Bibliografía

Especificaciones Dynamixels:

http://www.robosavvy.com/RoboSavvyPages/Support/Bioloid/AX-12%28english%29.pdf

Documentación OpenNI

http://www.openni.org/documentation

OSCeleton, de SenseBloom

https://github.com/Sensebloom/OSCeleton

Trigonometría, Wikipedia:

http://en.wikipedia.org/wiki/Inverse_trigonometric_functions

Vectores en R3 (Moises Villena)

http://es.scribd.com/doc/8689496/Vectores-en-R3#ad_unit=Doc_Sideboard_MediumRectangle_BTF_300x250&url=http%3A//es.scribd.com/doc/8689496/Vectores-en-R3&attributes=QuantSegs%3DD%26FourGen%3DTrue%26IABParent%3DTechnology%2520%2526%2520Computing%26Extension%3Dpdf%26Gender%3Dm%26User%3D26455116%26AdLayout%3D-1472436212%26DocUser%3D2396611%26UserState%3DIn%26IABChild%3DVisual%2520Basic%252CC/C%26Document%3D8689496&skip=0

Sistemas de coordenadas, wikipedia:

http://es.wikipedia.org/wiki/Sistema_de_coordenadas

Cambio sistema coordenadas

http://www.telefonica.net/web2/lasmatematicasdemario/Geometria/Analitica/Coordenadas/CoorRect.htm

Repaso geometría y trigonometría

http://thales.cica.es/rd/Recursos/rd99/ed99-0543-04/Distancia.html

Funciones de numpy para Python:

http://mathesaurus.sourceforge.net/matlab-numpy.html

Documentación oficial Python:

http://www.python.org/doc/

Documentación oficial Vpython:

http://www.vpython.org/contents/doc.html

Documentación pyserial:

http://pyserial.sourceforge.net/

Documentación simpleOSC (para python):

http://www.ixi-software.net/content/body_backyard_osc.html

Links sobre cinemática inversa:

http://support.robotis.com/en/software/roboplus/roboplus_motion/motionedit/poseedit/poseutility/roboplus_motion_ik.htm

http://apollo.upc.es/humanoide/trac/wiki/BioloidKinematics

http://www.fcet.staffs.ac.uk/sow1/robotmaterial/manipulator_kinematics.htm

http://eris.liralab.it/wiki/KDL-simple

http://es.wikipedia.org/wiki/Cinem%C3%A1tica_inversa

https://sites.google.com/site/proyectosroboticos/cinematica-inversa-i

11 Agradecimientos

AESS:

Javier Férnandez, Jonathan Gonzalez, Miquel Farré, Pau Beltran, Hilario Tome, Lluis Rovira, Andrés Pardo, etc. todos los miembros que han estado ahí echando una mano siempre que me ha hecho falta y me han ayudado a mantener el ánimo.

ESAII:

Joan Aranda, director de este PFC gracias al cual he podido aprender muchos conceptos que seguro me serán útiles en el futuro.

Josep Ruzafa, que me ayudó a encaminarme hacia un PFC.

Otros:

Jordi Tur Escandell, amigo y consultor en cualquier duda que tuviera matemática, gracias por tu esfuerzo.

Y en general a toda esa gente que estuvo ayudándome ya fuera con conocimientos o con su actitud.

12 Anexos:

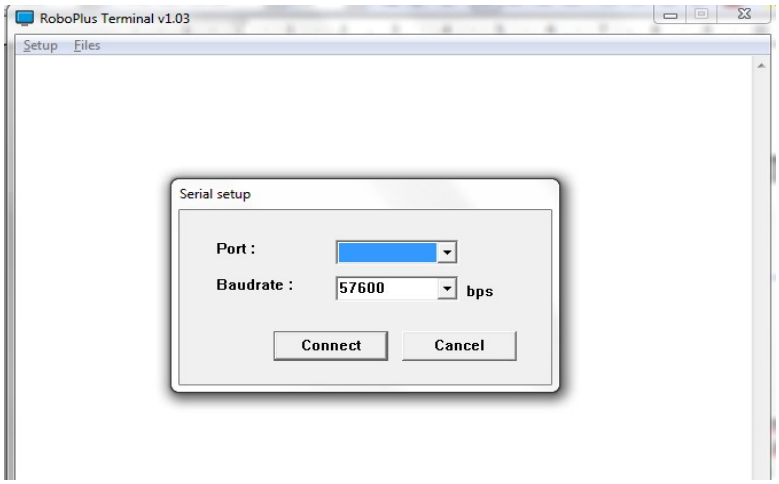
12.1 Documentación de la aplicación, guía instalación, guía de uso

12.1.1 Bioloid

Windows:

Necesitaremos instalar el firmware en el Bioloid. El firmware está compuesto por un archivo .hex. Para instalarlo podemos instalar RoboPlus de ROBOTIS. (Link y guía de instalación: http://support.robotis.com/en/software/roboplus_main.htm).

Este software contiene un terminal llamado RoboPlus Terminal.
Conectamos al puerto asignado en nuestro ordenador como puerto COM (a 57600 baudios).



Una vez conectado al Bioloid deberemos mantener pulsado # (AltGr+3) hasta que nos salga una consola. Borramos los caracteres "#" de más que nos han salido y:
-Ejecutamos el comando "LD" (sin las comillas).
-Vamos a Files → Transmit File y seleccionamos el .hex de nuestro firmware.
Para correr este nuevo firmware podemos reiniciar el Bioloid o escribir "GO".

Linux:
Nos hará falta algún tipo de consola para interactuar con serie y repetir las instrucciones anteriores.

12.1.2 Ordenador

12.1.2.1 Instalación entorno Python Windows:

Necesitaremos instalar Python, en este proyecto se ha usado Python 2.6:

http://sourceforge.net/projects/pywin32/files/pywin32/Build216/pywin32-216.win32-py2.6.exe/download

Instalamos PySerial para el soporte para serie:

http://pyserial.sourceforge.net/

Descarga: http://pypi.python.org/pypi/pyserial
Guía de instalación: http://pyserial.sourceforge.net/pyserial.html#installation

Instalamos SimpleOSC:

http://www.ixi-audio.net/content/body_backyard_python.html

Para ello instalamos primero pyOSC:
https://trac.v2.nl/wiki/pyOSC

Instalamos NumPy:

http://numpy.scipy.org/
Descarga: http://www.scipy.org/Download
Guía de instalación: http://www.scipy.org/Installing_SciPy

Instalamos Vpython:

http://www.vpython.org/index.html
Descarga y guía: http://www.vpython.org/contents/download_windows.html

Instalamos wxPython:

http://www.wxpython.org/download.php

12.1.2.2 Instalación entorno python Linux:

Hay que instalar las mismas librerías que en Windows.

12.1.2.3 Instalación OSCeleton

Objetivo usar: https://github.com/Sensebloom/OSCeleton#readme
(Ejemplos a compilar https://github.com/Sensebloom/OSCeleton-examples)

Que como dice hay que instalar lo que nos diga: https://github.com/avin2/SensorKinect

Video muestra: http://urbanhonking.com/ideasfordozens/2011/02/16/skeleton-tracking-with-kinect-and-processing/

Empezamos por SensorKinect:
SensorKinect (de su Readme):
Install notes:
1) Install unstable OpenNI (http://www.openni.org/downloadfiles/openni-binaries/20-latest-unstable) (OpenNI-Bin-Win32-v1.0.0.25.exe en la carpeta) On Linux/Mac please do: ./sudo install.sh

2) Install Sensor (this version...)(avin2-SensorKinect-0124bd2.zip en la carpeta) On Linux/Mac please do: ./sudo install.sh

3) Install unstable NITE (http://www.openni.org/downloadfiles/openni-compliant-middleware-binaries/33-latest-unstable) (NITE-Bin-Win32-v1.3.0.18.exe en la carpeta) On Linux/Mac please do: ./sudo install.sh
Use this license when asked during the installation: 0KOIk2JeIBYClPWVnMoRKn5cdY4=

3.5) Conectar Kinect y aceptar las instalaciones de drivers de Windows.

4) Test #1: Run the NiViewer sample to make sure depth & image streams are working.

5) Test #2: Run the OpenNI/NiUserTracker sample play with the skeleton. Don't forget to start with the calibration pose! (Explained in the PDFs)

5.9) Para que funcione el último test (Y otras cosas mas adelante):
Note: All of the NITE samples are using 320x240 resolution. You need to change it to 640x480 in the XML files inside the Data directory. I have prepared preconfigured XMLs in the "NITE\Data" dir. Just copy them to "c:\Program Files\Prime Sense\NITE\Data".

6) Test #3: Try the NITE/Sample-PointViewer sample for the hand tracking demo.

E instalamos OSCeleton:

OSCeleton-1.0_win32.zip
Ejecutamos el ejecutable.
Este es el servidor OSCeleton, los mensajes salen en
https://github.com/Sensebloom/OSCeleton#readme

Podemos probarlo con animata: http://animata.kibu.hu/downloads.htmlanimata_win-004.zip

Para ello tenemos que ejecutar OSCeleton.exe asi:
OSCeleton.exe -k -mx 640 -my 480 -ox -160

Si se quiere probar alguna demo:
https://github.com/Sensebloom/OSCeleton-examples

12.1.2.4 Uso de la aplicación

Por un lado debemos ejecutar OSCeleton de la siguiente manera:

OSCeleton.exe -w -r

Donde -w sirve para ver el visor de imagen de profundidad.

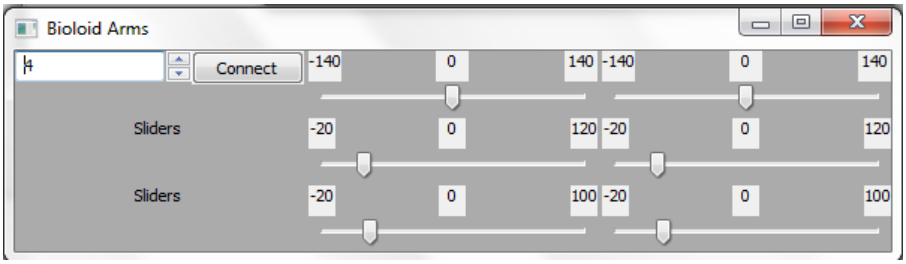
Donde -r sirve para obtener los datos en modo “mirror” que es como funciona el código en python.

Para que OSCeleton nos empiece a hacer tracking debemos poner la “Psi” pose:

Por otro lado tenemos que ejecutar el código python:

Simplemente podemos ejecutar BioloidArmsMainKinectOSC.py.

Para hacer que el Bioloid ejecute los movimientos habrá que elegir el puerto serie que está usando y darle al botón Connect.



12.2 Anexo código fuente firmware Bioloid

// Codigo para coger los mensajes por serie y ejecutar los movimientos

```
#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include "serial.h"
#include "dynamixel.h"

/// Control table address
#define P_GOAL_POSITION_L               30
#define P_GOAL_POSITION_H               31
#define P_PRESENT_POSITION_L    36
#define P_PRESENT_POSITION_H    37
#define P_MOVING                                46

// Default setting
#define DEFAULT_BAUDNUM         1 // 1Mbps
#define DEFAULT_ID                      1

void PrintCommStatus(int CommStatus);
void PrintErrorCode(void);

int main(void)
{
    int Value = 0;
    unsigned short GoalPos[2] = {0, 1023};
    int index = 0;
    int id = 1; // id de motor
    int movimiento = 0;
    int bMoving, wPresentPos;
    int CommStatus;

    serial_initialize(57600);                               // USART Initialize
    dxl_initialize( 0, DEFAULT_BAUDNUM ); // Not using device index
    sei();

    //printf( "\n\nSerial Comm. moving motor\n\n" );

    while (1)
    {
        char ReceivedData[20];
        gets(ReceivedData);
        if(strlen(ReceivedData) < 3)
            printf("Error. Formato orden: IdMotor CantMov\n");
        else{
```

```
46.             //Parsear mensaje
47.             char * pch;
48.             pch = strtok (ReceivedData," ");
49.             for(int i=0; pch != NULL; i++)
50.             {
51.                 if(i==0) id=atoi(pch); //coger id motor
52.                 if(i==1) movimiento=atoi(pch);// coger cantidad de movimiento
53.                 pch = strtok (NULL, " ,.-");
54.             }
55.
56.             // Ejecutar orden
57.             dxl_write_word( id, P_GOAL_POSITION_L, movimiento);

58.             PrintErrorCode(); //Por si hay algun error
59.             }
60.     }
61.
62.     return 1;
63.}
64.
65.
66.
67.// Print communication result
68.void PrintCommStatus(int CommStatus)
69.{
70.     switch(CommStatus)
71.     {
72.     case COMM_TXFAIL:
73.         printf("COMM_TXFAIL: Failed transmit instruction packet!\n");
74.         break;
75.
76.     case COMM_TXERROR:
77.         printf("COMM_TXERROR: Incorrect instruction packet!\n");
78.         break;
79.
80.     case COMM_RXFAIL:
81.         printf("COMM_RXFAIL: Failed get status packet from device!\n");
82.         break;
83.
84.     case COMM_RXWAITING:
85.         printf("COMM_RXWAITING: Now recieving status packet!\n");
86.         break;
87.
88.     case COMM_RXTIMEOUT:
89.         printf("COMM_RXTIMEOUT: There is no status packet!\n");
90.         break;
91.
```

```
92.     case COMM_RXCORRUPT:
93.         printf("COMM_RXCORRUPT: Incorrect status packet!\n");
94.         break;
95.
96.     default:
97.         printf("This is unknown error code!\n");
98.         break;
99.     }
100.}
101.
102.// Print error bit of status packet
103.void PrintErrorCode()
104.{
105.     if(dxl_get_rxpacket_error(ERRBIT_VOLTAGE) == 1)
106.         printf("Input voltage error!\n");
107.
108.     if(dxl_get_rxpacket_error(ERRBIT_ANGLE) == 1)
109.         printf("Angle limit error!\n");
110.
111.     if(dxl_get_rxpacket_error(ERRBIT_OVERHEAT) == 1)
112.         printf("Overheat error!\n");
113.
114.     if(dxl_get_rxpacket_error(ERRBIT_RANGE) == 1)
115.         printf("Out of range error!\n");
116.
117.     if(dxl_get_rxpacket_error(ERRBIT_CHECKSUM) == 1)
118.         printf("Checksum error!\n");
119.
120.     if(dxl_get_rxpacket_error(ERRBIT_OVERLOAD) == 1)
121.         printf("Overload error!\n");
122.
123.     if(dxl_get_rxpacket_error(ERRBIT_INSTRUCTION) == 1)
124.         printf("Instruction code error!\n");
125.}
```

12.3 Código fuente aplicación python:

```
from armgraphics import *
```

```
1. import wx
2. from ArmGui2 import *  #Import the Gui
3. import serial
4. import time
5. import thread
6. from simpleOSC import *
7. import numpy as np
8.
9.
10.
11.#Funciones utiles:
12.#Modulo de un vector
13.def modulus_vector(x):
14.   return np.sqrt((x*x).sum())
15.
16.#Distancia entre dos puntos:
17.def dist(x,y):
18.   return np.sqrt(np.sum((x-y)**2))
19.
20.#Cos(angulo) entre dos vectores
21.def cosangle(x,y):
22.   dot = np.dot(x,y)
23.   x_modulus = np.sqrt((x*x).sum())
24.   y_modulus = np.sqrt((y*y).sum())
25.   return dot / x_modulus / y_modulus
26.# np.rad2deg(angle)
27.
28.#Angulo entre dos vectores en grados
29.def angle(x,y):
30.   dot = np.dot(x,y)
31.   x_modulus = np.sqrt((x*x).sum())
32.   y_modulus = np.sqrt((y*y).sum())
33.   cosang = dot / (x_modulus * y_modulus)
34.   return cosang * 360 / 2 / np.pi
35.
36.
37.#Devuelve la solucion en la traslacion
38.#vx vy vz = vectors del plano del Kinect
39.#point = punto a cambiar de un sistema de coordenadas al otro
40.#shoulderpoint = punto del hombro
41.def solve_coords(point, shoulderpoint, vx, vy, vz):
42.   #vectores unitarios origen
43.   vox = np.array((1,0,0))
44.   voy = np.array((0,1,0))
```

```
45.   voz = np.array((0,0,1))
46.
47.   x = np.array((cosangle(vx,vox),cosangle(vx,voy),cosangle(vx,voz)))
48.   y = np.array((cosangle(vy,vox),cosangle(vy,voy),cosangle(vy,voz)))
49.   z = np.array((cosangle(vz,vox),cosangle(vz,voy),cosangle(vz,voz)))
50.   A = np.array((x,y,z))
51.   return np.linalg.solve(A,point-shoulderpoint)
52.
53.   #x= x0 + x'cosa + y'cosd + z'cosg
54.   #y= y0 + x'cosb + y'cose + z'cosh
55.   #z= z0 + x'cosc + y'cosf + z'cosi
56.
57.
58.
59.
60.# Resolver la cinematica inversa, actualizar la animacion y mandar las ordenes
61.def do_IK(hand, handcoords): #para mano izq y para mano derecha, hand = 0 izq,
   hand = 1 der
62.   global ser # serie
63.   D = 0
64.   try:
65.     target.pos = handcoords[0][0], handcoords[0][1], handcoords[0][2]
66.   except:
67.     print sys.exc_info()
68.
69.   if (target.x > Torso.size.x/2. or target.x < -Torso.size.x/2.)\
70.   or (target.y > Torso.size.y/2 or target.y < -Torso.size.y/2.)\
71.   or (target.z > Torso.size.z/2 or target.z < -Torso.size.z/2):
72.
73.     if hand == 0: # Mano izquierda
74.       rvector.axis = target.pos - R_Shoulder_Joint.pos
75.       rvector.pos = R_Shoulder_Joint.pos
76.       total_lower_arm_length = Lower_Arm_Length
77.     # Mirar si el punto esta en el dominio. C es una variable temporal
78.     # igual a la diferencia enrte la extension maxima del brazo y el punto
   deseado
79.       C = (mag(rvector.axis)**2 - Upper_Arm_Length**2 -
   total_lower_arm_length**2)/(2*Upper_Arm_Length*total_lower_arm_length)
80.       if C**2 <= 1: #Computar cinematica inversa
81.         D = sqrt(1-C**2)
82.         elbow_angle = atan2(D,C)
83.         if rvector.axis.x > 0:
84.           shoulder_angle = -
   (atan2(rvector.axis.x,sqrt(rvector.axis.z**2+rvector.axis.y**2))\
85.           +atan2(total_lower_arm_length*sin(elbow_angle),
   (Upper_Arm_Length+total_lower_arm_length*cos(elbow_angle))))+pi/2
86.         else:
```

```
87.             shoulder_angle = (atan2(-
    rvector.axis.x,sqrt(rvector.axis.z**2+rvector.axis.y**2))\
88.             -atan2(total_lower_arm_length*sin(elbow_angle),
    (Upper_Arm_Length+total_lower_arm_length*cos(elbow_angle))))+pi/2
89.           # Z se vuelve Y y X se vuelve Z asi que Y se vuelve X
90.           shoulder_pivot_angle = -atan2(rvector.axis.y,rvector.axis.z)
91.
92.           shoulder_pivot_angle *= 180 / np.pi #pasamos a grados
93.           shoulder_angle *= 180 / np.pi
94.           elbow_angle *= 180 / np.pi
95.
96.           #Formamos los mensajes para el bioloid, los +146 son debidos a la
    posicion inicial de los servos
97.           #para que sean el 0 que nosotros esperamos tenemos que sumarle esa
    cantidad
98.           spa_angle = int(3.5*( (shoulder_pivot_angle + 146) % 360) )
99.           comando_spa = "%d %d\n" % (2, int(spa_angle)) # sintaxis comando bioloid
    "%d %d\n" num_motor cantidad
100.           sa_angle = int(3.5*( (shoulder_angle + 146) % 360))
101.           comando_sa = "%d %d\n" % (4, int(sa_angle))
102.           e_angle = int(3.5*( (elbow_angle + 146) % 360 ) )
103.           comando_e = "%d %d\n" % (6, int(e_angle))
104.
105.           try:
106.             ser.write(comando_spa)       # Mandar comando por serie
107.             ser.write(comando_sa)
108.             ser.write(comando_e)
109.           except:
110.             print (" Serial Port Not Connected ")
111.
112.           # Actualizar animacion
113.           frame_1.Right_Arm_Pivot.SetValue(shoulder_pivot_angle)
114.           frame_1.Right_Shoulder.SetValue(shoulder_angle)
115.           frame_1.Right_Elbow.SetValue(elbow_angle)
116.
    update_Torso(shoulder_pivot_angle,frame_1.Left_Arm_Pivot.GetValue(),shoulder_angle
    ,\
117.
    frame_1.Left_Shoulder.GetValue(),elbow_angle,frame_1.Left_Elbow.GetValue())
118.           target.color = color.orange
119.         else:
120.           print "Domain Error"
121.           target.color = color.red
122.           D = 0
123.   #El codigo para el brazo izquierdo es casi lo mismo, cambia algun signo y algun
    pi
124.     elif hand == 1:
```

```
125.        rvector.axis = target.pos - L_Shoulder_Joint.pos
126.        rvector.pos = L_Shoulder_Joint.pos
127.        total_lower_arm_length = Lower_Arm_Length
128.
129.        C = (mag(rvector.axis)**2 - Upper_Arm_Length**2 -
    total_lower_arm_length**2)/(2*Upper_Arm_Length*total_lower_arm_length)
130.        if C**2 <= 1:
131.          D = sqrt(1-C**2)
132.          elbow_angle = atan2(D,C)
133.          if rvector.axis.x > 0:
134.
    shoulder_angle = (atan2(rvector.axis.x,sqrt(rvector.axis.z**2+rvector.axis.y**2))\
135.            -atan2(total_lower_arm_length*sin(elbow_angle),
    (Upper_Arm_Length+total_lower_arm_length*cos(elbow_angle))))+pi/2
136.          else:
137.            shoulder_angle = -(atan2(-
    rvector.axis.x,sqrt(rvector.axis.z**2+rvector.axis.y**2))\
138.            +atan2(total_lower_arm_length*sin(elbow_angle),
    (Upper_Arm_Length+total_lower_arm_length*cos(elbow_angle))))+pi/2
139.          shoulder_pivot_angle = -atan2(rvector.axis.y,rvector.axis.z)
140.
141.          shoulder_pivot_angle *= 180 / np.pi #pasamos a grados
142.          shoulder_angle *= 180 / np.pi
143.          elbow_angle *= 180 / np.pi
144.
145.
146.          spa_angle = int(3.5*((-shoulder_pivot_angle + 146 ) % 360) )
147.          comando_spa = "%d %d\n" % (1, int(spa_angle))
148.          sa_angle = int(3.5*( (-shoulder_angle + 146) % 360))
149.          comando_sa = "%d %d\n" % (3, int(sa_angle))
150.          e_angle = int(3.5*((-elbow_angle + 146 ) % 360) )
151.          comando_e = "%d %d\n" % (5, int(e_angle))
152.
153.          try:
154.            ser.write(comando_spa)       # Mandamos comando por serie
155.            ser.write(comando_sa)
156.            ser.write(comando_e)
157.          except:
158.            print (" Serial Port Not Connected ")
159.
160.
161.          frame_1.Left_Arm_Pivot.SetValue(shoulder_pivot_angle)
162.          frame_1.Left_Shoulder.SetValue(shoulder_angle)
163.          frame_1.Left_Elbow.SetValue(elbow_angle)
164.
    update_Torso(frame_1.Right_Arm_Pivot.GetValue(),shoulder_pivot_angle,frame_1.Right
    _Shoulder.GetValue(),\
```

```
            shoulder_angle,frame_1.Right_Elbow.GetValue(),elbow_angle)
            target.color = color.orange
        else:
          print "Domain Error"
          target.color = color.red
          D = 0
    else:
      print "Domain Error"
      target.color = color.red
      D = 0

def update_Torso(R_shoulder_pivot,L_shoulder_pivot,R_shoulder,L_shoulder,R_elbow,
  L_elbow):  #Actualizar animacion
    #Conectamos brazo a hombro
    R_Upper_Arm.pos = R_Shoulder_Joint.pos
    L_Upper_Arm.pos = L_Shoulder_Joint.pos

    R_Upper_Arm.axis =rotate(vector(0,0,Upper_Arm_Length),angle=(frame_1.Right_Arm_Piv
    ot.GetValue()/180.)*pi,\
    axis = vector(Upper_Arm_Length,0,0)) # Rotamos brazo
    L_Upper_Arm.axis =rotate(vector(0,0,Upper_Arm_Length),angle=(frame_1.Left_Arm_Pivo
    t.GetValue()/180.)*pi,\
    axis = vector(Upper_Arm_Length,0,0)) # Rotamos brazo

    R_Upper_Arm.axis = rotate(R_Upper_Arm.axis,angle=-
    (frame_1.Right_Shoulder.GetValue()/180.)*pi+pi/2,\
    axis = norm(cross(R_Upper_Arm.axis,vector(1,0,0)))) #Rotamos brazo
    L_Upper_Arm.axis = rotate(L_Upper_Arm.axis,angle=(frame_1.Left_Shoulder.GetValue()
    /180.)*pi-pi/2,\
    axis = norm(cross(L_Upper_Arm.axis,vector(1,0,0)))) #Rotatamos brazo

    R_Elbow.pos = R_Shoulder_Joint.pos + R_Upper_Arm.axis #Conectamos codo
    L_Elbow.pos = L_Shoulder_Joint.pos + L_Upper_Arm.axis

    R_Lower_Arm.pos = R_Elbow.pos        #Conectamos antebrazo
    R_Lower_Arm.axis = norm(R_Upper_Arm.axis)*Lower_Arm_Length

    if R_shoulder <= 0 :  #Parche para ambiguedad de angulo
      Rtemp = -1
    else:
      Rtemp = 1
    R_Lower_Arm.axis =rotate(R_Lower_Arm.axis,angle=+
    (frame_1.Right_Elbow.GetValue()/180.)*pi*Rtemp,axis = -
    norm(cross(R_Lower_Arm.axis,vector(1,0,0))))
```

```
202.    L_Lower_Arm.pos = L_Elbow.pos
203.    L_Lower_Arm.axis = norm(L_Upper_Arm.axis)*Lower_Arm_Length
204.    if L_shoulder <= 0 :  #Parche para ambiguedad de angulo
205.      Ltemp = -1
206.    else:
207.      Ltemp = 1
208.    L_Lower_Arm.axis = rotate(L_Lower_Arm.axis,angle=+
    (frame_1.Left_Elbow.GetValue()/180.)*pi*Ltemp,axis= norm(cross(L_Lower_Arm.axis,ve
    ctor(1,0,0))))
209.    R_Hand.pos = R_Lower_Arm.pos +
    norm(R_Lower_Arm.axis)*(Hand_Radius+Lower_Arm_Length) #Conectar mano
210.    L_Hand.pos = L_Lower_Arm.pos +
    norm(L_Lower_Arm.axis)*(Hand_Radius+Lower_Arm_Length)
211.
212.# Asociamos eventos a funciones
213.def bindControls():
214.    frame_1.Comm_Connect.Bind(wx.EVT_BUTTON, serialConnect)
215.    frame_1.Left_Arm_Pivot.Bind(wx.EVT_SCROLL,onScroll)
216.    frame_1.Right_Arm_Pivot.Bind(wx.EVT_SCROLL,onScroll)
217.    frame_1.Left_Shoulder.Bind(wx.EVT_SCROLL,onScroll)
218.    frame_1.Right_Shoulder.Bind(wx.EVT_SCROLL,onScroll)
219.    frame_1.Left_Elbow.Bind(wx.EVT_SCROLL,onScroll)
220.    frame_1.Right_Elbow.Bind(wx.EVT_SCROLL,onScroll)
221.
222.
223.# Funcion para atender eventos
224.def onScroll(event):
225.    joint_angle = event.GetPosition()
226.    joint = event.GetId()
227.
    update_Torso(frame_1.Right_Arm_Pivot.GetValue(),frame_1.Left_Arm_Pivot.GetValue(),
    frame_1.Right_Shoulder.GetValue(),\
228.
    frame_1.Left_Shoulder.GetValue(),frame_1.Right_Elbow.GetValue(),frame_1.Left_Elbow
    .GetValue())
229.
230.# Funcion para conectar puerto serie
231.def serialConnect(event):
232.    global ser
233.    try:
234.      ser = serial.Serial((int(frame_1.Comm_Port.Value)-1),57600)
235.      print (" Connected to "+str(ser.portstr))
236.    except:
237.      print (" Invalid Comm Port ")
238.
239.
240.
```

```
241.
242.#Address pattern: "/joint"
243.#Type tag: "sifff"
244.#s: Joint name, check out the full list of joints below.
245.#i: The ID of the user.
246.#f1: X coordinate of joint in interval [0.0, 1.0]
247.#f2: Y coordinate of joint in interval [0.0, 1.0]
248.#f3: Z coordinate of joint in interval [0.0, 7.0]
249.
250.
251.# define a message-handler function for the server to call.
252.# "manejador" de los mensajes OSC que lleguen
253.def data_handler(addr, tags, data, source):
254.    # Variables para las coordenadas de las diferentes articulaciones
255.    global rs # Right shoulder, hombro derecho
256.    global ls # Left shoulder, hombro izquierdo
257.    global rh # Right hand, mano derecha
258.    global lh # Left hand, mano izquierda
259.    global rhi # Right hip, cadera derecha
260.    global lhi # Left hip, cadera izquierda
261.    global re # Right elbow, codo derecho
262.    global le # Left elbow, codo izquierdo
263.    global rh_pk # cuenta de actualizaciones de coordenadas de rh "pk = packets"
264.    global lh_pk
265.    global rs_pk
266.    global ls_pk
267.    global rhi_pk
268.    global lhi_pk
269.    global re_pk
270.    global le_pk
271.    # Gestion de mano derecha...
272.    if data[0] == "r_hand":
273.        rh_pk+=1
274.        rh = np.array((data[2], data[3], data[4]))
275.        #print "---"
276.        #print "received new osc msg from %s" % getUrlStr(source)
277.        #print "with addr : %s" % addr
278.        #print "typetags : %s" % tags
279.        #print "the actual data is :%s" % data
280.        #print "---"
281.    if data[0] == "l_hand":
282.        lh_pk+=1
283.        lh = np.array((data[2], data[3], data[4]))
284.    elif data[0] == "r_shoulder":
285.        rs_pk+=1
286.        rs = np.array((data[2], data[3], data[4]))
287.    elif data[0] == "l_shoulder":
```

```
288.     ls_pk+=1
289.     ls = np.array((data[2], data[3], data[4]))
290.   elif data[0] == "r_hip":
291.     rhi_pk+=1
292.     rhi = np.array((data[2], data[3], data[4]))
293.   elif data[0] == "l_hip":
294.     lhi_pk+=1
295.     lhi = np.array((data[2], data[3], data[4]))
296.   elif data[0] == "r_elbow":
297.     re_pk+=1
298.     re = np.array((data[2], data[3], data[4]))
299.   elif data[0] == "l_elbow":
300.     le_pk+=1
301.     le = np.array((data[2], data[3], data[4]))
302.
303.
304.# Transformacion de coordenadas de un sistema a otro
305.def coordsTransform(nombrethread, dummyvar):
306.   global rs
307.   global rh
308.   global lh
309.   global ls
310.   global rhi
311.   global lhi
312.   global re
313.   global le
314.   global rh_pk
315.   global lh_pk
316.   global rs_pk
317.   global ls_pk
318.   global rhi_pk
319.   global lhi_pk
320.   global re_pk
321.   global le_pk
322.
323.   starttracking = False
324.   # bucle de actualizacion de coordenadas tratadas
325.   while 1:
326.
327.
    if starttracking == False and rh_pk > 0 and lh_pk > 0 and rs_pk > 0 and ls_pk > 0
  and rhi_pk > 0and lhi_pk > 0 and re_pk > 0 and le_pk > 0:
328.       starttracking = True
329.     elif starttracking == True:
330.       # nos copiamos las coordenadas
331.       hi = ls
332.       hd = rs
```

```
333.        ci = lhi
334.        cd = rhi
335.        mi = lh
336.        md = rh
337.
338.        # creamos los vectores iniciales del eje de coordenadas
339.        e1 = np.array((1,0,0))
340.        e2 = np.array((0,1,0))
341.        e3 = np.array((0,0,1))
342.
343.        # sacamos los 3 vectores del nuevo eje de coordenadas
344.        # el primer vector es directo, eje X
345.        v1 = hi-hd
346.        vtemp = hi - ci
347.        v1normal = v1 / modulus_vector(v1)
348.        vtempnorm = vtemp / modulus_vector(vtemp)
349.        c_ = ci + (float(np.dot(vtemp,v1)) / float(np.dot(v1,v1))) * v1
350.        # c_ es el punto del codo en proyeccion a la paralela del eje X a la misma
    distancia
351.        # de el eje X a la que estaba el punto del codo original (para formar el
    eje Y)
352.        v2 = hi - c_
353.        # el eje Z se saca con la normal
354.        v3 = np.cross(v1,v2)
355.
356.
357.        # normalizo los vectores
358.        v1 = v1 / modulus_vector(v1)
359.        v2 = v2 / modulus_vector(v2)
360.        v3 = v3 / modulus_vector(v3)
361.
362.        # saco los cosenos para la matriz a resolver
363.        a = cosangle(e1,v1)
364.        b = cosangle(e2,v1)
365.        c = cosangle(e3,v1)
366.        d = cosangle(e1,v2)
367.        e = cosangle(e2,v2)
368.        f = cosangle(e3,v2)
369.        g = cosangle(e1,v3)
370.        h = cosangle(e2,v3)
371.        i = cosangle(e3,v3)
372.
373.        # creo la matriz
374.        x = np.array((a,d,g))
375.        y = np.array((b,e,h))
376.        z = np.array((c,f,i))
377.        A = np.array((x,y,z))
```

```
378.
379.        #Resuelvo el sistema para los diferentes puntos que pasar al nuevo eje de
    coordenadas
380.        try:
381.          solmi = np.linalg.solve(A,mi-hi) # solucion mano izquierda
382.          solmd = np.linalg.solve(A,md-hi) # solucion mano derecha
383.          solcodoizq = np.linalg.solve(A,le-hi) # solucion codo izquierdo
384.          solcododer = np.linalg.solve(A,re-hi) # solucion codo derecho
385.          solhd = np.linalg.solve(A,hd-hi)  # solucion hombro derecho
386.        except np.linalg.LinAlgError:
387.          print "LinAlgError! Singular matrix, cant compute"
388.
389.
390.        # Toca escalar los puntos a las coordenadas del sistema de la simulacion
391.
392.        # Sacamos las distancias del brazo y antebrazo para ambos sistemas, kinect
    y bioloid
393.        d_brazo = dist(np.array((0,0,0)),solcodoizq)
394.        d_antebrazo = dist(solcodoizq,solmi)
395.        d_bioloid_brazo = 0.09
396.        d_bioloid_antebrazo = 0.043
397.        d_todo_kinect = d_brazo + d_antebrazo
398.        d_todo_bioloid = d_bioloid_brazo + d_bioloid_antebrazo
399.
400.
401.        # Escalamos el punto
402.        # Partiendo de codo izquierdo, con lo cual 0 0 0, faltaria restar para el
    otro codo el hombro
403.        v_hi_a_ci = (solcodoizq  / modulus_vector(solcodoizq) ) * d_bioloid_brazo
404.        v_ci_a_mi = (solmi - solcodoizq)
405.        v_ci_a_mi = (v_ci_a_mi /modulus_vector(v_ci_a_mi) ) * d_bioloid_antebrazo
406.        # Trasladamos teniendo en cuenta el centro del sistema de coordenadas (lo
    del + (0.0...) )
407.        puntomanoizq.pos = v_hi_a_ci + v_ci_a_mi + (0.05, 0.05, 0.025)
408.
409.        # Derecha
410.        v_hd_a_cd = (solcododer - solhd)
411.        v_hd_a_cd = (v_hd_a_cd  / modulus_vector(v_hd_a_cd) ) * d_bioloid_brazo
412.        v_cd_a_md = (solmd - solcododer)
413.        v_cd_a_md = (v_cd_a_md /modulus_vector(v_cd_a_md) ) * d_bioloid_antebrazo
414.        puntomanoder.pos = v_hd_a_cd + v_cd_a_md + (-0.05, 0.05, 0.025)
415.
416.        # Calcular cinematicas inversas para cada mano
417.        do_IK(0, puntomanoizq.pos)
418.        do_IK(1, puntomanoder.pos)
419.
420.
```

```
421.# Main de la aplicacion, parcialmente generado por wxPython
422.#########################
423.# Start the wx application handler
424.#########################
425.if __name__ == "__main__":
426.    # Iniciar GUI
427.    app = wx.PySimpleApp(0)
428.    wx.InitAllImageHandlers()
429.    frame_1 = MyFrame(None, -1, "")
430.    bindControls()
431.
432.    # Creamos variables globales (notadas en data_handler)
433.    rh_pk = 0
434.    lh_pk = 0
435.    rs_pk = 0
436.    ls_pk = 0
437.    rhi_pk = 0
438.    lhi_pk = 0
439.    re_pk = 0
440.    le_pk = 0
441.
442.    rs = np.array((0.0, 0.0, 0.0))
443.    rh = np.array((0.0, 0.0, 0.0))
444.    lh = np.array((0.0, 0.0, 0.0))
445.    ls = np.array((0.0, 0.0, 0.0))
446.    rhi = np.array((0.0, 0.0, 0.0))
447.    lhi = np.array((0.0, 0.0, 0.0))
448.    re = np.array((0.0, 0.0, 0.0))
449.    le = np.array((0.0, 0.0, 0.0))
450.
451.    # xyz vectores
452.    vx = np.array((0.0, 0.0, 0.0))
453.    vy = np.array((0.0, 0.0, 0.0))
454.    vz = np.array((0.0, 0.0, 0.0))
455.
456.
457.    # Iniciamos el recolector de los mensajes OSC
458.    initOSCServer('127.0.0.1', 7110)
459.    setOSCHandler("/joint", data_handler) # anyadimos nuestra funcion como data
       handler
460.    print "OSC recolector inicializado"
461.    dummyvar = 0
462.    try:
463.        thread.start_new_thread(coordsTransform, ("coordsTransform
       Thread", dummyvar))
464.    except:
465.        print sys.exc_info()
```

```
466.
467.  print "CoordsTransform thread iniciado"
468.  # Puntos reales pintados en blanco
469.  puntomanoizq = points(pos=(0.1, 0.1, 0.1), size = 40, color=color.white)
470.  puntomanoder = points(pos=(-0.1, 0.1, 0.1), size = 40, color=color.white)
471.
      ball = sphere(pos=L_Shoulder_Joint.pos, radius=Upper_Arm_Length+Lower_Arm_Length,
      opacity=0.1)
472.
      ball = sphere(pos=R_Shoulder_Joint.pos, radius=Upper_Arm_Length+Lower_Arm_Length,
      opacity=0.1)
473.
474.  # Acabar de iniciar GUI
475.  app.SetTopWindow(frame_1)
476.  frame_1.Show()
477.  app.MainLoop()
478.
479.  # Terminar bien thread
480.  try:
481.    thread.join()
482.  except:
483.    print sys.exc_info()
```

12.4 Anexo código declaración estructura simulación Bioloid

```
from visual import *
1.
2. # Parametros Torso
3.
4. C = 0
5. D = 0
6. drag = False
7. Upper_Arm_Length = .09
8. Lower_Arm_Length = .043
9. Hand_Radius = 0.0275
10.shoulder_pivot_angle = 0
11.shoulder_angle = 0
12.elbow_angle = 0
13.arm_radius = 0.02
14.
15.torso_x = .1
16.torso_y = .2
17.torso_z = .1
18.
```

```
19.# Crear graficos del torso
20.Torso = box(size = (torso_x,torso_y,torso_z),color = color.green)
21.R_Shoulder_Joint = sphere(pos = (Torso.size.x/2,Torso.size.y/4,Torso.size.z/4),rad
   ius = 0.03,color =color.red)
22.L_Shoulder_Joint = sphere(pos = (-
   Torso.size.x/2,Torso.size.y/4,Torso.size.z/4),radius = 0.03,color =color.red)
23.R_Upper_Arm = cylinder(pos = R_Shoulder_Joint.pos,axis = (Upper_Arm_Length,0,0),ra
   dius =arm_radius,color = color.blue)
24.L_Upper_Arm = cylinder(pos = L_Shoulder_Joint.pos,axis = (-
   Upper_Arm_Length,0,0),radius =arm_radius,color = color.blue)
25.R_Elbow = sphere(pos = R_Upper_Arm.pos +
   R_Upper_Arm.axis,radius = arm_radius,color = color.red)
26.L_Elbow = sphere(pos = L_Upper_Arm.pos +
   L_Upper_Arm.axis,radius = arm_radius,color = color.red)
27.R_Lower_Arm = cylinder(pos = R_Elbow.pos,axis = (Lower_Arm_Length,0,0),radius = ar
   m_radius,color =color.blue)
28.L_Lower_Arm = cylinder(pos = L_Elbow.pos,axis = (-Lower_Arm_Length,0,0),radius = a
   rm_radius,color =color.blue)
29.R_Hand = sphere(pos = R_Elbow.pos+R_Lower_Arm.axis,radius = Hand_Radius,color = co
   lor.red,make_trail=True, interval=10, retain=10)
30.R_Hand.trail_object.color = color.orange
31.L_Hand = sphere(pos = L_Elbow.pos+L_Lower_Arm.axis,radius = Hand_Radius,color = co
   lor.red,make_trail=True, interval=10, retain=10)
32.L_Hand.trail_object.color = color.orange
33.
34.# Crear esfera y vector para el error
35.rvector = arrow(pos = (0,0,0),color=color.orange, axis=(0,0,0.2))
36.target = sphere(pos = rvector.pos,color = color.orange,radius = 0.02)
```

12.5 Anexo código declaración del GUI

```
#!/usr/bin/env python
1. # -*- coding: iso-8859-15 -*-
2. # generated by wxGlade 0.6.3
3.
4. import wx
5. # begin wxGlade: extracode
6. # end wxGlade
7.
8.
9.
10.class MyFrame(wx.Frame):
11.   def __init__(self, *args, **kwds):
```

```
12.     # begin wxGlade: MyFrame.__init__
13.
14.     kwds["style"] = wx.DEFAULT_FRAME_STYLE
15.     wx.Frame.__init__(self, *args, **kwds)
16.     self.Comm_Port = wx.SpinCtrl(self, -1, "4", min=0, max=100)
17.     self.Comm_Connect = wx.Button(self, -1, "Connect")
18.     self.Left_Arm_Pivot = wx.Slider(self, 13, 0, -
    140, 140, style=wx.SL_HORIZONTAL|wx.SL_LABELS)
19.     self.Right_Arm_Pivot = wx.Slider(self, 10, 0, -
    140, 140, style=wx.SL_HORIZONTAL|wx.SL_LABELS)
20.     self.label_1 = wx.StaticText(self, -1, "Sliders")
21.     self.Left_Shoulder = wx.Slider(self, 14, 0, -20, 120, style=wx.SL_HORIZONTAL|
    wx.SL_LABELS)
22.     self.Right_Shoulder = wx.Slider(self, 11, 0, -20, 120, style=wx.SL_HORIZONTAL|
    wx.SL_LABELS)
23.     self.label_2 = wx.StaticText(self, -1, "Sliders")
24.     self.Left_Elbow = wx.Slider(self, 15, 0, -20, 100, style=wx.SL_HORIZONTAL|
    wx.SL_LABELS)
25.     self.Right_Elbow = wx.Slider(self, 12,0, -20, 100, style=wx.SL_HORIZONTAL|
    wx.SL_LABELS)
26.
27.     self.__set_properties()
28.     self.__do_layout()
29.     # end wxGlade
30.
31.   def __set_properties(self):
32.     # begin wxGlade: MyFrame.__set_properties
33.     self.SetTitle("Bioloid Arms")
34.     # end wxGlade
35.
36.   def __do_layout(self):
37.     # begin wxGlade: MyFrame.__do_layout
38.     grid_sizer_1 = wx.GridSizer(3, 3, 1, 3)
39.     sizer_2 = wx.BoxSizer(wx.HORIZONTAL)
40.     sizer_2.Add(self.Comm_Port, 0, 0, 0)
41.     sizer_2.Add(self.Comm_Connect, 0, 0, 0)
42.     grid_sizer_1.Add(sizer_2, 1, wx.EXPAND, 0)
43.     grid_sizer_1.Add(self.Left_Arm_Pivot, 0, wx.EXPAND, 0)
44.     grid_sizer_1.Add(self.Right_Arm_Pivot, 0, wx.EXPAND, 0)
45.     grid_sizer_1.Add(self.label_1, 0, wx.ALIGN_CENTER_HORIZONTAL, 0)
46.     grid_sizer_1.Add(self.Left_Shoulder, 0, wx.EXPAND, 0)
47.     grid_sizer_1.Add(self.Right_Shoulder, 0, wx.EXPAND, 0)
48.     grid_sizer_1.Add(self.label_2, 0, wx.ALIGN_CENTER_HORIZONTAL, 0)
49.     grid_sizer_1.Add(self.Left_Elbow, 0, wx.EXPAND, 0)
50.     grid_sizer_1.Add(self.Right_Elbow, 0, wx.EXPAND, 0)
51.     self.SetSizer(grid_sizer_1)
52.     grid_sizer_1.Fit(self)
```

```
53.     self.Layout()
54.     # end wxGlade
55. 
56. 
57. 
58.# end of class MyFrame
```