
UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Ingeniería Informática

Proyecto Fin de Carrera

# Detección de Intenciones Humanas con el uso de RGBD como Mecanismo de Sensorización

Daniel González Bodón

Julio, 2014


UNIVERSIDAD DE EXTREMADURA
Escuela Politécnica
Ingeniería Informática

# Proyecto Fin de Carrera
# Detección de Intenciones Humanas con el uso de RGBD como Mecanismo de Sensorización

Autor: Daniel González Bodón
Fdo:

Director: José Moreno del Pozo
Fdo:

**Tribunal Calificador**

Presidente: Bachiller Burgos, Pilar
Fdo:

Secretario: Andrés Hernández, Francisco M.
Fdo:

Vocal: Núñez Trujillo, Pedro
Fdo:

CALIFICACIÓN:
FECHA:

# ÍNDICE

# AGRADECIMIENTOS

*A Pepe por enseñarme el camino de la visión artificial y darme la oportunidad de conocer RoboLab y su gente.*

*A Luiky, por estar siempre ahí, al pie del cañón en todo. Por su atención, paciencia, cercanía y buen rollo.*

*A mis amigos de la Universidad, los que duran para siempre: Antonio, Iván, Enrique, Paco, Forlán, Yanfri, Walter, Josiche, Rulo… por haber formado parte de unos años inolvidables.*

*Y sobre todo a mi familia, a mis padres Rafa y Montse, mis niñas Irene y Coral, y a mis abuelos, los que están y los que no. Por su apoyo y paciencia incondicionales en los mejores y peores momentos.*

# 1. INTRODUCCIÓN

En el mundo de la robótica la visión artificial es uno de los campos más investigados y desarrollados, ya que la vista es uno de los sentidos más importantes y de los que más información aportan. Y es por ello, que en el laboratorio de Robótica de la Universidad de Extremadura, RoboLab, llevan varios años investigando sobre el tema.

Este trabajo ha dado sus frutos, se ha conseguido la navegación autónoma de robots esquivando los objetos que se encuentren a su paso, reconocer elementos de su entorno, enfocar, acercarse, seguir trayectorias… en definitiva, se ha dotado a los robots de una cierta solvencia para interactuar con el entorno que le rodea.

Por otro lado, la visión artificial ha sufrido una revolución en los últimos tiempos con la aparición de las cámaras RGBD, que ofrecen una gran mejora en la relación calidad-precio respecto a los sistemas usados anteriormente, como la visión estereoscópica, en su capacidad para inferir la posición tridimensional de los puntos de la imagen . Estos dispositivos, junto con las librerías de código abierto que se han ido desarrollando, permiten entre otras cosas la detección de personas y la extracción de información espacial de sus articulaciones, todo ello en tiempo real y de una manera eficiente. Esto incorporado a los robots consigue que su percepción del mundo sea mucho más amplia. Si bien es cierto que antes de la aparición de las cámaras RGBD se conseguía la detección de personas, y que algunas técnicas se han extrapolado, no se hacía de una manera tan eficiente, sencilla y versátil.

El ideal de robot es aquel que interactúa de una forma natural con el entorno y los objetos que le rodean  y se relaciona de igual a igual con las personas que encuentra a su alrededor. Hasta llegar a este punto queda un largo camino por recorrer, pues aunque se mueva por el entorno, y pueda detectar personas, no actúa de una forma coherente o natural por sí mismo. Para seguir avanzando hacia este

ideal, es necesario comenzar a dotar a los robots de una cierta conciencia que le permita tomar sus propias decisiones ante una situación concreta.

En este punto es donde entra nuestro proyecto. Para actuar de una manera coherente, el robot necesita entender la escena que ocurre ante él. Nuestro trabajo va a consistir en detectar las intenciones de las personas, para de esta manera tener una cierta información que le ayude a decidir. Está claro que la meta es lejana, muy pretenciosa e inalcanzable para un proyecto final de carrera, pero sentaremos las bases de éste análisis, centrándonos en la expresión corporal. Para ello realizaremos la detección de poses habituales que se dan en situaciones concretas. Esta detección abrirá un abanico de posibilidades, pudiendo utilizarse, por ejemplo, para la comunicación hombre-robot mediante gestos o en tareas de rehabilitación motora donde la adherencia al ejercicio es crucial, y donde una relación humano-robot más natural es imprescindible.

El proyecto se realizará mediante el framework de robótica RoboComp, desarrollado por RoboLab en la Universidad de Extremadura. Consistirá en la realización de dos componentes: OpenNI2Comp se encargará de la detección de personas a partir de una cámara RGBD, envolviendo las librerías para NUI (Natural User Interface) OpenNI2 y NITE2. Y El segundo componente PoseDetectorComp hará las veces de conciencia del robot, detectando la postura a partir de los datos ofrecidos por OpenNI2Comp, y comunicando la intención que de ella deriva. Estos componentes serán añadidos al repositorio de RoboComp bajo licencia GPL.

## 1.1. MOTIVACIÓN

La idea de la Interfaz Natural de Usuario, está cada día más presente en la tecnología que nos rodea, ya que ésta va evolucionando hacia una forma más natural e intuitiva. Está cada día más arraigada a nuestra forma de vida, hasta llegar a convertirse en algo esencial, y su aprendizaje es cada vez, una barrera menor. Esta gran aceptación es, en parte, gracias a que la tecnología comienza a comportarse como nosotros lo hacemos.

La comunicación con las máquinas debería ser análoga a la comunicación con otro ser humano. Por ello, es importante conseguir que los canales en los que se transmita la información sea el mismo que se usa entre personas. Estos son, principalmente el lenguaje tanto hablado como corporal. Conseguir que un hombre o una mujer y una máquina se comunicaran de esta manera sería un paso muy importante para el desarrollo. Las cámaras RGBD son un gran apoyo para lograr la transparencia buscada, pues permiten la interacción con la tecnología siendo el cuerpo del usuario el medio de comunicación.

A título personal, desde que vi por primera vez un personaje moverse como una persona lo hacía o mover objetos en una pantalla sin tocarla y simplemente siendo grabado con una cámara RGBD, mi curiosidad sobre estos temas creció, me pareció algo verdaderamente futurista. Quería utilizarlo, y entender cómo funcionaba. Este proyecto además de esto, me dio la oportunidad de desarrollar algo propio con este dispositivo.

En cuanto a la forma de enfocar el proyecto con cámaras RGBD, me pareció un reto personal y profesional el trabajar con un framework tan grande y diferente de lo que había visto en la carrera como es RoboComp. También es una motivación extra el tener que hacer un sistema que analice directamente el mundo real, con todas las imperfecciones que lo componen.

Posteriormente me di cuenta de que la robótica es de los campos de la Informática más complicados, y a la vez más bonitos para un Ingeniero de Software.

En robótica, a diferencia de la mayoría de software convencional, se debe manejar la incertidumbre. En muchas ramas del software, y sobre todo en el desarrollado en la carrera, este se produce sobre un espacio discreto, sí o no. Un click en un botón siempre va a ser cierto o falso, ya que se implementa sobre un entorno controlado. Cualquier evento puede ser previsto y siempre tendrá el mismo resultado.

En cambio, en la robótica se trabaja sobre el mundo real, con entornos abiertos no controlados y los sensores siempre recibirán algún tipo de ruido, pero sobre ellos se espera una percepción correcta. En la mayoría de los casos el correcto funcionamiento de un sistema no asegura su funcionamiento posterior, pues pueden darse circunstancias inesperadas que previamente no habían ocurrido y no se habían contemplado. Aun con todo esto, se debe dar siempre una respuesta válida y coherente, y encima, en tiempo real.

Otro de los puntos fuertes y más gratificantes de este proyecto, es la idea de aprender conceptos nuevos como: las librerías OpenNI2 y NITE2, el uso y aprendizaje de un lenguaje de marcado como XML como representación del mundo en forma de árbol (InnerModel), la programación orientada a componentes, cinemática directa e inversa… e incluso, otros temas más ligados a la psicología que se estudiarán para la detección de intenciones humanas, como la psicología ecológica o los *affordances*.

En definitiva, este proyecto se perfila como un trabajo muy interesante. En el plano del desarrollo y la innovación por las posibilidades futuras de la idea y el demostrado potencial de las cámaras RGBD; y en el plano personal y profesional por los conocimientos que se adquieren, tanto en software como en temas transversales a la informática, que van desde la psicología hasta las matemáticas, y por los retos que presenta trabajar en robótica y en la interacción con el mundo real, que obligan

a la mente a discurrir de una manera que otros tipos de proyectos no hubieran permitido.

## 1.2. OBJETIVO

Los objetivos principales de este proyecto se pueden resumir en dos hitos, en primer lugar deberemos envolver en el framework de robótica RoboComp desarrollado en la Universidad de Extremadura las librerías OpenNI2 y NiTE2, haciendo que RoboComp disponga de las funcionalidades principales de este software: captación de los datos de color y profundidad de una escena y seguimiento de las personas que en ella aparezcan. Con esta información, el segundo paso será definir en tiempo real las intenciones de estas personas analizando su expresión corporal. Esto no es algo trivial y quedará abierto a futuros desarrollos.

En la vida real, en la mayoría de los casos, no se puede conocer la intención de una persona solamente mirando su expresión corporal, ni siquiera con la base de conocimiento que hemos ido creando con la experiencia. Para llegar a una conclusión aceptable, aunque nunca asegurando su certeza, debemos tener otros factores en cuenta. El tono de voz, el tiempo y el lugar en el que se encuentre, las vivencias pasadas, los gestos faciales, la situación interna de la persona y otros detalles… en definitiva necesitamos de un contexto, que combinado, dan multitud de pistas sobre qué puede pretender, y a menudo nos equivocamos al interpretar las intenciones.

Puesto que captar todas estas circunstancias y posteriormente llegar a una conclusión es hoy por hoy muy difícil, tecnológicamente hablando. En este proyecto se pretende asentar una base sobre la que seguir investigando acerca de la relación entre expresión corporal e intenciones.

Para ello, contaremos con una serie de posturas predefinidas y cada una definirá una intención. El cometido de este proyecto será detectar si una persona está en una postura conocida e informar de qué intención representa.

El sistema que desarrollaremos constará de dos componentes, que entrarán a formar parte del repositorio de RoboComp.

- El primero de ellos envolverá las librerías y las adaptará a este framework. Además y como veremos posteriormente los datos que ofrece NITE2 no son válidos en RoboComp, y deberemos transformarlos para que sean de utilidad, en este y otros proyectos.

- El segundo, a partir de los datos que el primer componente le proporcione, analizará la escena para llegar a alguna conclusión. Además se comunicará con el simulador de RoboComp para mostrar gráficamente los datos del seguimiento de las personas. Es decir tendremos una representación gráfica de la percepción del robot.

Llegar a esto requerirá un desarrollo que lleva implícito una serie de objetivos, secundarios o paralelos, pero necesarios para la consecución de nuestros fines:

- Adquirir conocimiento sobre el mundo de la visión artificial, las cámaras RGBD y la detección de personas.
- Investigar sobre la psicología y la percepción humana, y las teorías y corrientes ideológicas que de ellas surgen, como la cognición clásica o la psicología ecológica.
- Entender el paradigma de programación con el que se va a trabajar: la programación orientada a componentes, que permite un mayor nivel de abstracción que la programación orientada a objetos, a la que estábamos acostumbrados.

- Trabajar con programación en tiempo real, lo que obliga a desarrollar código lo más eficiente posible.
- Trabajar con programación distribuida y paralela, con las que se hace necesario el manejo de hilos de ejecución, la comunicación entre componentes, etc.
- Conocer el framework RoboComp, formado por varias tecnologías y herramientas que necesitaremos para el desarrollo del proyecto, deberemos:
    - Adquirir conocimientos sobre la herramienta CMake para la construcción de componentes e integración de librerías de terceros.
    - Aprender el funcionamiento de la herramienta DSLEditor, con la que generaremos nuestros componentes.
    - Entender y usar el middleware de comunicación ICE, para poder implementar y definir las interfaces que más se adapten a nuestro proyecto, y que a la vez sean versátiles para su futuro uso.
    - Comprender la librería propia de RoboComp, llamada InnerModel y su representación del mundo, aprendiendo conceptos como árboles cinemáticos, matrices de transformación, cinemática directa e inversa...
- Utilizar conocimientos de programación adquiridos en la carrera para aplicarlos a problemas reales, como la estructura de árboles y sus recorridos, estructuras de datos dinámicas, mapas, funciones hash, aritmética de punteros, semáforos, desarrollo de interfaces gráficas, etc...

La búsqueda de un resultado final robusto junto con los objetivos específicos expuestos dan sentido a la realización de este trabajo, puesto que son los que logran el enriquecimiento personal y profesional que aporta, a un licenciado, la realización de su proyecto final de carrera.

# 2. CÁMARAS RGBD

Una cámara RGBD es un dispositivo que toma información 3D del entorno. Está formada por dos cámaras y un láser infrarrojo:

- Cámara RGB: Cámara convencional que obtiene la información de color del plano que observa.

- Láser infrarrojo: El láser infrarrojo del dispositivo proyecta un patrón de puntos sobre los la escena que tiene delante, estos rayos chocan contra los objetos que encuentra y vuelven hacia la cámara, llegando cada haz en un momento determinado, creando un patrón de profundidad.

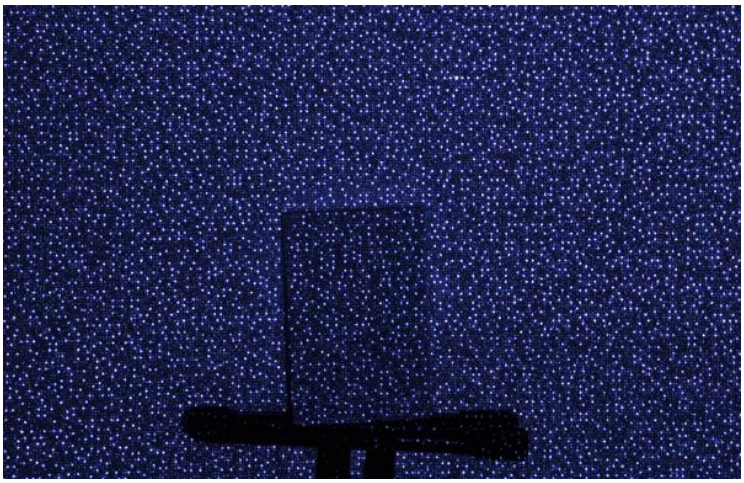

*Figura 1: Nube de puntos proyectado por Kinect*

Cámara Infrarroja: La cámara infrarroja recoge el patrón y por hardware se calcula la profundidad de cada punto. Para ello, la cámara tiene un patrón de puntos memorizado para una profundidad conocida. Al poner objetos delante, el patrón aparece “desplazado”, con este desplazamiento se calcula la profundidad.

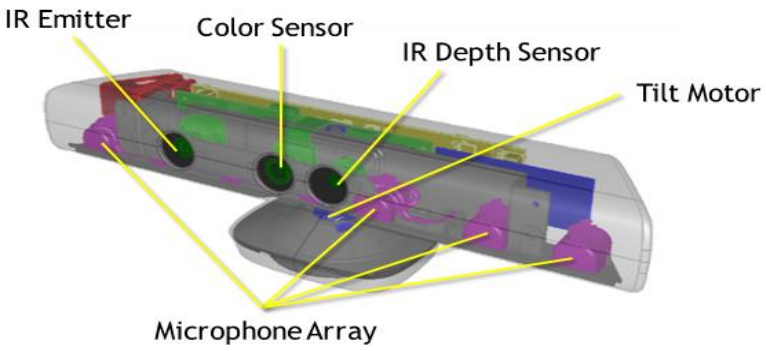


*Figura 2: Partes de una cámara RGBD*

Con estos dos tipos de datos la cámara RGBD asocia a cada punto tomado de la imagen RGB, información de profundidad. Los sensores RGBD son muy útiles para cartografía en 3D, localización, reconocimiento de objetos, seguimiento de personas… y por lo tanto son y serán importantes para el desarrollo de la robótica.

A continuación se tratarán las opciones de cámaras RGBD más importantes que ofrece el mercado.

## 2.1. KINECT DE MICROSOFT

Es un dispositivo RGBD creado por Alex Kipman y desarrollado por Microsoft, originariamente para la videoconsola Xbox360, y posteriormente para PC, para las plataformas Windows 7 y Windows 8. Fue pensado para videojuegos, con el fin de competir con los sistemas Wiimote con WiiMotionPlus y PlayStationMove. Cabe destacar que estos sistemas necesitan de mando y reconocen gestos concretos. Kinect reconoce el cuerpo entero y permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con ella, mediante una interfaz natural de usuario. También reconoce gestos, comandos de voz y objetos e imágenes. Con el origen de Kinect se anunció “el nacimiento de la próxima generación de entretenimiento en el hogar”.


*Figura 3: Kinect 360 y Kinect v2 de Microsoft*

## 2.2. ASUS XTION PRO

Es la alternativa de cámara RGBD de ASUS al Kinect de Microsoft. Además de este producto tiene el modelo XTION PRO, que tiene únicamente una cámara de profundidad, obviando la de color, ya que no es necesaria para la detección humana. Aunque si es necesaria para por ejemplo, hacer mapas tridimensionales de un lugar.


*Figura 4: Asus Xtion Pro*

Los modelos ASUS XTION están especialmente diseñados para desarrolladores. Utiliza sensores infrarrojos, tecnología de detección de la profundidad adaptativa, sensores de imágenes en color y flujo de audio, para capturar en tiempo real a los usuarios, sus gestos y la voz. El kit de desarrollo es completamente abierto, y tiene muchas facilidades para crear software. Además es compatible con el motor de juego Unity3D. Es el dispositivo que utilizaremos para este proyecto

## 2.3. COMPARATIVA

| CARACTERÍSTICA | KINECT XBOX | KINECT 2.0 | KINECT FOR WINDOWS v2 | ASUS XTION PRO LIVE |
|---|---|---|---|---|
| **Campo de visión del sensor IR(FOV)** | 57º horizontal por 43º vertical | 57,5º horizontal por 43,5º vertical | 70º horizontal por 60º vertical | 58º horizontal por 45º vertical |
| **Rango de profundidad** | 1,2 m - 3,5 m | 0,8m - 4,0m | 0,5m - 4,5m | 0,8m - 3,5m |
| **Resolución flujo de color** | 640x480 a 32 bits de color | 640x480 a 32 bits de color | 1920x1080 | 640x480 a 32 bits de color |
| **Resolución flujo de profundidad** | 320x240 a 16 bits de profundidad | 320x240 a 13 bits de profundidad | 1920x1080 | 320x240 a 16 bits de profundidad |
| **Latencia** | 102 ms | 90 ms | 20 ms | -- |
| **Motor de inclinación** | Sí | Sí | No | No |
| **Interfaz** | Única para Xbox360 | USB 2.0 | USB 3.0 | USB 2.0 |
| **Sonido** | 4 micro-array, 16 KHz | 4 micro-array, 16 KHz | Array de 4 micrófonos | 2 micrófonos |

## 2.4. IMPACTO EN LA ROBÓTICA ACTUAL

Antes de la aparición de las cámaras RGBD, la implementación de las funcionalidades que éstas desempeñan no era tarea sencilla, o era muy costosa. Se utilizaban dispositivos como láseres 3D o cámaras infrarrojas, cuyos precios eran muy altos debido a que no se fabricaban en masa. Por ello, la forma más utilizada y económica de detectar la distancia de los objetos era la visión estereoscópica. Ésta técnica consistía en utilizar dos cámaras previamente calibradas, debían estar colocadas exactamente igual en dos de los ejes de coordenadas, y separadas entre ellas una distancia conocida, y apuntando hacia el mismo sitio. Ambas cámaras detectaban la misma imagen pero con los objetos ligeramente cambiados de sitio. Esta desviación es la que se utilizaba para detectar a qué distancia se encontraba. En puntos posteriores se explica más a fondo esta técnica, cómo se detectaban los mismos objetos en dos imágenes distintas y cómo se calcula su distancia.

*Figura 5: Sistema estereoscópico*

Dadas las dificultades de detectar la profundidad de una imagen con las tecnologías que previamente se utilizaban, las cámaras RGBD han tenido una gran repercusión en el mundo de la robótica, tanto por su sencillez como por su precio, ya que son la primera alternativa económica al hardware utilizado anteriormente, que dejaba fuera del alcance del público en general la mayoría de proyectos o propósitos de éste ámbito. Además, la liberación de librerías (como FreeNect, OpenNI, PCL…)

que controlan estos dispositivos ha disparado la cantidad de desarrolladores interesados en este campo

La oportunidad que ha dado tener al alcance de todos tanto el hardware como el software necesario ha hecho que se estén desarrollando multitud de proyectos, y que se haya creado una comunidad muy amplia sobre el tema.

Para comprobar cómo ha influido esto en la robótica, se citan algunos ejemplos de avances importantes que se han logrado gracias a las cámaras RGBD:

- PR2 (Massachusetts Institute of Technology, MIT): PR2 es un robot capaz de desplazarse con naturalidad por el entorno, tanto al aire libre como en lugares cerrados. Los robots siempre han requerido tener una referencia de distancias en su conocimiento para permitirles desplazarse por determinado lugar. Hasta ahora eso se podía hacer con información pregrabada o con un movimiento perimetral con el que se calculan las distancias aplicando algún tipo de técnica que se comentarán en puntos posteriores, pero estas alternativas no permitían que el robot reaccionara bien ante la aparición de obstáculos.

  Gracias a las cámaras RGBD ahora pueden usar la información de profundidad de todo lo que tengan a su alrededor, utilizando lo que se conoce como SLAM, por su siglas en inglés, *Simultaneous Localization and Mapping*, sistema mediante el cual automáticamente el robot escribirá sobre la información del entorno a medida que nuevas referencias aparezcan. Esta técnica, ya se utilizaba con los perceptores previos a las cámaras RGBD pero es innegable que estos consiguen abaratar el proceso.


*Figura 6: PR2*

- GIST, Gestural Interface for Remote Spatial Perception (Instituto Tecnológico de Massachusetts y Universidad de Nevada): La idea consiste en que el dispositivo, equipado con una cámara RGBD, proyecte información sobre el mundo físico y permita al usuario interactuar con él moviendo la mano, señalando y realizando otro tipo de gestos. De esta manera, el GIST recopila los datos en respuesta a gestos de la mano como forma de incrementar la reducida percepción espacial de las personas con discapacidad visual.

  Por ejemplo, si una persona lleva puesto el GIST y realiza una señal "V" con los dedos, el dispositivo detecta el color dominante en el área ocupada por los dedos. Si el usuario extiende el puño cerrado, el sistema es capaz de identificar si hay una persona en esa dirección y a qué distancia se encuentra.

  Además el sistema también puede reconocer objetos y rostros y permite la interacción por el habla.

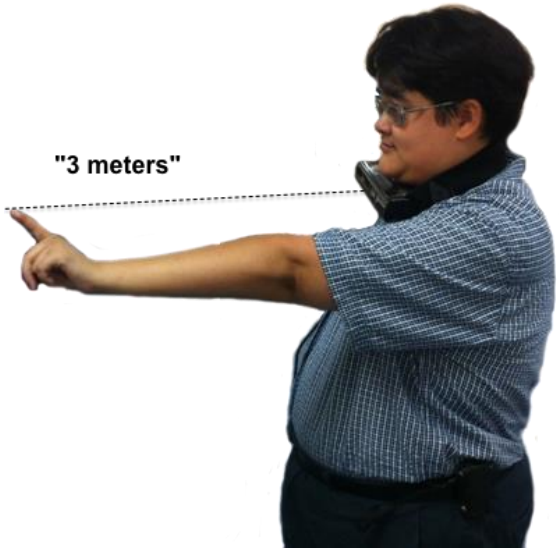


*Figura 7: GIST*

- Cámara de vigilancia avanzada (Corea del Norte): Al igual que muchas tecnologías surgen del lado militar y se extienden hasta llegar a todos, las fuerzas armadas también se aprovechan de las tecnologías desarrolladas para civiles. Es el caso de las cámaras RGBD. El gobierno de Corea del Norte utiliza el dispositivo Kinect de Microsoft como cámara fronteriza militar para vigilar la Zona Desmilitarizada entre las dos Coreas, una amplia zona despoblada que separa Corea del Norte de Corea del Sur. Kinect es capaz de detectar personas y sus gestos, es capaz de distinguir animales de personas, así dejará pasar un perro o un zorro, pero activará la alarma si lo que detecta es una persona. Todo ello sin necesidad de militares que vigilen.

*Figura 8: Zona desmilitarizada del lado de Corea del Norte*

- Soci-Bot Mini: Este robot permite adoptar el rostro de personas e identificar diferentes estados de ánimo. Es utilizado para asesorar a personas en puntos estratégicamente colocados en espacios públicos. El robot dispone de un sensor de profundidad. Con ese nivel de detalle, puede analizar el estado de ánimo y la edad del interlocutor para mantener animadas charlas. El software de conversación es una versión avanzada de Rosette.

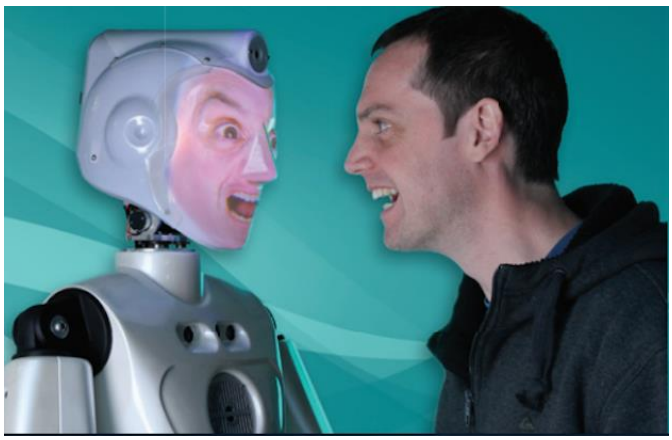

*Figura 9: Soci-bot Mini*

- Estacionamiento de vehículos (Universidad de Sidney): El sistema consiste en una cámara RGBD colocada en la parte posterior del automóvil, que junto a una pantalla táctil dentro del vehículo ayuda al conductor a estacionar.

*Figura 10: Estacionamiento de vehículos con Kinect*

Además de estos proyectos, las cámaras RGBD se están usando por ejemplo, como asistencia a los médicos, rehabilitando pacientes con enfermedades motoras y cognitivas o enfermedades neurológicas o traduciendo el lenguaje de signos.

Como vemos, se está trabajando muy concienzudamente con estos dispositivos, y en ámbitos muy diversos. Los avances están siendo considerables aun siendo una tecnología relativamente joven, por lo que se prevé un crecimiento exponencial tanto de la tecnología de cámaras RGBD como de proyectos innovadores en un futuro próximo.

# 3. SOFTWARE

## 3.1. PROGRAMACIÓN ORIENTADA A COMPONENTES

Dos de los principales problemas que se presentan cuando se crea software son la escalabilidad y la reusabilidad. Estos problemas son especialmente agudos cuando se trata a software que se va a emplear en robótica y es debido a que aprovechar software es algo excepcional en este campo. A pesar de la importancia de la reusabilidad, generalmente se suele perder de vista este aspecto y se acaba creando software monolítico y poco utilizable.

En el ámbito de la robótica es muy común que los investigadores implementen todos los algoritmos con un diseño rígido y orientado a una tarea y/o a un robot específico. De ser así, cuando finaliza la etapa de implementación el software desarrollado acaba siendo imposible de utilizar. Suele estar tan ligado a una plataforma o tarea específica que resulta más práctico empezar de cero (debido a las dependencias y efectos colaterales derivados de su rigidez).

La programación orientada a componentes [1] surge como solución a este tipo de problemas. Es un enfoque que no tiene necesariamente que ver con concurrencia o computación distribuida, sino con cómo se organiza el software. La programación orientada a objetos representó un gran avance respecto a la programación estructurada, sin embargo, cuando el número de clases y sus interdependencias crece, resulta demasiado difícil entender el sistema globalmente. Es por tanto beneficioso disponer de un grado mayor de encapsulamiento, que aúne diferentes clases relacionadas bajo una interfaz única, y permita comprender el sistema con menor grado de detalle. La programación orientada a componentes, que se propuso para solucionar este tipo de problemas, muchos la ven como el siguiente paso tras la programación orientada a objetos. [2]

Un componente es un programa que provee una interfaz que otros programas o componentes pueden utilizar. Es una pieza de código elaborado o a elaborar que codifica una determinada funcionalidad. Son los elementos básicos en la construcción de las aplicaciones en esta arquitectura, que se juntan y combinan para llevar a cabo una tarea concreta. A su vez, estos programas hacen uso de programación orientada a objetos (así como en programación orientada a objetos se hace uso de programación estructurada). Esta división en piezas de software de mayor tamaño que las clases, implementadas como subprogramas independientes ayuda a mitigar los problemas de los que se ha hablado antes y a aislar errores.

Además, desde su carácter intrínsecamente distribuido ayudan a repartir la carga de cómputo entre núcleos, incluso, dependiendo de la tecnología usada, entre diferentes ordenadores en red.

Desde el punto de vista del diseño se pueden ver como una gran clase que ofrece métodos públicos. La única diferencia desde este punto de vista es que la complejidad introducida por las clases de las que depende el componente (o clase) que no son del dominio del problema desaparece porque la interfaz del componente las esconde. Un componente puede ser arbitrariamente complejo, pero un paso atrás, lo único que se ve es la interfaz que ofrece. Esto es lo que lo define como componente.

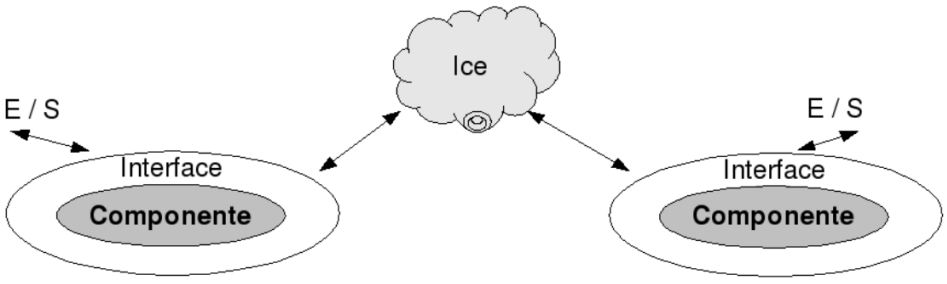


*Figura 11: Representación genérica de componentes*

Por tanto, si cada componente realiza una serie de tareas o responde a una serie de órdenes, necesitamos quien se encargue de esa comunicación. Es donde entra en juego Ice, pieza clave para la comprensión de la arquitectura y para la funcionalidad de la misma. Ice es un framework de comunicación muy interesante para su uso en robótica. A pesar de estar desarrollado por una empresa privada es open-source y está licenciado bajo GPL. Además de su carácter libre hay otras dos razones para elegir Ice.

- La primera razón es su facilidad de uso: al contrario que otras tecnologías como Corba, la filosofía de Ice es soportar sólo aquellas características que los usuarios vayan realmente a necesitar y evitar introducir otras características que raramente se usan y dificultan el aprendizaje.

- La segunda razón es la eficiencia: si bien Ice codifica la comunicación eficientemente tanto en términos de tiempo como de espacio, otras alternativas suelen codificar la comunicación en XML. El uso de XML tiene ventajas en otras aplicaciones donde es conveniente que los humanos puedan entender el tráfico y no sean factores críticos ni la latencia ni la eficiencia, pero dentro de un robot esto no ocurre.

Ice soporta dos tipos de comunicación, por llamada remota (tipo RPC) o por suscripción (mediante un servicio que hace de servidor de mensajes). Sin embargo, este último introduce un paso intermedio cuya latencia hace desaconsejable su uso en robótica donde la ejecución en tiempo real es fundamental.

Para realizar una conexión con un componente lo único que se ha de conocer es su “endpoint”, es decir, la información necesaria para realizar la conexión: la dirección o nombre del ordenador al que se va conectar, el protocolo, el puerto y el nombre de la interfaz. Por ejemplo:

< nombre >:< tcp/udp > −p puerto − h host

Donde 'nombre' es el nombre de la interfaz del componente al que se quiere hacer la conexión, 'puerto' es el puerto tcp o udp en el que el componente esté escuchando y 'host' el nombre del ordenador donde el componente se está ejecutando.

Por otra parte, como alternativa a la llamada anterior, Ice permite una configuración personalizable para cada componente, que se describe en un fichero de texto plano, el cual almacena información sobre los proxys, sobre los componentes con los que se conecta, sobre el propio componente y las interfaces que ofrece, y también incluyen parámetros para configurar la conexión (tamaños de los datos, dirección de la máquina remota, protocolo a utilizar, puerto de conexión…).

Desde el punto de vista del programador, una vez la conexión está hecha, el uso de componentes Ice es extremadamente simple. Cuando se realiza una conexión se crea una instancia de un proxy al componente en forma de objeto.

Al ejecutar un método del componente proxy el framework se encarga automáticamente de redirigir la llamada al componente remoto, por lo que la instancia del proxy se utiliza como si se tratase de una instancia de un objeto con la funcionalidad del componente.

Esta idea de usar un recurso remoto dentro de un programa no es nueva. Antes de la llegada de la programación orientada a objetos, ya había un protocolo en Unix llamado RPC para hacer llamadas remotas a procedimientos. Más tarde surgieron tecnologías como RMI, CORBA, DCOM, o Ice. [3]

Para usar programación orientada a componentes se ha de dividir el diseño del software en piezas que ofrezcan una interfaz. A cambio obtendremos mayor reusabilidad, utilizando los mismos componentes en diferentes contextos, de esta forma se reduce considerablemente el tiempo, el coste y el esfuerzo de desarrollo de nuevas aplicaciones, aumentando a la vez la flexibilidad, reutilización y fiabilidad de las mismas. Será más fácil aislar y encontrar fallos, consiguiendo eliminar la

necesidad de contemplar cientos de clases para comprender el software desarrollado.

## 3.2. ROBOCOMP

RoboComp [4], es un repositorio de componentes basados en Ice con aplicaciones en robótica y visión artificial. RoboComp se comenzó a desarrollar en RoboLab en 2005. Actualmente el proyecto está alojado en SourceForge, donde, además de tener la página del proyecto [5], dispone de un wiki [6] donde hay documentación, y un repositorio al que se puede acceder incluso directamente con un navegador web [7].

Dispone de componentes para captura y visualización de vídeo, manejo de varios robots, que posteriormente comentaremos, detección y mantenimiento de regiones de interés (ROI), lectura y visualización de láser, lectura de joystick y navegación, entre otros muchos. Además dispone de un generador automático de componentes nuevos (DSLEditor), de un programa gráfico de manipulación y control de componentes, managerComp y hasta de un componente encargado de grabar los datos producidos por los sensores para su posterior reproducción, replayComp.

Como dijimos antes, RoboComp ofrece funcionalidades para el manejo de algunos robots. Vamos a comentar algunos de ellos, desarrollados principalmente por RoboLab, para comprobar la versatilidad de este framework:

- Loki: Este robot ha sido construido como colaboración entre varias universidades y empresas españolas, siendo el diseño y la coordinación responsabilidad de la Uex. Este robot ha sido construido como una plataforma de investigación robótica social avanzada.

  La base móvil del robot contiene las baterías, la gestión electrónica de la energía, los controladores locales y una placa Xeon de alta gama. Sobre ella tiene una columna rígida hace de tronco, y sobre ella cuelgan dos brazos y

una cabeza expresiva. Cada brazo tiene siete grados de libertad (DOF) y una construcción antropomórfica con motores schunk. El antebrazo tiene tres DOF, en la muñeca, de seis DOF posee un sensor de fuerza y una mano de cuatro DOF con tres dedos.

La cabeza está conectada al torso con cuatro DOF y tiene dos cámaras orientables, una Asus Xtion y una Fle3. Cuenta con guiño de ojos y cejas para las expresiones faciales, así como de micrófono y altavoz para la síntesis de voz. El número total de grados de libertad es de 37. El robot actualmente realiza un repertorio básico de navegación, manipulación e interacción.

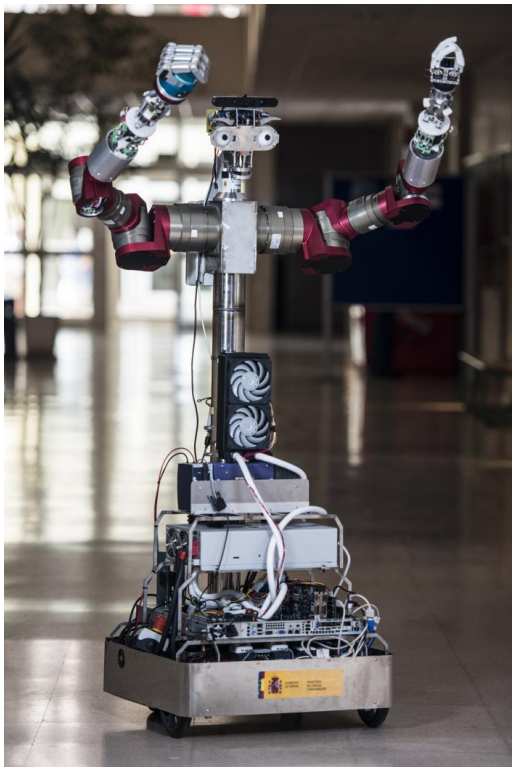

*Figura 12: Loki de RoboLab*

- Thor: Este robot es la última versión desarrollada del robot RobEx, el cual ha sido completamente rediseñado para soportar 150 Kg de peso. Ha sido creado como base de un manipulador móvil de alto rendimiento, con un brazo de 7 DOF, una mano y una cabeza expresiva.

Thor tiene un procesador Cortex embebido que ejecuta Linux y RoboComp. Los componentes básicos que se ejecutan dentro computan información odométrica y sirven de datos de láser y estado de la batería.

*Figura 13: Thor de RoboLab*

- Dulce: Nació como parte de una experiencia de investigación educativa para niños de 5º curso. Ha colaborado con RoboLab el Colegio de Educación Primaria Dulce Chacón. Dulce es una tortuga robótica que ejecuta programas de Logo.

  Durante esta experiencia, los alumnos recibieron clases semanales sobre Logo utilizando la aplicación de software abierto Linux KTurtle. Para posteriormente programar directamente el robot Dulce, ordenando a trazar letras y figuras geométricas en un tablero gigante colocado en el suelo.

*Figura 14: Dulce de RoboLab*

- Ursus 1.0: Este robot se ha construido con el apoyo del Hospital Virgen del Rocío de Sevilla, dentro del proyecto ACROSS. Está diseñado para proponer juegos a los niños con parálisis cerebral con el fin de mejorar su recuperación. También se puede utilizar como una herramienta para sus terapeutas. Para que el robot sea visualmente amigable para los niños, tiene una altura apropiada y se ha envuelto en un tejido que lo hace parecer un oso de peluche. Los pacientes observan la forma de realizar los ejercicios en una pantalla o proyector a través de videos. Dentro de ellos, el robot inyecta objetos virtuales de señalización de los límites de los movimientos. Estas características de realidad aumentada están destinadas a aumentar la sensación de inmersión en el usuario y extender la duración de los ejercicios físicos. Para conseguir la precisión necesaria para el seguimiento del brazo de los pacientes, Ursus utiliza una cámara que tiene colocada bajo el cuello y algunas técnicas de realidad aumentada. Ursus sigue en constante desarrollo y pronto será evaluado su última versión.

*Figura 15: Ursus de RoboLab*

- SMAR: Es un robot diseñado para el aire libre en colaboración con la empresa laDEX y financiado por la Junta de Extremadura en virtud de concesión. El robot es omnidireccional, cada robot puede orientarse independientemente, puede soportar hasta 250 Kg e incluye varios sensores para auto localización y mapeo. SMAR es una base autónoma muy versátil que puede ser aumentada con herramientas específicas.

*Figura 16: SMAR de RoboLab*

- Muecas: Es una cabeza robótica. Está diseñada para ser utilizada en robots sociales para transmitir expresiones. Los componentes más sofisticados que posee son los ojos. Están hechos de nylon con esferas huecas que tienen cámaras fireware en su interior. Los globos pueden alcanzar velocidades y aceleraciones cercanas a las del ojo humano. El cuello tiene 3 DOF. Además la cabeza posee cejas, boca y está planificada la adición de párpados. Loki dispone de una versión de esta cabeza.

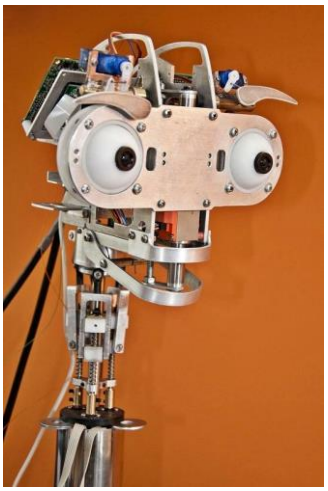

*Figura 17: Muecas de RoboLab*

Como podemos ver con los ejemplos, RoboComp es un framework para robótica muy potente y versátil, capaz de servir de apoyo a desde robots para movimiento en exteriores hasta a cabezas robóticas que gesticulan. Esto en parte, es gracias a su naturaleza de software

## 3.3. INNERMODEL

InnerModel (modelo interno) es una representación esencialmente geométrica que un robot tiene del mundo que le rodea, proporcionada por RoboComp y que podemos encontrar en el subdirectorio *Classes/innermodel*. Es su percepción, su visión del mundo y le permite saber dónde está aquello que le interesa, para poder relacionarse con el entorno. Dicho de otra manera, InnerModel se puede entender como un mundo virtual en el que se mueven objetos y personas modelados virtualmente. Como hemos dicho, es una representación esencialmente geométrica, por lo tanto carece de cualquier significado simbólico, se centra en qué está y dónde.

Esta información se almacena en un árbol cinemático, basado en el lenguaje XML. La idea es representar la escena en nodos de InnerModel de una forma incremental, comenzando desde el nodo "world". Colgando de éste se encontrará el nodo suelo, y colgando de él estará el propio robot, formado a su vez por uno o varios nodos. Esto es lo que da al robot conciencia de sí mismo, de dónde está y cómo es. Así, cada nuevo objeto o persona que entre en la escena se modela como un nodo o conjunto de nodos, enlazados entre sí. La posición relativa de todos estos nodos en el árbol cinemático de la escena se modifica continuamente, conforme se van detectando los cambios.

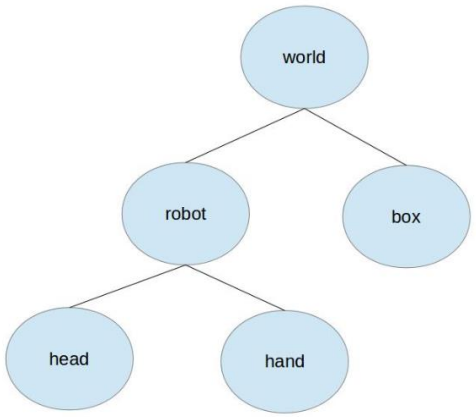


*Figura 18: Representación esquemática del mundo según InnerModel*

InnerModel tiene una jerarquía completa de tipos de nodos, cada una con unas características y significado especiales. De esta manera se distinguen los distintos tipos de objetos de la escena. Por ejemplo, el suelo es un nodo de tipo *PlaneNode*, y una persona está formada por nodos *JointNode*. Además todos estos nodos heredan de la clase RTMat, que permite hacer los cálculos geométricos más útiles en robótica de una forma muy rápida.

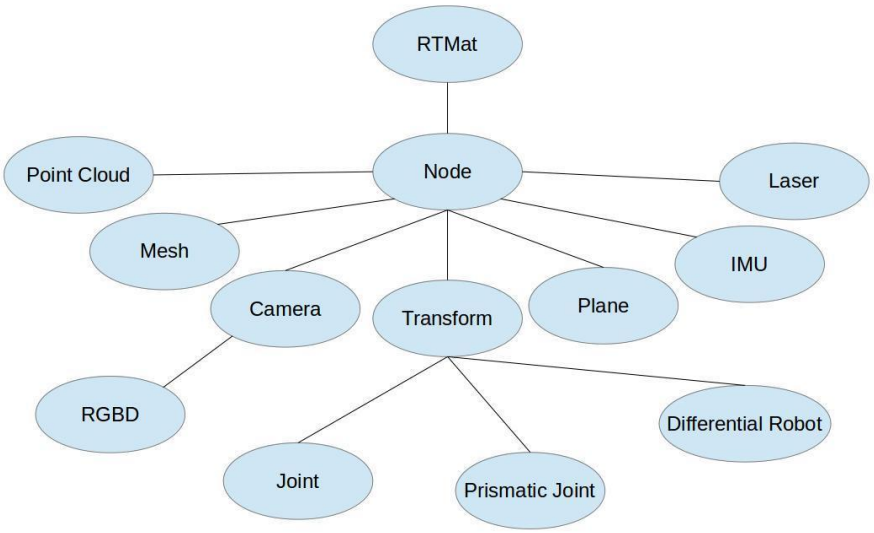


*Figura 19: Jerarquía de nodos de InnerModel*

Para interactuar con el árbol cinemático que nos proporciona InnerModel existen dos tipos de interfaces diferentes. La primera es la interfaz InnerModelManager.idsl, la cual permite que componentes externos creen, lean, editen o eliminen nodos del árbol cinemático que representa la escena. En este proyecto, para la simulación virtual del mundo real, la escena parte de un suelo con cuatro paredes y un robot sobre el suelo. A partir de aquí InnerModel irá interiorizando las personas que vayan apareciendo en la escena, actualizando sus movimientos constantemente o borrándolas cuando desaparezcan.

Los métodos de la interfaz InnerModelManager son los siguientes:

```
/// Node addition
bool addTransform(string item, string engine, string base,
Pose3D pose) throws InnerModelManagerError;
bool addJoint(string item, string base, jointType j) throws
InnerModelManagerError;
bool addMesh(string item, string base, meshType m) throws
InnerModelManagerError;
bool addPlane(string item, string base, Plane3D plane) throws
InnerModelManagerError;

/// Node information access
void getAllNodeInformation(out NodeInformationSequence
nodesInfo) throws InnerModelManagerError;

/// Generic node pose modification and access
// All get/sets and adds return false if an item or base doesn't
exist.
/// This methods transforms FROM item TO base
bool setPose(string base, string item, Pose3D pose) throws
InnerModelManagerError;
bool setPoseFromParent(string item, Pose3D pose) throws
InnerModelManagerError;
bool getPose(string base, string item, out Pose3D pose) throws
InnerModelManagerError;
bool getPoseFromParent(string item, out Pose3D pose) throws
InnerModelManagerError;
bool setPlane(string item, Plane3D plane) throws
InnerModelManagerError;
bool transform(string base, string item, coord3D
coordInItem,out coord3D coordInBase) throws
InnerModelManagerError;
Matrix getTransformationMatrix(string base, string item) throws
InnerModelManagerError;
bool setScale(string item, float scaleX,float scaleY, float
scaleZ) throws InnerModelManagerError;

/// Attribute-related methods
bool addAttribute(string idNode, string name, string type,
string value) throws InnerModelManagerError;
bool removeAttribute(string idNode, string name) throws
InnerModelManagerError;
bool setAttribute(string idNode, string name, string type,
string value) throws InnerModelManagerError;
bool getAttribute(string idNode, string name, out string type,
out string value) throws InnerModelManagerError;

/// Generic remove
// If an item doesn't exist or can't be removed, returns false.
All item's children are erased.
```

```
bool removeNode(string item) throws InnerModelManagerError;


/// PointCloud
//void addPointCloud(string id);
void setPointCloudData(string id, PointCloudVector cloud);


// Collision detection
bool collide(string a, string b);
```

Esta interfaz proporciona acceso directo al árbol cinemático de la escena, quedando en los componentes la responsabilidad de que todo lo almacenado sea correcto y coherente. De esta manera, se podría colocar a una persona de posturas físicamente imposibles, por lo que el componente o controlador que modifique el árbol, debería tener conocimiento sobre cómo mover a una persona, filtrando las posturas imposibles (como se propone en la Tesis de Juan P. Bandera [8]). En el caso de los robots sí que existen controladores que aseguran la veracidad de los movimientos.

La segunda forma de usar InnerModel es a través de interfaces de la capa de abstracción del hardware, esto es, de componentes que simulan sensores como el Láser o la Cámara, o que mueven los motores del robot. Estas interfaces dan acceso a elementos software en el InnerModel que leen datos sintéticos de los actuadores como si estuvieran percibiendo el mundo real.

De hecho, la forma de acceder a estos sensores simulados es a través de su interfaz Ice levantada en un puerto, tal y como si del otro lado de la interfaz hubiera un dispositivo real. Este acceso transparente permite utilizar InnerModel dentro del simulador que veremos a continuación.

### 3.3.1. RCInnerModelSimulator

RCInnerModelSimulator (RCIS) es el simulador de RoboComp. En robótica la simulación es indispensable, ya que ofrece la posibilidad de probar de forma virtual el funcionamiento de los algoritmos y programas implementados, así como en fases más avanzadas, simular las situaciones de error para depurarlos. Esto es muy útil, porque permite eliminar errores de diseño sin poner en peligro ningún tipo de sistema hardware, y además reduce los tiempos de desarrollo. De la misma forma permite a los desarrolladores trabajar con copias idénticas de sus robots, especificadas en el árbol de InnerModel, sin necesidad siquiera de disponer del robot físico

RoboComp InnerModel Simulator es una herramienta proporcionada por RoboComp y que se encuentra en el subdirectorio *Tools.* El uso de herramientas propias es una característica exclusiva de RoboComp no poseída por otros frameworks. El objetivo principal de esta herramienta es cargar un árbol cinemático que contiene una escena virtual, y ofrecer virtualmente a esta escena los mismos interfaces que RoboComp ofrece respecto a la realidad. De esta manera se recrea y visualiza un escenario virtual en el que se interactúa de manera similar a cómo sería en la realidad.

RoboComp InnerModel Simulator puede ejecutarse puramente como un simulador, o como una representación interna del robot, en la que se describe la información que éste percibe tanto del mundo como de sí mismo (propiocepción).

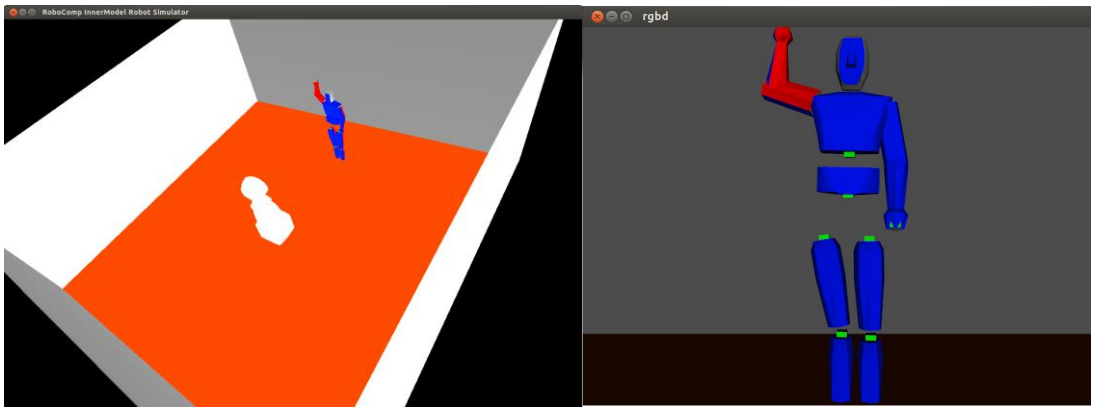


*Figura 20: RCIS, percepción del mundo y propiocepción*

Por otra parte, del entorno virtual hay que decir que puede editarse en tiempo de ejecución, permitiendo las operaciones típicas del árbol cinemático (traslaciones y rotaciones; adición y sustracción de nodos…), y que la visualización del entorno en 3D se realiza sincronizando el árbol cinemático con el motor gráfico OpenSceneGraph (OSG).

Las conexiones entre componentes, como en el mundo real, se realizan a través de sus interfaces, descritas en el lenguaje ICE (como dijimos antes), y son cargadas por el RCIS al arrancar.

La simulación entre componentes reales o simulados es transparente al desarrollador, ya que sólo tendrá que ajustar los parámetros mencionados anteriormente. De esta manera, un componente podría ser arrancado sobre un robot real, detenido y justamente después arrancado sobre un robot simulado, sin necesidad de compilar el componente nuevamente. De la misma manera un robot simulado podría tener arrancados sobre sí unos componentes simulados y otros reales. Esto serviría para probar algunos componentes reales en conjunto con sistemas simulados que no posee el robot realmente, como puede ser un láser o una cámara RGBD.

Los interfaces más destacables que ofrece el simulador de RoboComp son:

- DifferentialRobot: Reimplementa las funciones típicas de una base diferencial robótica (avance, giro, odometría…), permitiendo el movimiento del modelo del robot en el espacio virtual. A diferencia de lo que ocurre en un movimiento real, en la simulación no ocurren errores, ni sistemáticos ni no sistemáticos.
- JointMotor: Simula el comportamiento de un motor de corriente continua proporcionando funciones de posicionamiento. Admite rangos límites de movimiento angular, y ajustes de velocidad de giro, para conseguir un mayor realismo, aunque como ocurre con la base diferencia, no se introduce ruido en la precisión del motor.
- Laser: Simula un láser de barrido, configurable en campo de visión y resolución. Representa en la escena el campo de acción dibujando un polígono con superficie igual al área de barrido. Obtiene la distancia a los objetos por la intersección de estos haces con las mallas poligonales de los objetos de la escena.
- RGBD: Simula una cámara RGB y un dispositivo de infrarrojos para calcular la profundidad de la imagen. Su objetivo es simular el comportamiento de los sensores PrimeSense empleados por los dispositivos Kinect y Asus Xtion. Permite configurar los parámetros intrínsecos de ambos y reimplementa las funciones básicas de estos perceptores.
- InnerModelManager: Podría considerarse como la interfaz nativa de esta herramienta, ya que como dijimos en el punto anterior, proporciona acceso directo a los nodos del árbol cinemático permitiendo su edición y actualización en tiempo de ejecución. De la misma forma proporciona acceso a la lista de atributos clave-valor contenidos en él, los cuales sirven para representar información semántica asociada al nodo.

En el proyecto se ha utilizado el simulador de RoboComp para comprobar que se obtenían y transformaban al modelo RoboComp correctamente los datos obtenidos con el dispositivo Xtion Pro, la librería OpenNI2 y el middleware NiTE2. En él también se ha simulado el robot Gualzru (robot blanco de la imagen anterior), el cual está provisto de una cámara RGBD también simulada que permite ver el modelo de la

persona dentro del simulador. Este sistema recrea el mundo real, haciendo la cámara RGBD de Gualzru las veces de la cámara Asus Xtion real, y abstrayéndose de tal forma que obvia cualquier otro elemento real que no sean personas, que son los únicos elementos que se introducen en el simulador, porque son los que realmente interesan en este proyecto.

## 3.4. ENTORNO DE DESARROLLO

A la luz de lo leído es fácil suponer que para desarrollar una aplicación basada en esta arquitectura, es necesario utilizar un conjunto de herramientas más o menos complejas y librerías que soporten las tareas exigidas a las capacidades del robot que utilice los componentes que en este proyecto se desarrollan. La naturaleza de los problemas, eminentemente en tiempo real, a su vez exige que el tiempo de proceso sea mínimo y siempre se persigue, por filosofía del laboratorio y en la medida de lo posible, un mínimo consumo de recursos. En las siguientes páginas describiremos el software usado.

### 3.4.1. IPP

IPP es una biblioteca para procesar señales de una y dos dimensiones. IPP es desarrollada por Intel, y es software privativo. A pesar de que es gratuito para el uso personal bajo GNU/Linux, hay que pagar la licencia si se desea usar comercialmente, y su código fuente no es público. La elección de esta biblioteca frente a otras de similar objeto se tomó por su gran eficiencia.

La diferencia en el rendimiento respecto a sus competidores radica en que hacen uso de las instrucciones SIMD que aportan las extensiones de x86, 3DNow!, MMX y SSE y derivadas que Intel incluye en sus procesadores. La eficiencia del software desarrollado, no sólo en este proyecto sino en el resto del software se debe en gran

parte a la de esta biblioteca. De ella se han usado multitud de funciones en este y otros componentes relacionadas con el tratamiento de imágenes.

### 3.4.2. QT

Qt4 es un framework de desarrollo cuyo objeto principal es la creación de interfaces gráficas. A pesar de que sea este su principal uso, engloba una gran cantidad de funcionalidades distintas: interfaz multiplataforma con el sistema operativo (sistema de ficheros, procesos, hilos entre otras cosas), comunicación por red mediante sockets, interfaz con OpenGL, conexiones SQL, renderizado de HTML, y módulos de reproducción y streaming multimedia.

Atendiendo al lenguaje de programación, Qt está escrita en C++, pero existen multitud de bindings que hacen posible su uso desde otros lenguajes como Java, C#, Python, Perl y otros muchos.

Inicialmente fue desarrollada por Trolltech, una empresa noruega, bajo una licencia privativa. Después de varios cambios en la política de licencias pasó a distribuirse bajo una doble licencia GPL/QPL (esta última privativa). Finalmente, tras la compra de Trolltech por parte de Nokia, Qt fue licenciada bajo LGPL en 2009, eliminando así cualquier debate sobre la licencia de la biblioteca.

### 3.4.3. ICE

Ice es el middleware del que se habló en la sección anterior que permite crear componentes software. Es usado por RoboComp y, por tanto, por los componentes desarrollados en el proyecto de fin de carrera. Es el principal producto de la empresa estadounidense ZeroC, y se distribuye bajo una doble licencia GPL+privativa para habilitar que las empresas desarrollen software privativo con Ice, a cambio del pago

de una licencia. Las principales características de Ice frente a otras tecnologías similares es su rapidez, baja latencia y escalabilidad.

Uno de los problemas a resolver a la hora de crear componentes software es la creación de un lenguaje de definición de interfaces. Ice usa Slice, un lenguaje que se creó especialmente para este propósito.

### 3.4.4. CMAKE

CMake es una aplicación que permite generar automáticamente ficheros Makefile y ficheros de proyecto de varios IDE como KDevelop o Eclipse entre otros. CMake permite delegar la creación de ficheros Makefile, consiguiendo un resultado multiplataforma y robusto, sin llegar a perder el control del proceso de compilación.

CMake es una iniciativa libre que nació como respuesta a la ausencia de una alternativa suficientemente buena durante el desarrollo de una librería llamada ITK. Si bien dispone de soporte para Qt y algunas otras extensiones, es muy simple hacer nuevas extensiones.

### 3.4.5. KDEVELOP

Es un entorno de desarrollo integrado para sistemas GNU/Linux y otros sistemas Unix, publicado bajo licencia GPL, orientado al uso bajo el entorno gráfico KDE, aunque también funciona con otros entornos, como Gnome.

El mismo nombre alude a su perfil: KDevelop KDE Development Environment (Entorno de Desarrollo para KDE). A diferencia de muchas otras interfaces de desarrollo, KDevelop no cuenta con un compilador propio, por lo que depende de gcc para producir código binario.

### 3.4.6. GNU/LINUX

Finalmente el sistema operativo usado durante el desarrollo y los experimentos, GNU/Linux. Gracias a él disponemos de un entorno de desarrollo gratuito y libre, herramientas de compilación, depuración, y una gran cantidad de bibliotecas complementarias. Se ha utilizado la distribución Ubuntu, por razones subjetivas.

## 3.5. OPENNI2

OpenNI (Open Natural Interaction) es un framework open source que provee APIs para el desarrollo de aplicaciones que utilicen interacción natural para la interacción con el usuario, esto es, interacción basada en los sentidos humanos como pueden ser el oído o la visión.

OpenNI facilita la comunicación con los sensores de audio, video y profundidad de los dispositivos hardware compatibles como Kinect o ASUS Xtion (ambos basados en el chip PS1080 de PrimeSense), así como también, la comunicación con middleware de percepción encargado de analizar y comprender los datos obtenidos de la escena mediante algoritmos de computer vision. Por ejemplo, software que recibe datos visuales como una imagen y retorna la posición de la palma de la mano detectada en la imagen, o identifica ciertos gestos predefinidos y alerta a la aplicación cuando los detecta.

Lo anterior se puede visualizar mejor en la siguiente imagen que ilustra la arquitectura del framework:

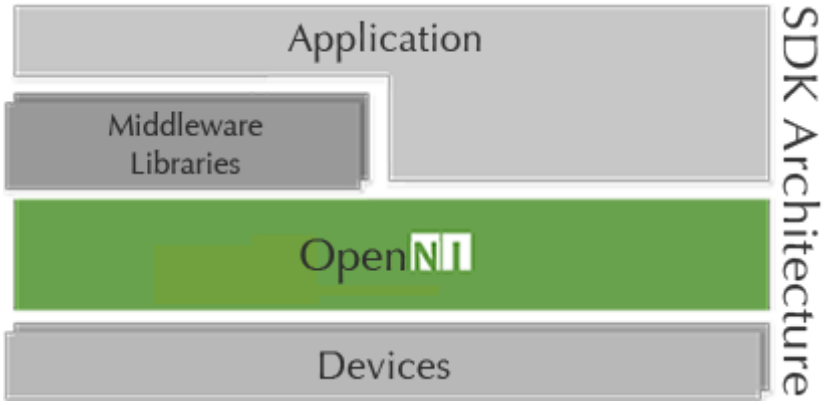


*Figura 21: Arquitectura SDK OpenNI*

Como se puede observar en la Figura, OpenNI provee a las aplicaciones una interfaz de comunicación con los diferentes sensores hardware así como también, con los diferentes componentes middleware que se registren en el framework. Un punto a destacar es que provee únicamente las interfaces, relegando la lógica que implementa cada funcionalidad a los diferentes componentes, sean sensores o middleware; algunas operaciones de la API estarán implementadas por los dispositivos hardware y otras por los componentes middleware. Así, por ejemplo, OpenNI provee operaciones para obtener el mapa de profundidad (mapa en el cual cada punto representa la distancia desde el sensor) de la escena o los datos crudos de imagen RGB desde los sensores, cuya lógica estará implementada en el propio sensor.

Análogamente para los componentes middleware, provee operaciones para obtener la posición de la mano o del esqueleto del usuario, cuya lógica estará implementada en el propio middleware. De esta forma, distintos proveedores de hardware y middleware OpenNI-compatible pueden desarrollar su propia implementación de las diferentes funcionalidades provistas por OpenNI y luego registrarlos en el framework para su utilización.

Además, se pueden tener registrados varios componentes que implementen la misma funcionalidad y elegir cuál de ellos utilizar al momento de solicitar los datos.

Esta API común provista por OpenNI lo hace extremadamente flexible para la incorporación de nuevos componentes middleware así como también para el desarrollo y ejecución de aplicaciones, que pueden utilizar la API de forma transparente e independiente del dispositivo hardware (sensores) que genere los datos.

Resumiendo, OpenNI permite interactuar básicamente con dos componentes. Por un lado componentes hardware como por ejemplo sensores 3D y cámaras RGB que proveen datos crudos obtenidos a partir de la escena actual, y por otro componentes middleware que se encargan de procesar y comprender esos datos crudos para brindar información de mayor nivel de abstracción, los cuales son en definitiva el cerebro del sistema que puede ver la escena y analizarla. Los sensores observan el entorno y el middleware hace de motor de percepción para comprender la interacción del usuario con ese entorno.

De todas formas, cabe comentar que OpenNI además de proveer la API se encarga de gestionar y llevar el rastro de todos los componentes registrados (sensores y middleware) y su contexto, entre otras muchas funcionalidades y utilidades adicionales como implementación de diferentes tipos de datos como listas y hashes, logging, manejo de eventos, etc.

### 3.5.1. INSTALACIÓN DE OPENNI2 EN LINUX

En este punto vamos a comentar cómo instalar OpenNI2 en distribuciones Linux, y que sea reconocible por RoboComp. El proceso está formado por unos sencillos pasos que enumeramos:

1) Descargar la librería desde la página oficial [9]. Tiene un nombre del tipo OpenNI-Linux-x64-X.X.tar.bz2 para arquitecturas de 64 bits o OpenNI-Linux-x86-X.X.tar.bz2 para máquinas de 32 bits.
2) Descomprimir el archivo en el directorio donde se desee alojar la librería

3) Acceder a la carpeta de la librería y ejecutar el archivo install.sh. Esto se debe hacer desde la terminal con el siguiente comando: sudo ./install.sh. Con esto ya se tendría la librería instalada y disponible.
4) Crear la variable de entorno OPENNI2. Para que RoboComp sepa la ubicación de la librera. Los pasos a seguir son:
    a. Acceder desde la consola a la carpeta del usuario: cd ~
    b. Acceder al archivo bashrc con el comando: sudo nano .bashrc
    c. Añadir la línea EXPORT OPENNI2= [directorio de la librería]. Un ejemplo podría ser: EXPORT OPENNI2=/home/usuario/Software/OpenNI-Linux-XXX-X.X.

## 3.6. NITE2

Es uno de los componentes middleware comúnmente más utilizados. NITE (Natural Interaction Technology for End-user) es la implementación de PrimeSense del motor de percepción, el cual implementa entre otras cosas tracking esqueletal, hand-tracking, reconocimiento de varios gestos predefinidos o la identificación de usuarios y determinación de cuál de ellos tiene el control en cada momento.

Como ya hemos dicho, NITE2 trabaja sobre OpenNI y consta de las siguientes capas:

- Módulos OpenNI: Los módulos de OpenNI soportados por NITE son el generador de gestos y generador de manos. Además NITE ofrece unos ejemplos que muestra el módulo analizador de escena y generador de usuarios, con el seguimiento esqueletal.
- Infraestructura OpenNI: Explicada en el punto anterior **3.5.**
- Gestor de Controles: Recibe un flujo de puntos y los encamina al control de NITE apropiado.
- Controles: Cada control recibe un flujo de puntos y los traduce en una acción significativa para ese control. El control entonces llama a los callbacks de la

aplicación que puede cambiar el control activo actual en la capa del gestor de controles, definiendo el flujo de la aplicación.

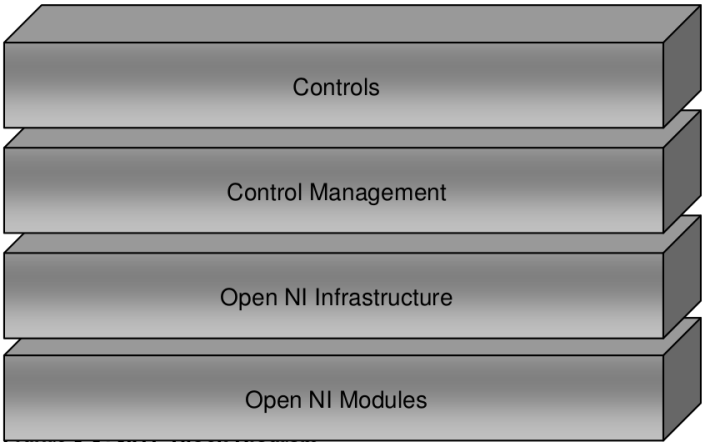


*Figura 22: Capas de NITE*

Una vez vista la arquitectura del middleware, pasaremos a explicar con mayor detalle las dos principales funcionalidades de NITE, el seguimiento del cuerpo completo y el seguimiento de manos para la detección de gestos.

### 3.6.1. TRACKING ESQUELETAL

Una de las principales y más útiles funcionalidades de este middleware es el seguimiento del cuerpo entero humano. Este proceso consta de dos fases fundamentales:

**Calibración**

La fase de calibración se realiza justo cuando se detecta a una persona en escena, y su función es ajustar el modelo del esqueleto a las proporciones del cuerpo del usuario. Esta fase es necesaria y previa al seguimiento. Para una mejor calibración el usuario debe adoptar la postura de la imagen:

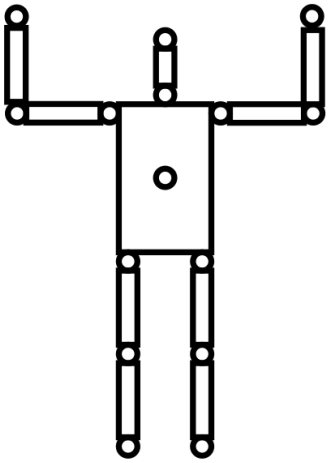

*Figura 23: Pose de calibración de NITE*

Es importante que la cabeza y los brazos sean completamente visibles en esta fase, y que no se flexionen el torso ni las rodillas.

**Seguimiento**

La segunda fase, y en la que NITE empieza a dar una salida es la fase de seguimiento. La API devuelve la posición y orientación de las articulaciones del esqueleto.

Los valores devueltos de la posición son más fiables que los de la orientación. A pesar de que NITE permite que la longitud de cada segmento del cuerpo (codo-hombro, hombro-mano…) varíe en función del tiempo, la orientación es más ruidosa.

Para entender la salida del seguimiento de personas hay que tener en cuenta ciertos detalles que definen la transformación de las articulaciones.

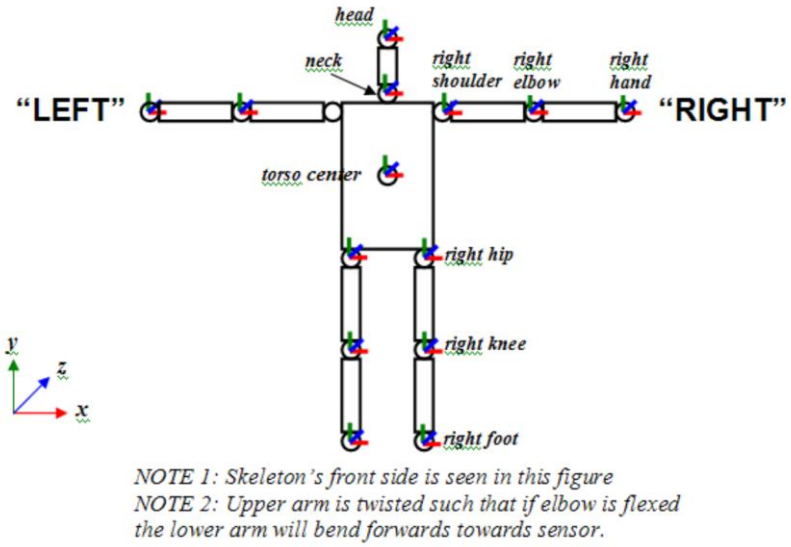


*Figura 24: Sistema de coordenadas de NITE*

La figura superior muestra el sistema de coordenadas y la representación esqueletal de un usuario que está de cara al sensor. Lo explicamos con más detalle abajo, pero como primer detalle es importante decir la definición de los lados "IZQUIERDA" y "DERECHA" del esqueleto.

Suponiendo que el usuario está de cara al sensor, en la pose en T que se muestra en la figura, se asume que la imagen de profundidad está en modo espejo. Esto significa que la pantalla funciona como un espejo. El algoritmo llama lado derecho al lado izquierdo del modelo, y viceversa.

Las posiciones y orientaciones de las articulaciones se dan en el sistema de coordenadas del mundo real. El origen de coordenadas es el sensor. +X se orienta hacia la derecha, +Y hacia arriba y +Z en la dirección en la que crece la profundidad. Se puede ver más claramente en la imagen superior.

Las posiciones vienen dadas en milímetros, y las orientaciones en NITE2 en cuaterniones [13], a diferencia de versiones anteriores, que venían en matrices de rotación ortonormales. Además con cada articulación se devuelve un valor de

confianza, que está entre 0 y 1, siendo 0 un valor que da pocas garantías de la fiabilidad de los datos de la articulación y 1 un valor muy fiable.

Para el correcto funcionamiento del seguimiento de personas y el entendimiento de problemas que encontraremos en este proyecto, acerca de NITE2 debemos saber que:

- La mayor parte superior del cuerpo debe estar en el campo de visión del dispositivo.
- La distancia ideal para el seguimiento es de 2,5 metros.
- Para mejores resultados es mejor evitar ropas anchas y pelo suelto.
- El seguimiento del brazo es menos estable cuando está cerca de otras partes del cuerpo, en especial del torso. Es posible que acaben mezclándose.
- El seguimiento de las piernas es todavía un tanto ruidosa, funciona mejor si el usuario tiene las piernas bien separadas.
- Si los brazos o las piernas están extremadamente separadas, cerca del límite humano puede perderse el seguimiento.
- Si el esqueleto se queda bloqueado en una postura extraña o el seguimiento es erróneo, basta con volver a ponerse en la postura de calibración [Figura 23].
- Movimientos muy rápidos pueden provocar un fallo en el seguimiento.

### 3.6.2. TRACKING DE MANOS Y CONTROL POR GESTOS

NITE2 ofrece la posibilidad de hacer el seguimiento de las manos de uno o varios usuarios. El seguimiento obtiene la posición de una mano o varias y puede, mediante un algoritmo, detectar qué gesto se está haciendo. NITE2 tiene varios gestos predefinidos que comentaremos posteriormente con mayor detalle. Además se ofrece la posibilidad del control de las aplicaciones mediante estos gestos. En este punto es cuando entra en juego los conceptos de sesiones y controles definidos en NITE2.

#### *3.6.2.1. SESIONES*

Frecuentemente existen varias personas en la escena. Cuando se implementa una interfaz, sólo una de ellas debe poder controlarla durante un periodo de tiempo. Pues bien, una sesión es un estado en el que un usuario tiene el control del sistema. Mientras la sesión dura, se hace el seguimiento de las manos y se les asigna un ID fijo. Para comenzar una sesión hay que definir un gesto identificativo de inicio de sesión. El usuario que realice dicho gesto tomará el control.

Respecto a las sesiones existen tres estados:

- En sesión: En este estado se está haciendo el seguimiento de las manos y detectando el gesto identificativo.
- No sesión: En este estado se está buscando un gesto identificativo para iniciar una nueva sesión. Una vez que el gesto es reconocido, el estado pasa a 'En sesión'.
- Reenfoque rápido: En este estado, previamente se había estado en el estado 'En sesión' pero se ha perdido el foco de las manos. Una vez que esto pasa se deja un periodo de tiempo prudencial para poder retomar este foco. Cuando un gesto se identifica el estado vuelve a pasar a 'En sesión'. En el caso de que se consuma este tiempo el estado pasa a 'No sesión'. Este estado es opcional.

El flujo de estados puede definirse con el siguiente esquema:

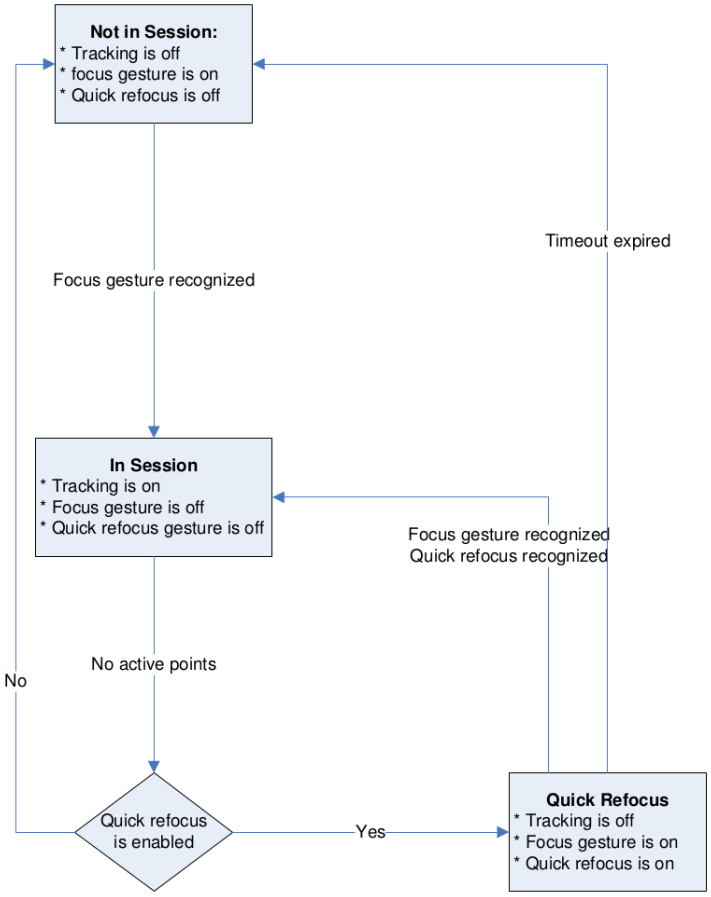


*Figura 25: Máquina de estados de una sesión de NITE*

Al principio, el estado es de 'No sesión' y se está buscando el gesto de inicio de sesión. Cuando el gesto es identificado el estado cambia a 'En sesión', y comienza el seguimiento de manos para la detección de gestos. Cuando deja de haber manos (o se han perdido), el estado pasa a 'No sesión' si el reenfoque rápido está desactivado o al estado de 'Reenfoque rápido' en caso contrario. En este estado se busca tanto el gesto de inicio de sesión como otro que se haya definido de reenfoque rápido, si alguno de los dos es detectado antes del paso de un tiempo configurable, el estado pasa a en 'En sesión', si este tiempo se agota el estado pasa a 'No sesión'.

#### *3.6.2.2. CONTROLES*

Los controles son objetos que reciben algún tipo de datos y realizan alguna acción con esos datos.

El control más común de NITE es el control de puntos, el cual registra cuándo se ha creado, se ha movido o desaparece algún punto nuevo. El control de puntos recibe los puntos activos de alguna fuente (por ejemplo del gestor de sesiones) y trata de entender esos puntos como alguna acción. Los controles tienen eventos que se pueden registrar cuando algún gesto es reconocido. Los controles disponibles de NITE son:

- Detector de pulsado: Este control detecta un movimiento de "empuje" hacia el sensor y viceversa
- Detector de deslizamiento: Este control intenta detectar el deslizamiento de la mano en cualquier dirección.
- Detector estacionario: El control detecta cuando la mano lleva fija en un lugar durante un tiempo
- Detector de ondas: El control detecta un movimiento ondulatorio de la mano. Una onda se produce con una serie de cambios de dirección (por defecto cuatro) dentro de un tiempo de espera.
- Detector de círculos: Se detecta un movimiento circulatorio. Debe hacerse un círculo completo, ya sea en el sentido de las agujas del reloj (considerado dirección positiva), o en sentido contrario (negativa).
- Deslizador 1D. Detecta un movimiento en una zona de deslizamiento alineada con cualquiera de los tres ejes. Cada zona se divide en partes iguales devolviendo un valor entre 0 y 1 dependiendo de la posición de la mano dentro de esa zona
- Deslizamiento seleccionable 2D. Es un control igual al anterior pero definido sobre el plano XY. Devolviendo dos valores entre 0 y 1 en lugar de uno.

- Punto principal: Todos los gestos anteriores funcionan con un punto de la mano, no con todos los que la forman. Este control define ese punto, y en el caso en que lo pierda, si existen otros puntos, define a otro como principal.

Para este proyecto utilizaremos únicamente el seguimiento del cuerpo completo. Aunque tendremos que transformar los datos que NITE nos ofrece, puesto que ni las posiciones ni las orientaciones están en el mismo sistema de referencia ni de la misma forma que utiliza RoboComp. La transformación necesaria la explicaremos en puntos posteriores.

### 3.6.3. INSTALACIÓN DE NITE2 EN LINUX

Como hicimos con OpenNI2, vamos a comentar cómo instalar NITEI2 en distribuciones Linux, para que sea accesible por RoboComp. El proceso está formado por los siguientes pasos:

1) Descargar la librería desde el siguiente enlace [10]. Tiene un nombre del tipo NITE-Linux-x64-X.X.tar.bz2 para arquitecturas de 64 bits o NITE-Linux-x86-X.X.tar.bz2 para máquinas de 32 bits.
2) Descomprimir el archivo en el directorio donde se desee alojar la librería
3) Acceder a la carpeta de la librería y ejecutar el archivo install.sh. Esto se debe hacer desde la terminal con el siguiente comando: sudo ./install.sh. Con esto ya se tendría la librería instalada y disponible.
4) Crear la variable de entorno NITE2. Para que RoboComp sepa la ubicación de la librera. Este paso puede hacerse así:
    a. Acceder desde la consola a la carpeta del usuario: cd ~
    b. Acceder al archivo bashrc con el comando: sudo nano .bashrc
    c. Añadir la línea EXPORT NITE2= [directorio de la librería]. Un ejemplo podría ser: EXPORT NITE2=/home/usuario/Software/NITE-Linux-XXX-X.X

5) Una vez creado el componente que se vaya a implementar es necesario copiar la carpeta NiTE2 que se encuentra en el directorio de la librería NITE-Linux-XXX-X.X/Redist en la carpeta del componente nombreComponente/bin, es decir, en el mismo directorio que el ejecutable del componente.

# 4. ESTADO DEL ARTE

## 4.1. DETECTANDO LA TERCERA DIMENSIÓN

Existen muchos métodos para detectar distancias en robótica. Unos necesitan contactos y otros no. Dada la naturaleza de este proyecto, nos centraremos en el estudio de los que no lo requieren. Dentro de éste grupo hay dos tipos:

- Métodos activos: Para obtener la información necesitan de la emisión de algún tipo de radiación, como pueden ser rayos X, infrarrojos o luz.
- Métodos pasivos: No emiten radiación por sí mismos, se bastan con la radiación ambiental.

A continuación trataremos los métodos más utilizados para detectar distancias.

### 4.1.1. MÉTODO RGB

El método RGB es  pasivo, no necesita emitir radiación alguna. Únicamente necesitan luz visible, es decir, que la escena este iluminada para conseguir detectar las distancias de los objetos. No es un método en concreto, sino que hay distintas propuestas que comentamos:

- Georgios Vouzounaras, Ingeniero de Software de California propuso un método para determinar la geometría tridimensional de un objeto con una única imagen RGB detectando los puntos de fuga. El método cuenta con el conocimiento de las estructuras geométricas. Es necesario que aparezcan el suelo el techo y las paredes para que el algoritmo funcione. Es capaz de reconstruir objetos de geometría sencilla pero no objetos esféricos.
- Ashutosh Sexana, profesor de la Universidad de Cornell, investigó sobre cómo reconstruir una escena 3D a partir de una o muy pocas imágenes RGB. Consiste en segmentar la imagen en pequeños parches llamados superpíxeles, y utilizando el campo aleatorio de Markov (MRF) se intenta encontrar la posición y orientación tridimensional de cada uno.

- Stewart Weiss, profesor asociado de la Universidad de Nueva York, estudió la detección de objetos a partir de imágenes 2D tratando el concepto de invariantes geométricas. Esas invariantes las encontraban al hacer ciertas suposiciones para modelos tridimensionales de los objetos. A través de ciertas relaciones algebraicas era capaz de describir modelos invariantes en 3D.
- Eddy Zheng, profesor de la Universidad de Rutgers, ideó un método para reconstruir modelos 3D a partir de varias imágenes RGB captadas por cámaras que no necesitan ser previamente calibradas. Combina técnicas como la detección de esquinas de Harris y la detección de líneas para la definición del objeto. Este método requiere cuatro cámaras y una mesa giratoria para captar el objeto en su integridad.

Como podemos comprobar, estos métodos requieren un procesamiento intermedio en mayor o menor medida complejo entre la captación y la detección. Además para algunos son necesarios objetos externos, como más cámaras o mesas giratorias. Y todos tienen puntos débiles, funcionan únicamente de unas características físicas concretas de la escena o no detectan todos los objetos.

### 4.1.2. MÉTODO ESTEREOSCÓPICO

Es otro método pasivo, distinto a los anteriores. Necesita una precisa preparación previa, pero posteriormente ofrece buenos resultados. Es un método muy estudiado y que se usaba habitualmente antes de la llegada de las cámaras RGBD.

La visión estereoscópica constituye un procedimiento para la obtención de la forma de los objetos de la escena. Toma como referencia el modelo estereoscópico biológico, donde el desplazamiento relativo de los ojos permite obtener la profundidad de los objetos o tercera dimensión mediante un simple proceso de triangulación a partir de dos imágenes generadas por el mismo objeto de la escena

3D en cada ojo. Esto se debe al hecho de que los ojos están distanciados, haciendo que las imágenes de los objetos en sendos ojos se muestren desplazadas según la distancia de los objetos a los ojos. Podemos ver cómo funciona la visión estereoscópica en la siguiente imagen:

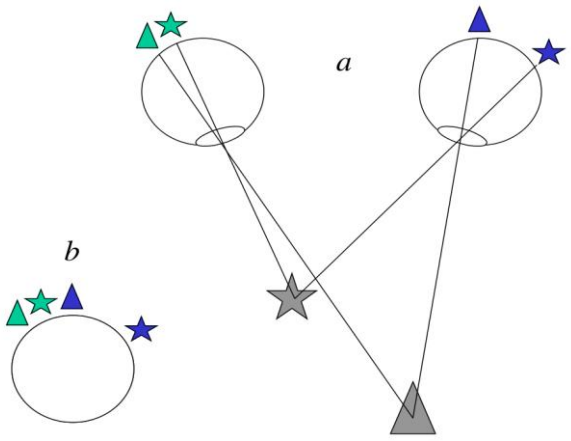


*Figura 26: Idea base de la visión estereoscópica*

Si superponemos las imágenes captadas por cada ojo obtenemos la imagen de la figura 26.b. Como podemos comprobar la separación relativa entre las estrellas es mayor que la que hay entre los triángulos, y esto es porque está más cerca de los ojos. Cuanto más lejos está un objeto menos distancia relativa hay, así llega una cierta distancia en la que no distinguimos la profundidad de los objetos. Ésta separación relativa de las imágenes se denomina disparidad y es el eje de este método.

Para recrear este sistema de forma artificial son necesarias dos cámaras RGB con las que se obtienen las correspondientes imágenes del par estéreo. Deben estar colocadas exactamente igual en dos de los ejes de coordenadas y separadas entre sí en el otro eje una distancia relativa y conocida. El procedimiento consiste en captar dos imágenes de una misma escena, cada una con una cámara. Las imágenes se verán ligeramente desplazadas una de la otra, este hecho permitirá la obtención de la distancia de un determinado objeto.

Una vez tomadas ambas imágenes se ejecutará una técnica de correspondencia, para determinar las mismas partes de la escena en ambas imágenes. Hay dos grupos de técnicas:

- Técnicas basadas en las características: Estos métodos restringen la búsqueda a un conjunto disperso de características. Se emplean propiedades simbólicas y numéricas de las características, obtenidas a partir de los llamados descriptores, los cuales se encargan de procesar y extraer características de una determinada imagen.
- Técnicas basadas en el área: En esta técnica los elementos a comparar son ventanas de la imagen de dimensión fija, y el criterio de semejanza es una medida de la correspondencia entre las ventanas de las dos imágenes. El elemento correspondiente queda determinado por la ventana que maximiza el criterio de semejanza dentro de la región de búsqueda.

Cuando ya se tiene determinada las partes de las imágenes que corresponden entre sí, basta con aplicar una sencilla ecuación matemática de triangulación para calcular la distancia a la que se encuentra un determinado punto de la imagen.

### 4.1.3. MÉTODO TIME-OF-FLIGHT

Este método calcula la distancia de un objeto a partir del tiempo que tarda un haz de luz infrarroja en ir y volver. Por lo tanto, es un método activo. Existen varios métodos, con varios dispositivos distintos, englobados en este método. Se pueden utilizar dispositivos como diodos láser, cámaras PMD (Photonic Mixer Device), cámaras infrarrojas...

El procedimiento consiste en lanzar un rayo láser o infrarrojo y calcular cuánto tiempo tardan en volver. El tiempo determinará la distancia a la que está el objeto contra el que ha golpeado. Y viene cuantificada por la siguiente ecuación:

$$d = CT/2$$

Siendo C la velocidad de la luz y T el tiempo que ha tardado en llegar.

El esquema del funcionamiento del *tiempo de vuelo* es el siguiente:

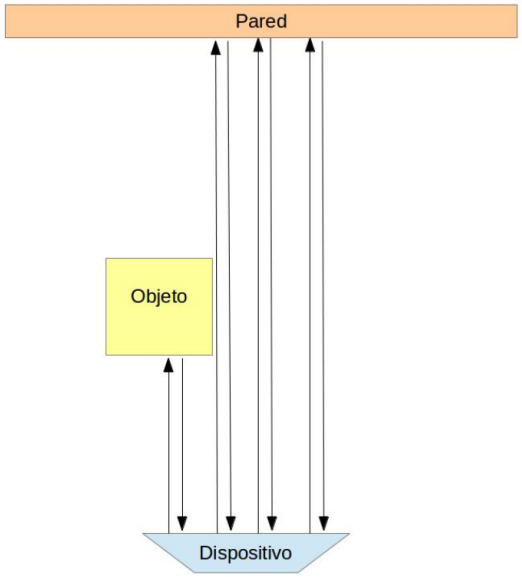


*Figura 27: Time of flight*

Los rayos que inciden en el objeto tardan menos tiempo en llegar (tienen menos recorrido y la velocidad de la luz es constante), que los que golpean contra la pared del fondo, por lo que ese objeto está más cerca.

El gran problema de este método es el precio de los dispositivos que se utilizan.

### 4.1.4. MÉTODO DE LUZ ESTRUCTURADA

El método de luz estructurada está basado en la idea del tiempo de vuelo. Es aquí donde podemos englobar a las cámaras RGBD. Éste método consiste en lanzar una nube de rayos infrarrojos. Cada uno de ellos rebotará en alguna parte de la escena y llegará en un momento determinado. Conociendo un mapa determinado de distancias conocidas, el dispositivo compara por hardware el desplazamiento que existe entre el mapa conocido y el obtenido, calculando así la distancia a la que está cada punto.

Es común que junto con el láser infrarrojo y la cámara, haya en el dispositivo una cámara RGB trabajando en paralelo, en este caso, se asocia a cada uno de los puntos de la imagen RGB una distancia determinada por la cámara infrarroja. Con estos datos se pueden crear mapas tridimensionales de un nivel de detalle que viene dado por la resolución de ambas cámaras.

Este método ofrece unos datos muy precisos. Además, las cámaras RGBD son bastante económicas en comparación con los láseres y otros dispositivos del apartado anterior. Pero como todas las técnicas y dispositivos comentados anteriormente, Las cámaras RGBD también tienen puntos débiles, y que son necesarios comentar:

- Hay en ciertos tipos de superficies que reflejan los rayos infrarrojos y no llegan correctamente, fallando en algunas distancias.
- No puede usarse en exteriores debido al sol.
- Los cristales no los detecta, ya que los rayos infrarrojos los atraviesan.

Aun con estos errores, las ventajas hacen que las cámaras RGBD sean, que hoy por hoy el método más utilizado, y el que usaremos en este proyecto.

## 4.2. DETECCIÓN Y SEGUIMIENTO DE PERSONAS

### 4.2.1. DETECCIÓN Y SEGUIMIENTO DE PERSONAS A PARTIR DE IMÁGENES RGB

Esta técnica se basa en detectar las zonas de la imagen donde se produce movimiento, separar la zona estática de la zona dinámica, y detectar si la zona dinámica corresponde a una persona. Para cada uno de los pasos a realizar hay distintas técnicas aplicables, que comentaremos ahora.

#### *4.2.1.1. SUSTRACCIÓN DE FONDO*

Para detectar las zonas dinámicas y separarlas de la zona estática se suele utilizar lo que se conoce como técnicas de extracción de background. Estas técnicas se diferencian entre sí en el tipo de modelo de fondo que utilizan y en las técnicas basadas en dicho modelo de fondo para encontrar las regiones estáticas.

*Figura 28: Sustracción de fondo*

Existen dos categorías de técnicas: aproximaciones que usan un modelo de fondo y aproximaciones que usan varios modelos. El primero de estos grupos se puede dividir a su vez en dos subcategorías dependiendo del uso que dichas aproximaciones hagan con las máscaras de foreground obtenidas:

- Análisis imagen a imagen. Esta categoría describe los métodos que emplean modelos de segmentación frente-fondo bastante comunes, seguidos de otro tipo de análisis. Dependiendo del tipo de análisis pueden ser:
    - Aproximaciones clásicas: basadas en el uso de técnicas sencillas de segmentación frente-fondo y un postprocesado de la máscara de foreground seguido a su vez de alguna otra etapa de análisis.
    - Basados en la acumulación de máscaras de foreground. Dicha acumulación se realiza frame a frame y con ella se puede moldear una máscara final de foreground, de donde se obtienen las regiones estáticas.
    - Basados en algunas propiedades del modelo de fondo utilizado, como por ejemplo considerando las transiciones entre los diferentes estados de un modelo de mezcla de Gaussianas u observando el valor de algunos parámetros como, por ejemplo, el peso de las Gaussianas.
- Análisis de máscaras de foreground muestreadas. Estas aproximaciones intentan detectar regiones estáticas analizando la secuencia de vídeo a diferentes velocidades, aprovechándose de las ventajas espacio-temporales que ello conlleva.

El segundo grupo, el cual es menos utilizado, está formado por técnicas que combinan más de un modelo de fondo por cada píxel:

- Aproximaciones basadas en el análisis imagen a imagen: Se combinan las propiedades de los diferentes modelos de fondo que se utilizan.
- Aproximaciones basadas en el sub-muestreo. Estas aproximaciones detectan regiones estacionarias analizando la secuencia de vídeo a través de los diferentes modelos de fondo debido a que cada modelo de fondo se muestra con una tasa binaria diferente.

Los algoritmos de sustracción de fondo más utilizados en la detección de personas son el modelo de mezcla de Gaussianas (MOG) y el modelo de Bayes (FGD):

- **MOG** (del inglés Mixture of Gaussians): Es un modelo de caracterización de los píxeles del fondo basados en el método de mezcla de Gaussianas. Se caracteriza porque tienen en cuenta a la hora de modelar el fondo, los posibles cambios de iluminación en la imagen, secuencias multimodales, objetos moviéndose lentamente, y el ruido de la cámara.
- **FGD** (del inglés Foreground Detection based on background modeling and Bayes classification): Propone un marco bayesiano para incorporar características espectrales, espaciales y temporales en el modelado del background. Deriva de una fórmula de la regla de decisión de Bayes para la clasificación de background y foreground. El background es representado usando estadísticas de las principales características asociadas con objetos de background estacionarios y no estacionarios.

Todos los algoritmos desarrollados para este propósito deben solventar problemas como el ruido, la exclusión de sombras y reflejos del foreground, cambios de iluminación, la actualización del fondo de escena (ya que el mundo no es estático) y la determinación de los parámetros de funcionamiento de los algoritmos.

#### *4.2.1.2. DETECCIÓN DE PERSONAS*

Una vez se realiza la extracción del foreground, el siguiente paso consiste en analizar la zona de alguna manera y determinar si es una persona o no. Este es un proceso complejo, debido a que no hay ningún modo fácil de definir la manera con que una persona entra en escena, ya que las personas adoptar un número indefinido de posturas y además ir modificándola según avanza la escena. Además este proceso debe dar una respuesta en tiempo real, por lo que el coste computacional no debe ser demasiado alto.

El análisis de la zona dinámica puede ser de distintos tipo:

- **Análisis de contornos**: Son muy convenientes cuando la postura de las personas es siempre la misma y el rango de tipos de personas a detectar es bajo.

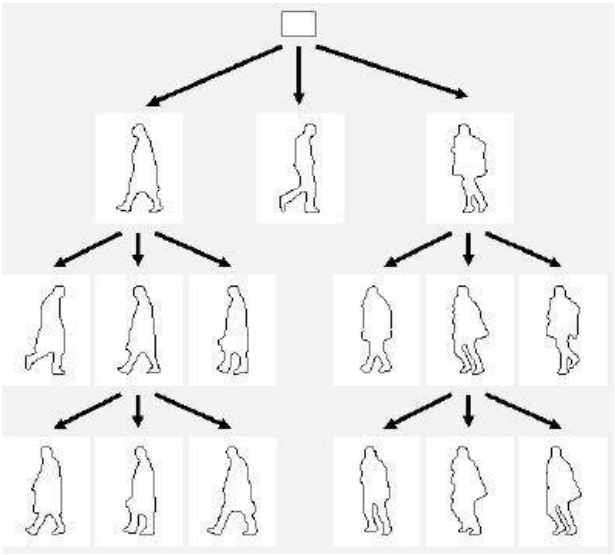

*Figura 29: Árbol de análisis de contornos*

- **Análisis de regiones**: Se calcula de modo iterativo la elipse más grande contenida en cada región obtenida en la etapa anterior.
- **Análisis de características**: Se analizan ciertas características intrínsecas de las personas, como pueden ser el área o las relaciones entre las medidas medias de un humano.

### 4.2.2. MOTION CAPTURE

La captura de movimiento (MoCap) es el proceso de registrar el movimiento de objetos o personas en tiempo real. Surgió como una herramienta de análisis para investigaciones biomecánicas en las décadas de 1970 y 1980 y posteriormente se extendió a campos como la educación, industria militar, entretenimiento deportes, aplicaciones médicas, visión artificial, robótica, cine y televisión...

Las personas que van a ser capturadas llevan marcas cerca de cada articulación para identificar el movimiento por posiciones o los ángulos entre ellas. Las marcas

pueden ser acústicas, inerciales, LED, magnéticas, reflectivas o combinación de ellas. A continuación comentaremos las distintas técnicas existentes para la captura de movimiento, las cuales se pueden dividir en sistemas ópticos y no ópticos.

#### *4.2.2.1. SISTEMAS ÓPTICOS*

Los sistemas ópticos utilizan datos capturados a partir de imágenes para triangular la posición 3D del sujeto entre una o más cámaras calibradas para proporcionar proyecciones superpuestas. La adquisición de datos se lleva a cabo tradicionalmente usando marcas especiales que el actor lleva colocadas. Sin embargo, los sistemas más recientes son capaces de generar datos precisos mediante el seguimiento de características de la superficie identificados dinámicamente para cada sujeto. Estos sistemas producen datos de 3 grados de libertad para cada marcador, y la información de rotación se refiere a la orientación relativa de tres o más marcadores (hombro, codo y muñeca proporcionan el ángulo del codo).

Sistemas híbridos más novedosos combinan sensores inerciales con sensores ópticos para reducir la oclusión, aumentar el número de usuarios y mejorar la capacidad de rastrear sin tener que limpiar manualmente los datos.

Las marcas utilizadas en este tipo de sistemas son:

- **Marcas pasivas**: Estas marcas están recubiertas con un material retrorreflectante para reflejar la luz que se genera cerca de las lentes de las cámaras. Puede ajustarse el umbral de las cámaras para que sólo los marcadores se muestreen, haciendo caso omiso de la piel y la tela.
  EL centroide del marcador se estima como una posición dentro de la imagen de dos dimensiones que se captura. Para calibrar las cámaras se utiliza un objeto con marcadores en posiciones conocidas. Con dos cámaras capturando la misma marca puede obtenerse la posición 3D de la misma.

Estos sistemas constan normalmente de 2 a 48 cámaras, y la velocidad de captura varía normalmente entre 120 y 160 fps.

*Figura 30: Sistema óptico de marcas pasivas en la película 'Avatar'*

- **Marcas activas**: Las marcas activas a diferencia de las pasivas, emiten su propia luz. Suelen ser LED que se iluminan, y esto aumenta la distancia y el volumen de captura. La forma de calcular su posición también es triangulándola, aunque existe la alternativa de utilizar algoritmos que requieren un tratamiento adicional de los datos.

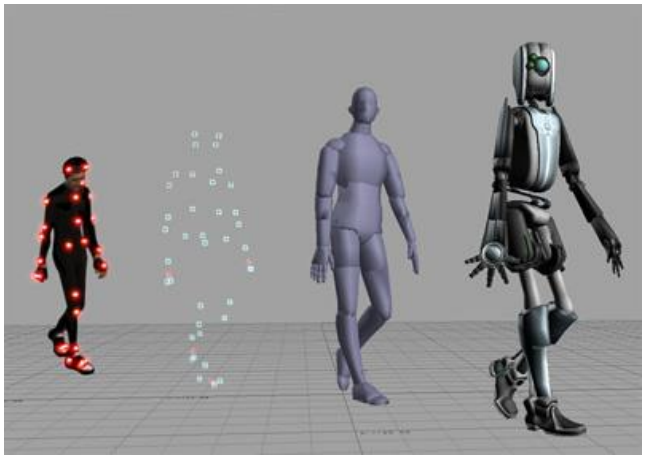

*Figura 31: Detección de movimiento con marcas activas*

- **Marcas Activas moduladas en el tiempo**: Los marcadores activos pueden refinarse por incidencia de luz estroboscópica, o capturar múltiples marcas a

través del tiempo y modular la amplitud y anchura del pulso proporcionando un identificador a las marcas. Estos identificadores reducen el intercambio de marcadores y suministran datos muchos más limpios que otras tecnologías.

LEDs con procesamiento a bordo y sincronización por radio permiten la captura al aire libre con luz del sol directa, logrando una captura desde 120 a 960 fps con un obturador electrónico de alta velocidad. Este sistema con 8 cámaras, 12 mpx de resolución y 120 Hz, para un actor, puede tener un coste de 20.000$.

#### *4.2.2.2. SISTEMAS NO ÓPTICOS*

Además de sistemas basados en imágenes, existen otros sistemas para capturar el movimiento. En este apartado vamos a comentar los más importantes:

- Sistemas inerciales: Está basado en sensores inerciales en miniatura, modelos biomecánicos y algoritmos de fusión de sensores. Los datos de movimiento de los sensores son, menudo, transmitidos de forma inalámbrica al ordenador, donde el movimiento es grabado o visualizado. Los sensores inerciales más utilizados son los giroscopios para medir la rotación. Estas rotaciones son trasladadas al esqueleto en el software.

  Al igual que con los marcadores ópticos, a más cantidad de giroscopios, captación de movimientos más naturales. Además capturan la totalidad de los seis grados de libertad de un ser humano en tiempo real. Las principales ventajas de este sistema son que no tiene son la portabilidad y la captación en zonas amplias. Y como puntos débiles podemos destacar la incomodidad para el sujeto, una menor precisión y una desviación agravada en el tiempo.

*Figura 32: Sistema inercial de MoCap en la película 'TED'*

- Movimiento mecánico: Esta técnica captura directamente los ángulos de cada articulación del cuerpo. Consiste en que el sujeto se acopla a una estructura esquelética, y cuando se mueve lo hacen las partes mecánicas articuladas, midiendo el movimiento relativo del actor. Este sistema trabaja en tiempo real, es de coste relativamente bajo, está libre de oclusiones y es inalámbrico. Las estructuras están formadas, en general, por varillas articuladas de metal o plástico unidas entre sí con potenciómetros que se articulan en las articulaciones del cuerpo, y tienen un coste de entre 25.000$ a 75.000$.

*Figura 33: Sistema mecánico de MoCap*

- Sistemas magnéticos: Los sistemas magnéticos calculan la posición y orientación por el flujo magnético relativa de tres bobinas ortogonales al

transmisor y a cada receptor. La salida del sensor es de seis grados de libertad, y proporciona datos útiles con sólo dos tercios de sensores de los necesarios en los sistemas ópticos. Los marcadores no se ocluyen por elementos no metálicos, pero son susceptibles a la interferencia magnética y eléctrica de objetos metálicos. Otro problema de este sistema es que el cableado del sistema impide movimientos extremos.

### 4.2.3. RECONOCIMIENTO DE POSES HUMANAS EN PARTES, POR JAMIE SHOTTON

Jamie Shotton y su equipo propone un método que predice con precisión y rapidez la posición 3D de las articulaciones de un cuerpo a partir de una imagen de profundidad, sin usar información temporal. Siempre buscando dos metas: robustez y eficiencia.

Todos los métodos anteriores mostraban bastantes limitaciones. Algunos conseguían buena respuesta temporal en el seguimiento frame a frame pero fallaba en la reinicialización, y por lo tanto en la robustez. Fue el lanzamiento de Kinect el que provocó un salto en el desarrollo de esta materia. La preinicialización y recuperación que propone Shotton pueden complementar cualquier algoritmo de seguimiento, mejorándolo.

El método está inspirado en el reconocimiento de objetos basado en dividir los objetos en partes. Consiste principalmente en dividir el cuerpo en partes, etiquetadas con probabilidades de que se encuentre en esa zona cada una de las articulaciones, posteriormente se decide gracias a un bosque de decisión y a una base de datos entrenada con una gran cantidad de imágenes y posturas, la posición de cada articulación.

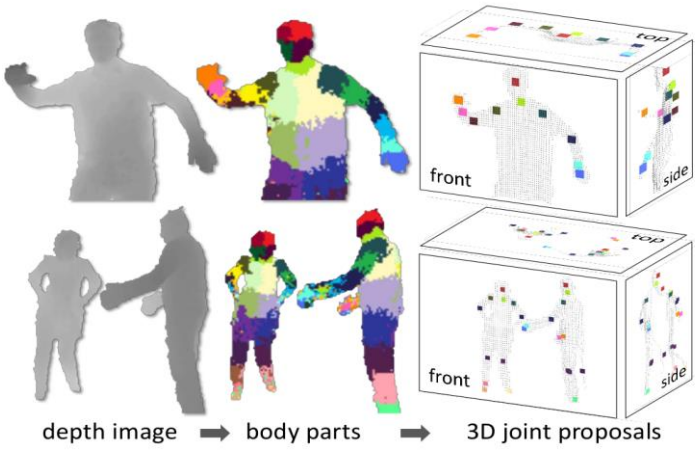


*Figura 34: División del cuerpo en partes, algoritmo de NITE*

El primer paso del método es la segmentación del cuerpo en partes. Este paso se trata como una clasificación pixel a pixel. Esto evita una búsqueda combinatoria sobre las diferentes partes del cuerpo, aunque esto hace que dentro de cada parte haya grandes diferencias contextuales.

Para el entrenamiento de la base de conocimiento, se generaron una gran cantidad de imágenes de profundidad sintéticas realistas de seres humanos de múltiples formas y tamaños y con una gran variedad de posturas, formando una base de datos muy grande. Además se ha implementado un bosque de decisión entrenado y reajustado con estas miles de imágenes.

*Figura 35: Figuras sintéticas para el entrenamiento del algoritmo de NITE*

Para calcular la posición de cada articulación, cuando llega un frame simplemente comparando las características de profundidad de las mismas se producen invariantes de traslación 3D, esto hace que se vaya recorriendo el bosque de decisión en profundidad hasta llegar a la decisión que por probabilidad es la más acertada, dicho de otra manera, es la posición que mayor probabilidad tiene de ser la de una cierta articulación.

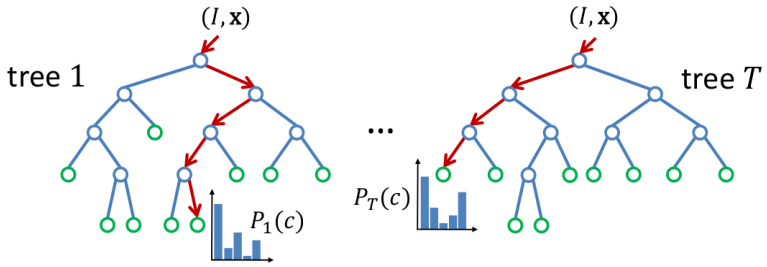


*Figura 36: Bosque de decisión del algoritmo de NITE*

Este proceso de decisión es muy rápido, por lo que se mantiene la eficiencia. Para mayor rapidez, esta comparación puede hacerse en paralelo con distintos píxeles en una GPU. Este proceso funciona a 200 fps en la GPU de la videoconsola Xbox360.

## 4.3. AFFORDANCES EN LA ROBÓTICA

Este proyecto no llega a tratar el tema de las *affordances* como se tenía pensado en un principio, pero asienta las bases para futuros proyectos. Las *affordances*, como veremos más adelante, son oportunidades de acción que el robot detecta en el medio que lo rodea. Estas oportunidades vienen determinadas tanto por los objetos de la escena, como por el plano físico y mental del sujeto. La detección de personas, así como el análisis de lo que está ocurriendo en una escena, detectadas la actitud o intenciones de las personas que en ella aparecen, pueden ofrecer al robot información útil de lo que le rodea, con el fin de determinar las opciones de acción

que tiene disponibles. Para entender mejor esta materia, que es el camino que se definió para este proyecto vamos a comentarla con más precisión y comenzando desde el principio, con la teoría psicológica fuera de la robótica.

### 4.3.1. PSICOLOGÍA ECOLÓGICA

Para la ciencia cognitiva tradicional [20], la cognición se explica mediante la "metáfora del ordenador": nuestra mente sería análoga a un mecanismo consistente en la manipulación de representaciones según reglas predeterminadas, ya sean sintácticas o algebraicas. Pese al desarrollo de nuevos modelos que supusieron para algunos una revolución en los años ochenta (redes neuronales artificiales, neurocomputación), esta metáfora siguió sirviendo para comprender los procesos mentales. Así estos modelos simplemente proporcionaron un rostro joven para esta vieja idea. Como consecuencia, los procesos perceptivos no son la fuente principal de la cognición, y adquieren un papel pasivo y desvinculado de la acción del agente.

Una de las tendencias que se han rebelado contra esta concepción es la psicología ecológica [21]. Fue concebida por J. J. Gibson como una manera de entender la naturaleza de los procesos perceptivos. Gibson defendió que los agentes conocen su medio explorándolo, actuando. Así pues, el comienzo del proceso cognitivo no es pasivo, sino activo. El objeto de la percepción no es una representación, sino un bucle o proceso continuo entre la acción del sujeto y la información del entorno. El agente recoge cierta información específica del entorno (visual, auditiva, etc.) que es relevante para su acción, guiándolo en su adaptación y en la consecución de sus fines. Las consecuencias metafísicas son inmediatas: el agente percibe directamente la realidad a través de sus acciones, sin necesidad de procesar los estímulos que recoge del medio. Por lo tanto tenemos tres nociones básicas que se contraponen al cognitivismo.

- La cognición no necesita de representaciones, cómputos o lenguajes de pensamiento.
- No es necesario un procesador central, sino que todo el proceso perceptivo está repartido por el sistema.
- La cognición está integrada en el medio.

Para definir el momento de la percepción, puesto que carecemos de representaciones y el sujeto aislado no es suficiente, Gibson acuñó el término *affordances*.

### 4.3.2. AFFORDANCES

Un *affordance* [21] es la relación epistémica entre el agente y su medio. Esta surge cuando el agente detecta una información específica que le permite modular su acción. Es decir, los *affordances* no son más que oportunidades para la acción. Cuando un humano percibe la "agarrabilidad" de una taza, ello constituye una relación epistémica con su entorno. Esto se da gracias a la información especificada por la taza y también por el hecho de que el humano sea activo y tenga pulgares oponibles que permitan agarrarla. La combinación de ambos elementos (información del objeto y del agente) permite una nueva acción, con el consiguiente mantenimiento del ciclo continuo percepción-acción.

Además de las cualidades físicas del agente, influye su experiencia y vivencias. Un animal o una persona se desenvuelven mejor en una zona ya conocida que en otra en la que nunca han estado, y esto es debido al aprendizaje y adaptación obtenidos gracias al continuo ciclo del que hablábamos con anterioridad.

### 4.3.3. INTELIGENCIA ARTIFICIAL ECOLÓGICA

El concepto de Inteligencia Artificial Ecológica [22] [23] surge de la idea de Psicología Ecológica comentado anteriormente. Y es que, ¿por qué un robot es capaz de simular el pensamiento humano en las tareas de mayor exigencia intelectual y sin embargo no tiene aquellas capacidades más naturales y cercanas al dominio de cualquier ser vivo? Si contamos con procesadores con una potencia mayor que la del cerebro humano, ¿por qué se presentan como importantes avances los robots que a duras penas consiguen subir cuatro escalones? Es decir, puede trabajar perfectamente con lenguajes abstractos, relaciones lógicas y complejos razonamientos matemáticos, incluso puede ganar jugando al ajedrez al mejor jugador del mundo y no puede reconocer patrones, desplazarse por el medio, navegar sin planes o ser autónomo. Esto es porque se ha construido una Inteligencia Artificial sobre los supuestos de que el razonamiento es reductible a computación simbólica y resolución de problemas, y esta visión no ha dado sus frutos.

El modelo tradicional de input - procesamiento- output hace que se quede mucha información útil en el camino. Parece, por tanto, que el camino hacia la autonomía robótica pasa por un ajuste online de las interacciones máquina-entorno. Estas capacidades parecen igual de modelizables y computables que el resto de operaciones que un ordenador puede realizar. Y no sólo computables, sino que también pueden ser mecanizadas. Éste es el núcleo del argumento planteado por Sverker Runeson (1977) para mostrar cómo los sistemas perceptivos pueden ser, en sí mismos, inteligentes, en tanto que no necesitan de elaboración cognitiva posterior para estimar aquellas magnitudes que le son relevantes dada su naturaleza. Percibir, por ejemplo, si un objeto está o no dentro de nuestro alcance es una tarea lo bastante relevante para nosotros como para que se realice de una forma mucho más rápida y precisa que cualquier otra estimación espacial geométrica. Percibir dicha propiedad es percibir una posibilidad para la acción, una *affordance*.

Estar abiertos continuamente a la revisión de la conducta y que no exista un plan concreto preparado, sino que éste sea una construcción que se establece a lo largo de dicha interacción, posibilita mayor armonía con el entorno.

Conseguir autonomía en una nueva generación de robots y en una nueva I.A. implica que éstos estén abiertos a continua retroalimentación y plasticidad, escapando así del modelo tradicional de procesamiento y elaboración de estrategias que ha funcionado bastante bien para jugar al ajedrez y operar con matrices pero que no ha permitido avances en el control motor. La autonomía resultante de un enfoque como éste es el de una propiedad emergente de un organismo coordinado con su entorno, de tal manera que su capacidad de adaptación al mismo le confiere continuamente nuevas y revisables oportunidades para la acción.

El núcleo de la propuesta ecológica pasa por un realismo que asume la existencia de patrones organizados de información en el ambiente que especifican propiedades físicas de dicho entorno. Los seres vivos, como sistemas acoplados con su ecosistema, tienen un soporte físico adecuado para detectar dicha información estructurada sin necesidad de realizar transformaciones computacionales complejas. La robótica basada en dicho enfoque generará dispositivos sensibles a información ecológicamente relevante para la naturaleza del artefacto diseñado, intentando tener siempre una inspiración biológica.

En resumen, podríamos establecer que los principios que guían la Inteligencia Artificial Ecológica implementables en robótica son:

- Adaptación. Esta inteligencia artificial permite la estabilización de comportamientos adaptativos para los robots.
- Minimalismo: Se busca reproducir con la menor cantidad de procesamiento posible comportamientos biológicamente inspirados. Esto implica evitar una excesiva centralización de tareas, pudiendo diseñar en paralelo tareas más simples.

- Dependencia del contexto: La dependencia del contexto permite ganar en flexibilidad a los autómatas ecológicos El agente y su entorno están en continua confluencia. Es el entorno y no una estrategia inicial lo que debe modelar el comportamiento.
- Flexibilidad: Será producto de la dependencia del contexto y la ausencia de un gran procesamiento central.

## 4.4. REPRESENTACIÓN SOFTWARE DE UNA ESCENA

La forma en la que representar los datos físicos de una escena en un sistema software es importante. Son datos con los que se interactúa continuamente, por lo que la forma de almacenarlos tiene que permitir un acceso rápido, ordenado y que a la vez sea lo más simple posible. Además debe ser lo bastante versátil como para que pueda representar cualquier escena que pueda darse

En InnerModel, como ya se trató en puntos anteriores, la escena se define como un árbol cinemático definido en un fichero XML. Ahora veremos qué consecuencias tiene esto, qué ventajas y también qué desventajas. Además hablaremos de las alternativas más comunes que se tienen a esta representación.

### 4.4.1. ÁRBOL CINEMÁTICO

Un árbol cinemático representa la escena de forma jerárquica. Cada nodo define una parte de la escena, y los enlaces entre nodos definen relaciones entre ellos en el mundo real. Cada objeto del mundo, o cada persona pueden estar definidos por varios nodos, es decir, en la mayoría de los casos son subárboles del árbol principal, y no nodos únicamente. Por ejemplo, el tren superior de una persona podría definirse de la siguiente manera:

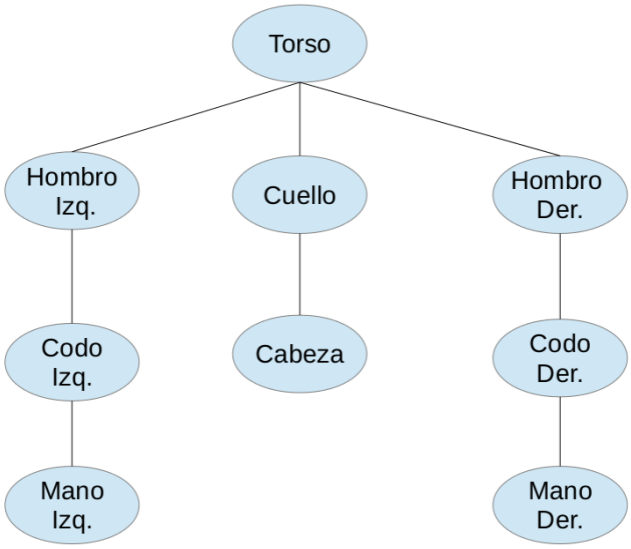


*Figura 37: Árbol cinemático del tren superior humano*

Con esta representación, la modificación de un nodo afecta a todos sus descendientes. Por ejemplo, en el árbol de la [Figura 37] un cambio en la posición del torso de una persona afecta también a los brazos y a la cabeza, o el giro de un hombro implica el cambio de posición del codo y la mano del mismo brazo.

En cuanto a los nodos, ya dijimos que aplicando herencia pueden ser de varios tipos, y esto ya les da un significado concreto. No obstante los nodos poseen atributos que los definen con más precisión. Existen atributos comunes a todos, como son la posición y la rotación en cada eje de coordenadas X, Y, Z. Estos datos son relativos con respecto al padre, lo que permite saber rápidamente la posición de un nodo

tomando como origen otro. Y también existe la posibilidad de definir otros atributos de forma individual que ofrezcan cualquier información específica. Podría definirse por ejemplo, el nombre y apellidos de una persona en su primer nodo, el color de la camiseta en el torso, o el tipo de material en un nodo que representen objetos.

Como hemos podido comprobar es una estructura muy versátil, pero tiene un problema que se da en situaciones más o menos frecuentes, y es que no admite ciclos. Al no admitir ciclos, hay escenas que no pueden representarse en su totalidad. Veamos un ejemplo de ello:

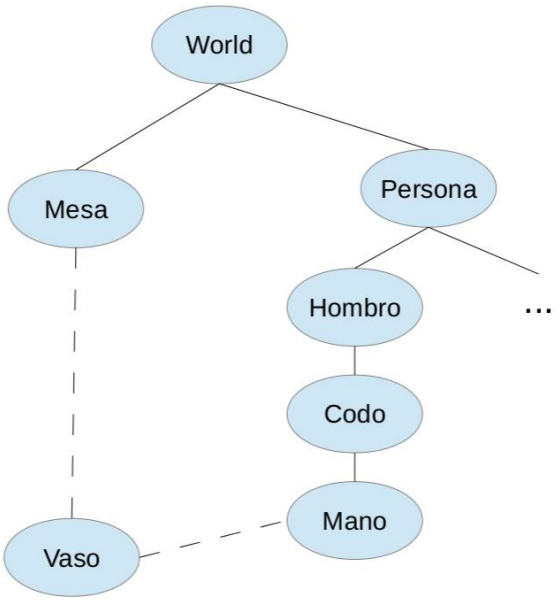


*Figura 38: Problema de ciclos en árbol cinemático*

Imaginemos una escena en la que hay una persona, una mesa y una taza. La taza está encima de la mesa, y a su vez está agarrada por la persona. En cuanto a su representación sería lógico pensar que el vaso debe ser descendiente del nodo mesa, pero también debería ser descendiente del nodo mano de la persona. Y no puede definirse en ambos sitios pues representaría dos objetos distintos, ni tampoco ser descendiente de ambos a la vez, porque se formaría un ciclo

La solución a este problema es utilizar la estructura por excelencia, el grafo.

### 4.4.2. GRAFO CINEMÁTICO

La idea del grafo cinemático es la misma que la del árbol, pero como dijimos antes, con la excepción de que pueden definirse ciclos. La escena anterior quedaría representada como indicamos en la siguiente figura:

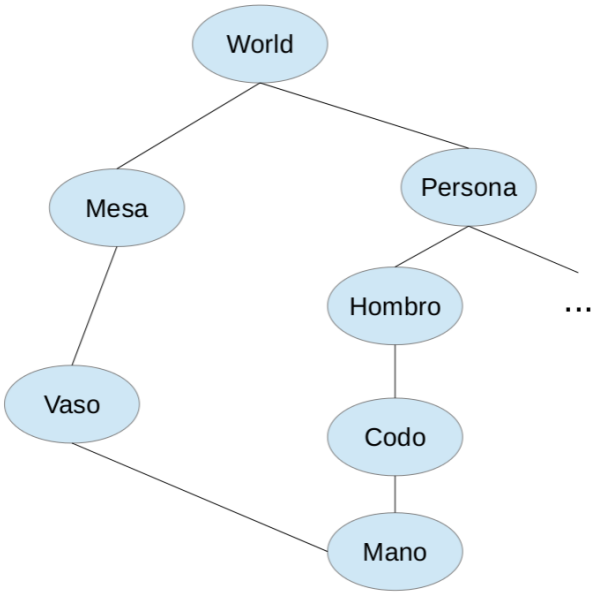


*Figura 39: Grafo cinemático de una escena con ciclo*

De esta manera puede deducirse fácilmente que el vaso se encuentra tanto en la mesa como en la mano de la persona. Y por consiguiente es posible definir cualquier tipo de escena, y tan compleja como sea necesario.

Con estas estructuras, que contienen información sobre la forma de la escena se suelen desarrollar soluciones para cálculos físicos, que permiten mover la escena con cierto criterio. Por este motivo, el grafo cinemático puede no ser suficiente para ciertos requerimientos.

Por ejemplo, si queremos detectar colisiones entre objetos, trayectorias, etc. debemos tener más información acerca de la escena. Es por ello, que se definió el brazo con enlaces.

### 4.4.3. GRAFO CON ENLACES

Este grafo se basa en la idea base de las estructuras anteriores, pero introduce más información a partir del concepto de enlace. Los enlaces definen las características de la unión de nodos, que en las estructuras anteriores, eran tomadas por defecto como sólidos rígidos. Para explicar esto, analicemos la siguiente figura:

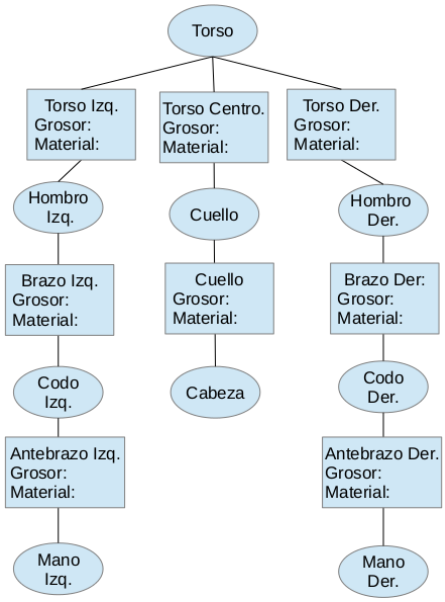


*Figura 40: Ejemplo de grafo con enlaces del tren superior humano*

Como vemos, es un grafo de la forma nodo-enlace-nodo. En este caso los enlaces definen los miembros de una persona o de un robot con forma humana. Podría definirse tanto un brazo humano como uno metálico, y con distintas formas. Dada esta información se puede calcular el comportamiento de distintos objetos, de distintos materiales y distintas formas interactuando entre sí. Lo que otorga a esta estructura una nueva dimensión mucho más compleja que las anteriores.

En cualquier caso, la elección de una u otra forma de representación depende de los propósitos que se pretendan, pues aunque el grafo con enlaces es la estructura más completa, también es la más compleja, y no tiene porqué ser siempre la mejor opción.

# 5. DISEÑO Y DESARROLLO DEL SISTEMA

## 5.1. PLANIFICACIÓN DEL PROYECTO

El proyecto consistirá en la implementación de dos componentes que se añadirán al repositorio de RoboComp. El objetivo final del sistema será detectar las intenciones de las personas que se encuentren delante de una cámara RGBD, analizando su postura.

El primero de los componentes es OpenNI2Comp, será los ojos del robot, realizará las funciones más importantes de la librería OpenNI2 y NITE2, envolviéndolas en el ámbito de RoboComp. Hará la captura de los flujos de color y profundidad, pudiendo crear archivos *.oni con ellos y reproducirlos. Además hará el seguimiento esqueletal de los usuarios de la escena.

Un fichero *.oni es un archivo de video que almacena el flujo de color y de profundidad captado por la cámara RGBD, pudiendo reproducirse posteriormente, obteniendo los datos de cada flujo por separado. La capacidad del componente de poder reproducir este tipo de archivos es importante, pues teniendo los videos se puede prescindir de la cámara RGBD.

El segundo componente, PoseDetectorComp, hará las veces de mente del robot, y tomará decisiones a partir de los datos proporcionados por OpenNI2Comp. Este componente conocerá algunas posiciones corporales que asociará a intenciones humanas. Su objetivo será detectar si alguna de las personas tiene alguna postura conocida que le permita determinar su intención.

Además inyectará los datos obtenidos en el RCIS para que podamos ver, en una escena tridimensional la percepción que tiene el robot de los datos que está procesando.

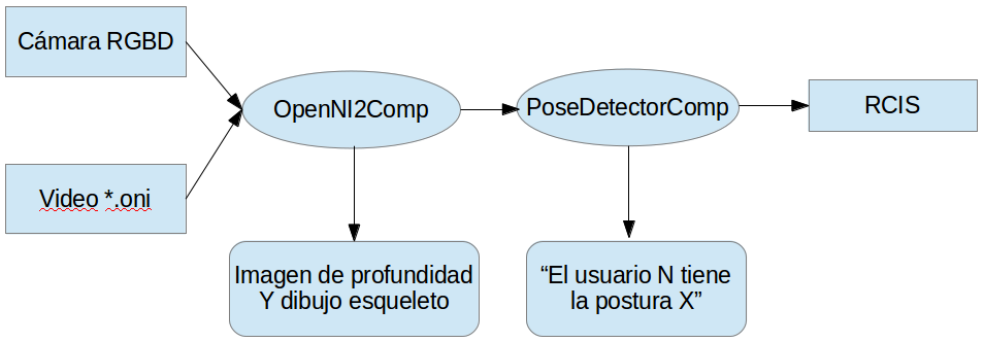


*Figura 41: Esquema de funcionamiento general del sistema*

En la imagen podemos ver un esquema general del sistema a muy alto nivel de abstracción. Los pasos, en este nivel de detalle, son claros y sencillos:

- OpenNI2Comp:
  - Obtención de flujos de color y profundidad proporcionados por una cámara RGBD o un video *.oni, y seguimiento esqueletal. En la interfaz gráfica del componente se mostrará la imagen de profundidad y el esqueleto del tracking.
  - Transformación y adecuación del conjunto de datos obtenidos al framework RoboComp.
- PoseDetectorComp:
  - Inyección de los datos adaptados en el simulador RoboComp InnerModel Simulator.
  - Detección de intenciones humanas a partir de los datos del seguimiento adaptados y aviso a través de la interfaz.

## 5.2. CREACIÓN GENÉRICA DE UN COMPONENTE

Antes de proceder con la especificación de los componentes construidos se mostrará cómo crear un componente genérico. Cuando se incluye un nuevo componente en RoboComp se usa generalmente una herramienta que genera automáticamente (DSL Editor) un esqueleto con las funcionalidades básicas, por

supuesto, el generador de código también incluye el código necesario para crear una conexión a los componentes que se le especifique y puede volver a ser generado y adaptado cuando sea necesario.

Además, a los componentes que se generen automáticamente se le pueden añadir después todas las clases que sean necesarias. Además, muchas veces se desea añadir en el fichero de configuración algún parámetro específico y, seguramente, modificar el fichero de interfaz por defecto (que no ofrece ningún método). De no usar el generador automático de código, la creación de componentes, incluso pequeños, sería bastante tediosa, y susceptible de errores ya que es necesario integrar el código del middleware Ice y el código propio del componente.

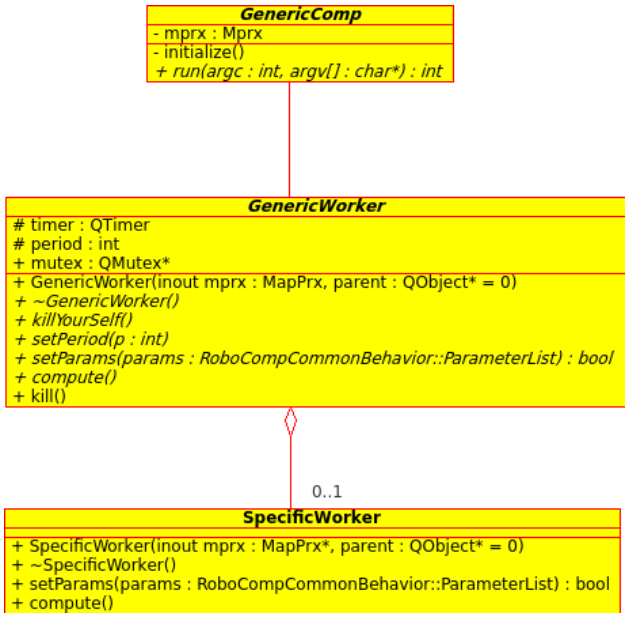


*Figura 42: Diagrama de clases de un componente genérico*

Como se puede observar en la ilustración, además del código de programa principal, el componente plantilla consta de tres clases: GenericComp, GenericWorker y SpecificWorker. Cuando se usa el generador de código éste modifica los nombres de la clase GenericComp en función al nombre del componente asignándole a esta clase el nombre que se le haya dado al componente.

Cada componente consta de, al menos, dos hilos de ejecución, siendo estos el hilo principal y un hilo separado para responder a las llamadas Ice. El último de estos dos hilos se crea cuando se instancia una interfaz Ice (a la clase creada se le tiene que pasar como parámetro la clase que se hace cargo de las llamadas). El hilo principal de ejecución se encarga de la parte activa del componente, de hacer las llamadas Ice necesarias si las hay, y los cálculos asociados al comportamiento del componente.

Para facilitar la comprensión del código la clase GenericComp delega en SpecificWorker todo el trabajo y GenericWorker queda como una clase que, de forma transparente y asíncrona, responde a las llamadas remotas.

Las clases SpecificWorker y GenericWorker, que como ya hemos visto se encargan de la parte interna y externa respectivamente, se ejecutan en exclusión mutua gracias a un mutex que comparten. El bloqueo del mutex se realiza cada vez que GenericWorker recibe una llamada o que SpecificWorker va a cambiar alguna parte de su estado que pueda modificar las respuestas de GenericWorker a sus invocaciones remotas.

Como se puede ver en el diagrama, GenericWorker tiene acceso a la clase SpecificWorker.

Dicha condición, que sólo se da en ese sentido, es necesaria porque generalmente toda la información relativa al estado del componente se guarda dentro de SpecificWorker.

La clase SpecificWorker, como dijimos anteriormente es en la que está delegada todo el trabajo. No puede ser regenerada, pues a partir de ella se crea el código que implementa las funciones específicas del componente, aunque como hemos comentado pueden crearse tantas clases de apoyo como sean necesarias. En ella está el método principal del componente, *compute()*, que es el que se ejecuta

continuamente. Es esta clase con la que más trataremos en este proyecto, no siendo necesaria la implementación de ninguna otra.

## 5.3. OPENNI2COMP

El primer componente del sistema desarrollado en este proyecto se podría definir como un wrapper desarrollado en forma de componente para RoboComp, de forma que este framework ofrece gran parte de la funcionalidad de las librerías OpenNI2 y NITE2. Por lo tanto, este componente se encarga de la captura de los flujos RGB y de profundidad (D) así como del tracking de personas, y de la transformación de estos datos a la forma con la que se trabaja en RoboComp.

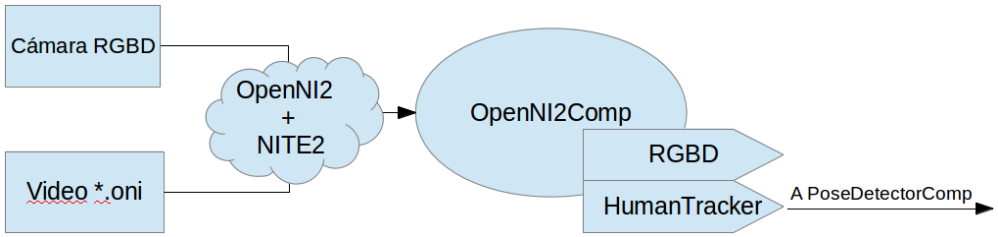


*Figura 43: Esquema de funcionamiento de OpenNI2Comp*

En los siguientes apartados vamos a ir desgranando este componente analizando todos los detalles importantes.

### 5.3.1. CMAKE PARA OPENNI2 Y NITE2

Puesto que los componentes en RoboComp se construyen a partir de ficheros CMake, el primer paso para el desarrollo de este componente era crear los ficheros cmake de OpenNI2 y NITE2 para poder trabajar con estas librerías en el componente.

La filosofía de CMake es la división del trabajo en varios ficheros reutilizables. De esta manera tendremos un fichero principal llamado CMakeLists.cmake, en el que,

entre otras cosas, haya llamadas a otros ficheros *.cmake que contendrán el código para compilar cada una de las librerías que necesitemos.

Ya que las nuevas versiones de OpenNI 2 y NITE 2 nunca habían sido utilizadas en RoboComp, nuestro primer objetivo era crear dos ficheros openni2.cmake y nite2.cmake de uso general, de forma que como hemos comentado, una llamada a estos bastará para poder utilizar las librerías en el componente que deseemos. De la misma forma, teniendo estos dos ficheros podrían incluirse estas librerías en cualquier componente que construyamos a través del creador de componentes de RoboComp DSLEditor con un simple click de ratón, ya que son añadidos como módulos a través de su GUI. Estos ficheros han sido incorporados al núcleo del repositorio de RoboComp y se pueden encontrar en la ruta robocomp/Cmake.

### 5.3.2. INTERFACES IMPLEMENTADAS

Es importante hacer una buena elección de las interfaces a utilizar, ya que será la forma de comunicación entre nuestro componente y el mundo. Una interfaz .ice en RoboComp podría entenderse, de manera simplificada, como un .h en la programación orientada a objetos.

Una mala interfaz equivale a mala comunicación y por lo tanto, con toda probabilidad a poca funcionalidad del componente. OpenNI2Comp implementa dos interfaces, cada una podría asociarse con cada una de las dos librerías principales que utilizamos.

#### *5.3.2.1. INTERFAZ RGBD*

La primera interfaz es la interfaz RGBD. Será la encargada de dar salida a los flujos de color y profundidad de la forma correcta. Esta interfaz ya estaba diseñada, nuestro cometido ha sido reimplementarla para que los datos que devuelve sean los obtenidos por un dispositivo RGBD genérico, de esta forma cualquier componente que se conecta actualmente a esta interfaz, podrá seguir haciéndolo, de manera transparente y sin cambiar nada en su código, si decide hacerlo a través de Openi2Comp .

La interfaz también permite configurar el modo de registro de la cámara (None, DepthInColor, ColorInDepth) y obtener algunos parámetros relativos a la captura de vídeo:

- Altura y anchura de las imágenes de color y profundidad
- Tamaño en bytes de las imágenes de color y profundidad
- FPS (frames por segundo) de cada uno de los flujos
- Periodo
- Otros parámetros referentes a la estructura del robot ajenos a este proyecto.

```
struct CameraParameters
  {
     int focal;
     int width;
     int height;
     int size;
     int FPS;
  };


  struct TRGBDParams
  {
     CameraParameters color;
     CameraParameters depth;
     int timerPeriod;
     bool talkToBase;
     bool talkToJointMotor;
     string driver;
```

```
    string device;
};
```

Para dar forma a los datos que la interfaz ofrece se define un punto de color como una estructura con tres bytes que contiene los valores de rojo, verde y azul del punto; y el punto 3D está definido como una estructura de cuatro valores reales. Con estas estructuras, se definen los flujos de color y profundidad como secuencia de puntos de color y valores reales respectivamente. Además se describe el tipo PointSeq como secuencias de puntos 3D, y las imágenes de color y profundidad como secuencia de bytes y números reales:

```
sequence<byte> imgType;
sequence<float> depthType;

 struct ColorRGB {
     byte red;
     byte green;
     byte blue;
  };

  struct PointXYZ {
     float x;
     float y;
     float z;
     float w;
  };

  sequence<float> DepthSeq;
  sequence<ColorRGB> ColorSeq;
  sequence<PointXYZ> PointSeq;
```

Por otro lado, los métodos que ofrece la interfaz son los siguientes:

```
TRGBDParams getRGBDParams( );


void setRegistration (RoboCompRGBD::Registration value);


Registration getRegistration ( );


void getData(imgType& rgbMatrix, depthType& distanceMatrix,
RoboCompJointMotor::MotorStateMap &hState,
```

```
RoboCompDifferentialRobot::TBaseState& bState);


void getDepthInIR(depthType& distanceMatrix,
RoboCompJointMotor::MotorStateMap &hState,
RoboCompDifferentialRobot::TBaseState& bState);


void getImage(ColorSeq& color, DepthSeq& depth, PointSeq&
points, RoboCompJointMotor::MotorStateMap &hState,
RoboCompDifferentialRobot::TBaseState& bState);


void getDepth(DepthSeq& depth,
RoboCompJointMotor::MotorStateMap &hState,
RoboCompDifferentialRobot::TBaseState& bState );


void getRGB(ColorSeq& color, RoboCompJointMotor::MotorStateMap
&hState, RoboCompDifferentialRobot::TBaseState& bState);


void getXYZ(PointSeq& points,
RoboCompJointMotor::MotorStateMap &hState,
RoboCompDifferentialRobot::TBaseState& bState);
```

Como vemos, la interfaz RGBD ofrece salida para los datos que este componente desea distribuir, además de otras funcionalidades con las que OpenNI2Comp no entra en juego. A pesar de esto último pensamos que esta interfaz se adapta muy bien a lo que se buscaba, por lo que la hemos reimplementado. Esto nos sirvió para depurar errores en fases tempranas de su desarrollo, ya que integrada con otras herramientas de RoboComp, nos permitió comprobar que el trabajo realizado era correcto, por ejemplo, gracias a la herramienta RCMonitor, pudimos comprobar que la captura de los flujos de datos RGB y D eran correctas y que se realizaba a 30 frames por segundo, con un consumo de CPU aceptable, estas frecuencias son fundamentales, como dijimos, para las aplicaciones en tiempo real buscadas en la robótica.

#### *5.3.2.2. INTERFAZ HUMANTRACKER*

La interfaz HumanTracker se encarga de dar salida para RoboComp al seguimiento de personas que ofrece NITE2. HumanTracker ha sido diseñada y definida en este proyecto, ya que no existía ninguna que nos ofreciera las posibilidades que buscamos. Se ha implementado una interfaz sencilla y suficiente, con la expresividad adecuada y con la versatilidad para ofrecer los datos de manera que el usuario que lo necesite le pueda sacar todo el jugo a esta parte de la librería OpenNI2. Se puede obtener desde la lista de usuarios completa con todos sus datos, a la posición de una articulación concreta de un usuario concreto.

Para empezar, se ha definido una estructura de datos TPerson que contenga la información del seguimiento de cada usuario. Esta estructura contiene el estado del usuario, una lista con la posición de las articulaciones identificadas por el nombre de cada una, siendo el origen de coordenadas el dispositivo; y una lista de matrices de transformación, que contienen las posiciones y orientaciones relativas de cada articulación, es decir, tomando como origen de coordenadas de cada articulación el nodo de la que ésta cuelga. Ésta última lista es la más útil, puesto que es la forma con la que RoboComp trabaja.

Los estados por los que puede pasar una persona y que coinciden con los ofrecidos por NITE son:

- None: Estado cuando no hay ninguna persona.
- Calibrating: Estado en el que NITE está escalando el modelo al tamaño de la persona.
- Tracked: Estado en el que se encuentra una persona cuando se le está haciendo correctamente el seguimiento.
- ErrorNotInPose: Estado de error que ocurre cuando la persona no está en la posición indicada, según NITE2.
- ErrorHands: Estado de error provocado por las manos del usuario
- ErrorHead: Estado de error provocado por la cabeza del usuario

- ErrorLegs: Estado de error provocado por las piernas del usuario
- ErrorTorso: Estado de error provocado por el torso del usuario

También se ha definido una lista de personas, ya que el seguimiento es multiusuario, en el que cada uno se define con el identificador que NITE le asigna.

```
enum TrackingState {None, Calibrating, Tracked,
ErrorNotInPose,ErrorHands, ErrorHead, ErrorLegs, ErrorTorso
};

sequence <float> RTMatrix;
dictionary<string, RTMatrix>RTMatrixList;

sequence <float> joint;
dictionary<string, joint>jointListType;

struct TPerson{
   TrackingState state;
   jointListType joints;
   RTMatrixList rotations;
};

dictionary<int, TPerson> PersonList;
```

Como hemos dicho antes, la interfaz HumanTracker ofrece un amplio abanico de formas para obtener el dato que el usuario desee, esto puede verse en los métodos definidos:

```
void  getJointsPosition(int id, out jointListType jointList);

void  getRTMatrixList(int id, out RTMatrixList RTMatList);
void  getUserState(int id, out TrackingState state);
void  getUser(int id, out TPerson user);
void  getUsersList(out PersonList users);
```

Se han diseñado e implementado métodos para obtener la lista de posiciones absolutas, la lista de matrices de transformación o el estado de un usuario dado; todos los datos de un usuario o la lista de usuarios con todos sus datos.

### 5.3.3. FUNCIONAMIENTO

En este apartado comentaremos cómo se ha implementado cada una de las funcionalidades del componente, por qué se ha hecho de esta manera, y cómo se han solucionado los problemas encontrados.

#### *5.3.3.1. OBTENCIÓN DE FLUJOS*

El primer cometido de este componente es obtener los flujos de color y profundidad de la cámara RGBD o de un archivo de video *.oni. Para ello se han creado una serie de métodos privados que hacen esta función.

Los pasos que seguimos son los siguientes:

**Inicialización**

1) Abrir el dispositivo (cámara o archivo oni) con las funciones que ofrece OpenNI2.
2) Inicializar los flujos de color y profundidad. Esto es, crearlos y arrancarlos, con las funciones de OpenNI2. Además hay que inicializar las estructuras en los que se almacenarán con los tamaños requeridos.

Todos los errores que se pueden dar en la inicialización están controlados, abortando la ejecución del componente en errores críticos.

**Ejecución**

3) Detección de la llegada de un frame. Una función, que actúa como un "listener", espera a que llegue un frame de cualquiera de los dos flujos. Cuando detecta la llegada de uno, manda una respuesta indicando el flujo por el que ha llegado.

4) Lectura de un frame. Dependiendo de la respuesta recibida se lanza la función de lectura de flujo RGB o la de lectura de flujo de profundidad, y se lee el frame indicado.
5) Bloqueo con un mutex de la lectura de datos del frame que vamos a sobrescribir para asegurar la consistencia de los mismos.
6) Se hace una copia del bloque de memoria en el que se ha leído el frame, en el bloque de memoria donde está la estructura que devuelve la interfaz. Se hace de esta manera para ganar en eficiencia y rapidez, ya que es importante que este paso se haga rápidamente, para que los hilos externos estén bloqueados el menor tiempo posible.
7) Se desbloquea el mutex.

#### 5.3.3.1.1. IMAGEN DE PROFUNDIDAD COMO RGB

El flujo de profundidad no es más que una secuencia de distancias. Es necesario mostrarla de alguna manera que podamos entenderla. Para ello, hemos implementado una función que crea una imagen en escala de grises a partir de la imagen de profundidad. Esto se realiza mediante un mapeo asignando un color dependiente de la profundidad, siendo las distancias más largas los tonos más claros y las más cercanas lo más oscuros.

*Figura 44: Imagen de profundidad en escala de grises*

Se han utilizado para esta implementación funciones de la librería IPP por su rapidez de cómputo y eficiencia.

Sobre esta imagen, una vez hecho el seguimiento de las personas, pintamos sus esqueletos. Para ello tenemos que proyectar las posiciones 3D ofrecidas en el seguimiento en posiciones de la pantalla 2D, para que la imagen que ofrecemos sea coherente. Para ello utilizamos funciones de conversión de datos que ofrece NITE2, correctamente adaptadas a nuestras necesidades, ya que dibujamos con la clase RCDraw desarrollada en RoboComp y que permite dibujar figuras geométricas básicas con QT.

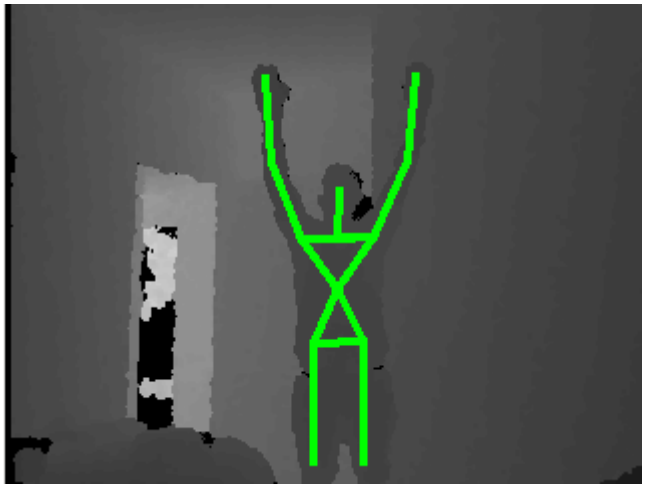

*Figura 45: Dibujo de un esqueleto sobre una imagen de profundidad en escala de grises*

#### *5.3.3.2. TRACKING DE PERSONAS*

Como hemos dicho anteriormente, el seguimiento de personas lo haremos con la librería NITE2. Este middleware ofrece funciones que hacen que esta tarea compleja se convierta en algo sencillo. Al igual que en la obtención de flujos el seguimiento de personas podemos dividirlo en dos fases, cada una divisible en varios pasos:

**Inicialización**

1. Inicialización de la librería NITE2 y creación del tracker del cuerpo completo asignándole como fuente de información el dispositivo creado en la fase de inicialización de la obtención de flujos.
2. Inicialización de las estructuras que contendrán los datos del seguimiento. En este paso se crean todas las listas (articulaciones y orientaciones) y se inicializan a valores nulos, para que posteriormente en la fase de ejecución únicamente haya que modificar datos y no crear y destruir objetos. De esta manera el sistema mejora su eficiencia. Además se crea un diccionario para transformar el tipo enumerado de NITE2 que identifica a las articulaciones en esta librería en cadenas de caracteres que los identifiquen en nuestro sistema.

**Ejecución**

3. Lectura de un frame por parte del tracker creado.
4. Obtención de usuarios del frame en una lista A. Este paso se realiza con una función de NITE2. Una vez obtenida la lista la recorremos, introduciendo en una lista interna los usuarios nuevos, lanzando su seguimiento esqueletal, y actualizando los datos de los que ya existían.
5. Cálculo de la posición de la articulación cintura, que definimos como el punto central entre ambas caderas y el torso. El valor de confianza de esta articulación será la media de la confianza de las tres articulaciones nombradas.
6. Cálculo de las posiciones y orientaciones relativas en radianes de cada articulación. Este paso lo explicaremos con más detalle en el apartado [5.3.3.2.1].
7. Bloqueo de una lista de usuarios B con un mutex
8. Copiado de la lista A en la lista B.
9. Desbloqueo del mutex.

Los tres últimos pasos se han hecho de esta manera por eficiencia. Podríamos haber bloqueado la lista A mientras se va actualizando según los usuarios obtenidos por NITE2, pero el tiempo de bloqueo sería mayor. La mejor solución era tener dos listas. La lista A se actualiza con los datos de los usuarios que ofrece NITE2, una vez la lista A está completa se bloquea la lista B para copiarle los datos de la primera. Esta segunda lista es la que se devuelve al hilo externo que la pida. De esta manera tenemos disponible una lista válida en casi todo momento, sólo tenemos el bloqueo de lectura activo durante el tiempo que dura una copia y no durante un procesamiento de datos.

#### 5.3.3.2.1. DE NITE2 A ROBOCOMP

En este apartado describiremos los cambios que hemos realizado para adaptar los datos ofrecidos por NITE2 a los requerimientos de RoboComp. Para ello vamos a citar las diferencias:

- Las posiciones que ofrece NITE2 de cada articulación son tomando como origen de coordenadas la posición del sensor, y RoboComp para cada articulación toma como origen de coordenadas la posición de la articulación de la que cuelga (hombro desde el torso, codo desde el hombro…)

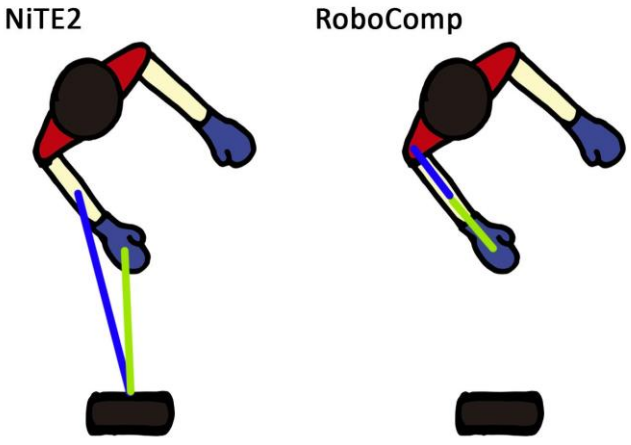


*Figura 46: Posiciones de NITE2 vs traslaciones de RoboComp*

Vemos como ejemplo las posiciones del codo y la mano del usuario que ofrece NITE2 y las posiciones de las mismas articulaciones que necesitan RoboComp e InnerModel.

- Con las orientaciones ocurre lo mismo que con las posiciones. Además existen dos diferencias más, y es que NITE2 ofrece las orientaciones en cuaterniones y RoboComp trabaja con radianes y RoboComp para el sentido de las orientaciones utiliza la regla de la mano izquierda [11] y NITE2 la regla de la mano derecha [12].

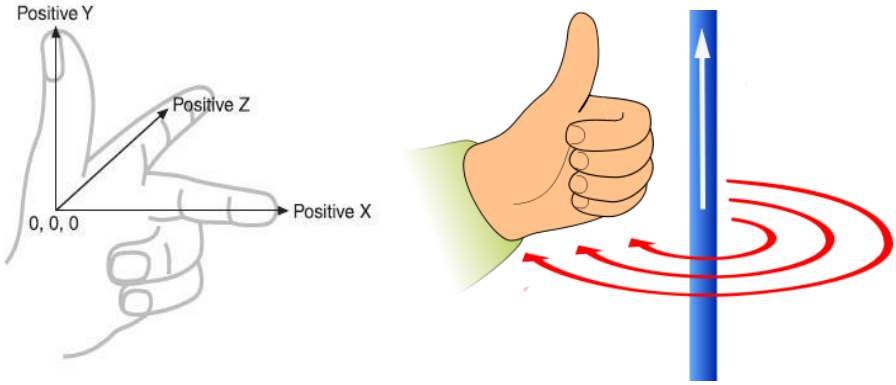


*Figura 47: Regla de la mano izquierda*

- Para NITE2 todas las articulaciones tienen el mismo eje de coordenadas, (+X a la derecha, +Y hacia arriba y +Z en el sentido que aumenta la distancia al sensor). En RoboComp cada articulación tiene su sistema de coordenadas, siendo el eje +Z el que se alinea con el hueso de la extremidad que forma.

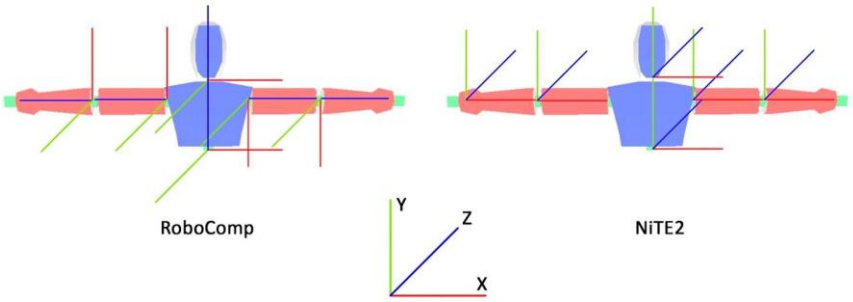


*Figura 48: Sistemas de coordenadas de NITE vs RoboComp*

En la imagen podemos ver la diferencia que comentamos. Los ejes de coordenadas en NITE2 son todos iguales (y están expresados desde el sensor), hecho que no ocurre en RoboComp, donde cada articulación tiene un sistema de referencia distinto que depende del padre y que sigue el criterio de alinear el eje Z en sentido del hueso mediante rotaciones. Esto se debe a razones prácticas en los cálculos relativos a la cinemática inversa.

Como vemos los datos que obtenemos y los que tenemos que ofrecer son muy dispares, y no existen funciones en la librería NITE2 que conviertan estos datos al formato que necesitamos, por lo que es necesario realizar algunos cálculos matemáticos para adaptarlos.

Para hacer la conversión que necesitamos utilizamos las posiciones absolutas de las articulaciones (con origen de coordenadas el dispositivo), para calcular las posiciones y orientaciones relativas (con origen de coordenadas la articulación padre). A partir de operaciones matemáticas de transformación que veremos a continuación, se calculan las matrices de transformación, que contienen estos datos y posteriormente se los asignamos adecuadamente a las personas creadas en la lista en el seguimiento.

Las matrices de transformación son matrices que definen la rotación de un punto y su traslación respecto a otro. A partir de estas se puede calcular la posición de dicho punto una vez se ha realizado la rotación de los ejes que dicta la matriz. Se pueden definir simplificadamente de la siguiente manera:

$$\begin{bmatrix} R(\alpha,\beta,\gamma) & T_T \\ 0 & 1 \end{bmatrix}$$

*Ecuación 1: Forma básica de una matriz de transformación*

Es una matriz de 4x4. Las tres primeras filas contienen las rotaciones $\alpha, \beta, \gamma$, que correponden con los ejes X, Y, Z respectivamente. En la última columna se define la traslación del punto, y la última fila contiene tres ceros y finalmente un uno.

Para construir las matrices de transformación de la cadena cinemática de InnerModel a partir de las posiciones de las articulaciones hemos tenido la suerte de contar con la ayuda de la Doctora Pilar Bachiller Burgos, a quien se lo agradecemos de manera explícita aquí. Nos ha diseñado un método matemático para realizar la operación y posteriormente lo hemos implementado con éxito.

Para tratar de explicarlo de manera sencilla omitimos parte del desarrollo matemático y lo haremos para construir la primera matriz de transformación de nuestro InnerModel, la del Torso.

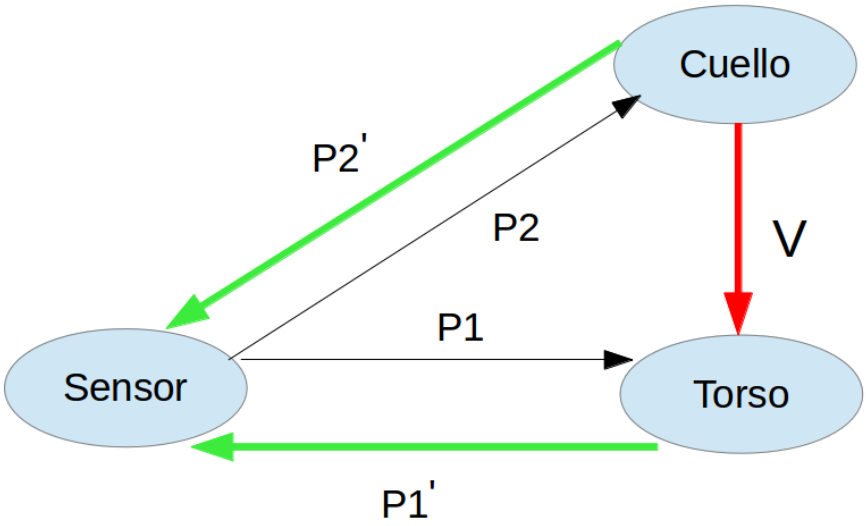


*Figura 49: Cálculo de las matrices de transformación*

- Si observamos el dibujo, solo tenemos los datos de los vectores de translación (en negro) desde el sensor y también conocemos la matriz de transformación de nuestro sensor respecto a su padre que es la identidad (a más profundidad en las articulaciones RT será la matriz de Transformación de toda la cadena. Es decir, para el hombro RT será igual a RTSensor*RTTorso). De este "cuelga" nuestro árbol cinemático que empieza en el Torso. Por tanto podríamos obtener los vectores (P1 y P2) vistos desde los nodos Torso y Cuello (P2', P1') en verde, genéricamente y en coordenadas homogéneas:

$$P_i' = RT^{-1} \cdot \begin{bmatrix} P_i \\ 1 \end{bmatrix}$$

*Ecuación 2: Cálculo de $P_i$*

En este caso ya que es el primer nodo de la cadena, conocemos directamente la traslación del Torso respecto al padre, el sensor, en este caso P1'. Pero genéricamente deberíamos multiplicar la traslación por la matriz inversa de la matriz RT.

En este punto tendríamos la translación T(Tx ,Ty, Tz) de un nodo respecto a su padre, es decir, la translación de nuestras futuras matrices de transformación.

- Para el cálculo de las rotaciones de una articulación es necesario obtener el vector $\vec{V}$ visto desde el padre. Para ello debemos trabajar en la línea definida anteriormente.

$$\vec{V} = P_2' - P_1'$$

*Ecuación 3: Cálculo del vector $\vec{V}$*

El vector se calcula restando los vectores que definen la posición del sensor desde ambas articulaciones ($P_{1'}$ y $P_{2'}$).

Incluimos a continuación, para el vector normalizado calculado, la obtención de sus ángulos, el desarrollo de las ecuaciones ha sido omitido para no abrumar al lector:

$$R_X = \arctan\left(\frac{-V_Y}{V_Z}\right) \qquad R_X = \arcsin\left(V_X\right) \qquad R_Z = 0$$

*Ecuación 4: Cálculo de las rotaciones*

Estas ecuaciones tienen en cuenta tanto el sentido de giro de los ejes como el eje principal de nuestra articulación. En nuestro caso siempre será el eje Z, y las rotaciones se producen siguiendo la regla de la mano izquierda, esto es, en el sentido de las agujas del reloj. En este punto tenemos ya nuestra matriz de transformación, formada por los valores calculados (Rx,Ry,Rz,Tx,Ty,Tz).

- La rotación del eje Z no se puede calcular para ninguna de las articulaciones, pero para el torso hemos podido calcularla al conocer la posición de los hombros. Por ejemplo, de la cabeza no podemos calcular su rotación, pues NITE2 no hace el seguimiento de las orejas. Veamos gráficamente cuál es la idea que seguiremos:

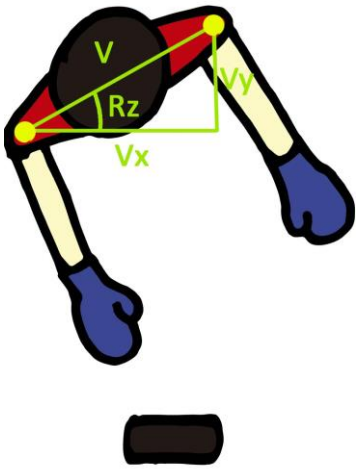


*Figura 50: Cálculo de la rotación del torso en el eje Z*

Las ecuaciones que dan valor a esta rotación son las siguientes:

$$V = V_{HOMBROIZQ} - V_{HOMBRODER} \qquad R_Z = arctg\left(\frac{V_Y}{V_X}\right)$$

*Ecuación 5: Cálculo de la rotación Z del torso*

Las operaciones a realizar son sencillas, se calcula el vector que une ambos hombros, y la rotación del eje Z es la arco tangente del resultado de la división de la posición Y del vector entre la posición X.

#### 5.3.3.3. GRABACIÓN/REPRODUCCIÓN DE VIDEOS RGBD

Además de las funcionalidades principales le hemos añadido a este componente la opción de grabar videos *.oni y reproducirlos. Se han insertado dos botones en la interfaz gráfica para este propósito.

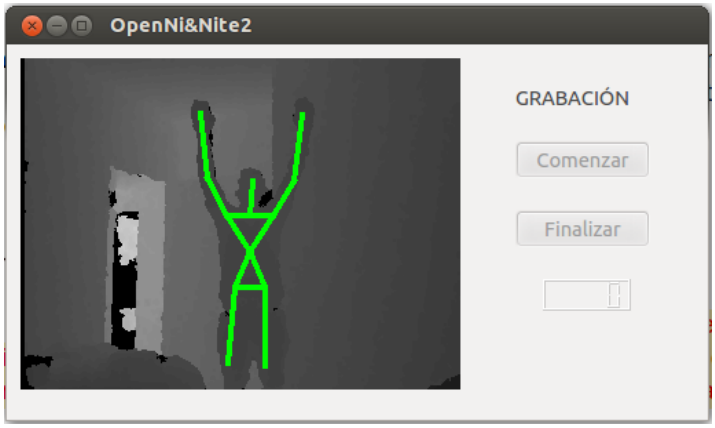


*Figura 51: Interfaz de OpenNI2Comp*

Cuando el usuario desee comenzar a grabar un video debe pulsar el botón 'Comenzar', el contador inferior comenzará a mostrar los segundos grabados, para terminar el video se debe pulsar el botón 'Finalizar'. Se creará un archivo de nombre 'Video.oni' en el directorio en el que se encuentra el ejecutable del componente.

Esta funcionalidad, que a priori, parecía iba a ser sencilla de implementar, nos ha traído un problema inesperado. La grabación del video se consiguió pronto, pues OpenNI tiene clases que hacen esta tarea sencilla, y se logra reproducirlo en nuestro componente y otras herramientas correctamente. El problema estuvo en el momento de hacer el seguimiento esqueletal del video. Fallaba al arrancar el componente, dando el siguiente error:

Couldn't getXN_STREAM_PROPERTY_ZERO_PLANE_DISTANCE

Esta variable es de la librería OpenNI2.

Probando con vídeos grabados con la herramienta NiViewer de OpenNI2, el seguimiento funcionaba correctamente. Por lo que el problema reside en nuestro desarrollo.

A día de hoy no hemos conseguido resolverlo, pues nos centramos en la implementación de las funcionalidades principales, considerando este error de poca importancia ya que existen herramientas para la grabación de videos *.oni. Lo que consideramos de extrema necesidad es la reproducción de videos *oni, y esto sí se ha conseguido.

## 5.4. POSEDETECTORCOMP

Este es el segundo componente del sistema implementado en este proyecto. Va a ser la imaginación del robot. Tendrá en su mente los datos obtenidos de la cámara RGBD como si de su percepción se tratase.

Los seres humanos cuando vemos una escena, almacenamos automáticamente en nuestra mente únicamente los datos más relevantes para nosotros, obviando muchos otros que no nos importan. De esta manera podemos reaccionar más rápidamente a cualquier situación que se nos plantee repentinamente. En este componente, cuando el robot esté frente a una escena va a almacenar en su mente únicamente a las personas, y obtendrá conclusiones acerca de sus intenciones analizando sus posturas.

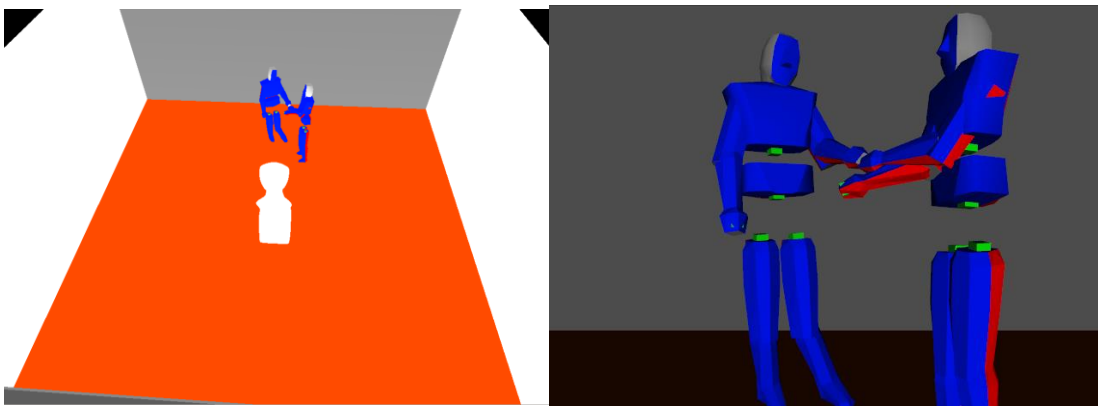

*Figura 52: Ejemplo de escena captada con una cámara RGBD y reproducida en el RCIS*

Dependiendo de los propósitos del robot, los datos relevantes para él serán unos u otros. Para este proyecto, cuyo objetivo es detectar las intenciones de humanas, sólo interesan las personas, lo demás se obvia. Este ejercicio de abstracción es esencial para llegar más rápidamente a la salida deseada.

Este componente se comunica con el componente OpenNI2Comp y con el simulador RoboComp InnerModel Simulator.

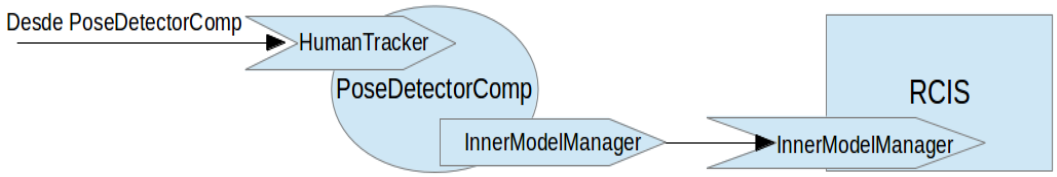


*Figura 53: Esquema de funcionamiento de PoseDetectorComp*

En el esquema podemos ver cómo este componente "consume" de OpenNI2Comp a través de la interfaz HumanTracker, y por otro lado, detecta si algún usuario está haciendo alguna postura conocida e inyecta la escena que el robot tiene en su imaginación en el simulador de RoboComp a través del proxy InnerModelManager.

### 5.4.1. DETECCIÓN DE POSTURAS

La detección de intenciones humanas es la función principal de este componente y el culmen de este proyecto. Para hacer esto vamos a definir un conjunto de posturas, ya que la pose corporal ofrece unas pistas claras sobre las intenciones de las personas.

Esto será una base para futuros proyectos, tanto de fin de grado como de máster, con propósitos similares. Aunque ofrece muchas pistas, es imposible hasta para los seres humanos aun teniendo una gran base de conocimiento detectar las intenciones de una persona simplemente con la pose corporal. Para detectarlas con una cierta fiabilidad es necesario tener varias fuentes de información como pueden ser el entorno, el tono de voz o los antecedentes de las personas. Además de no tener un conjunto cerrado de intenciones, o como por el momento esto es imposible, tener un conjunto muy amplio de ellas. Es decir tener un contexto de la escena ya que no es lo mismo, por ejemplo, agitar el brazo diciendo “adiós” que agitarlo diciendo “hola”, son gestos afines con significados opuestos, que son dependientes del contexto, incluso tan simplificado como en este ejemplo que es una única palabra de diferencia.

Aunque detectar poses con una cámara RGBD está siendo bastante estudiado recientemente [14] [15], la novedad de este proyecto reside en la forma de hacerlo ya que usaremos la idea de resta de árboles cinemáticos.

#### *5.4.1.1. DEFINICIÓN DE POSTURAS*

Para detectar las posturas se han definido once posturas humanas como árboles cinemáticos, cada una define una intención. Estas posturas son conocidas a priori, por lo que serán la base de conocimiento del robot que utilizará para detectar las intenciones de los usuarios que se encuentren en la escena

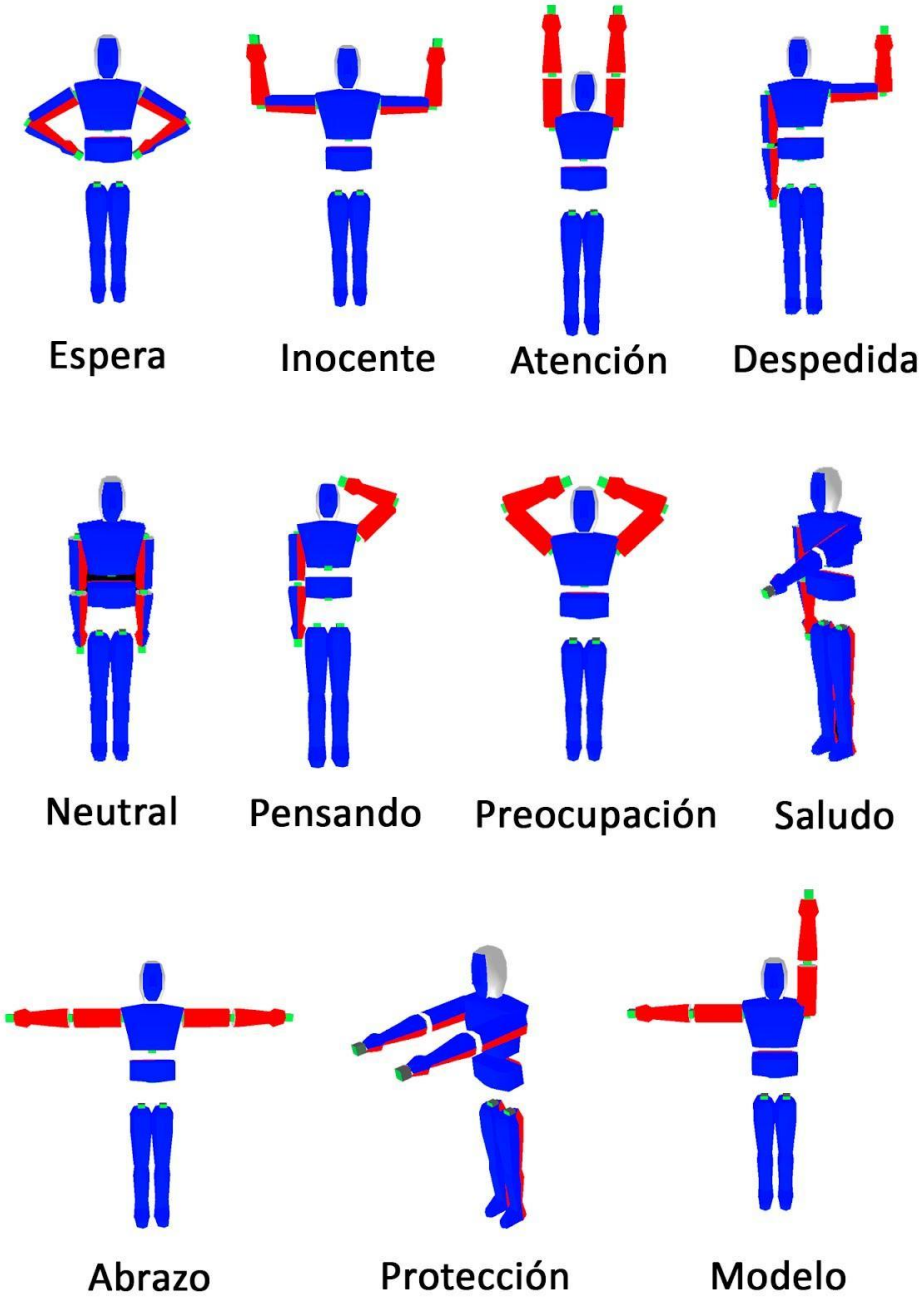


*Figura 54: Posturas definidas para su detección*

Hemos definido un conjunto de posturas lo suficientemente amplio y heterogéneo como para asegurar el correcto funcionamiento del proceso de detección, y las intenciones asociadas a cada postura son meramente orientativas. Para el uso de este componente en otros proyectos, es necesario estudiar su ámbito para definir un conjunto de posturas e intenciones acorde al mismo.

#### *5.4.1.2. COMPARACIÓN DE ÁRBOLES CINEMÁTICOS*

La comparación de árboles cinemáticos para la detección de intenciones es el punto más novedoso del proyecto, en cuanto a investigación se refiere.

La idea principal es tener un árbol cinemático por cada postura definida, como hemos explicado en el punto anterior. Y restarlos con el árbol cinemático del usuario para que si su resta está cerca de cero, averigüemos en qué pose se encuentra.

La forma de comparar los árboles no es trivial. Los datos que podemos obtener de ellos plantean problemas de comparación que son insalvables para poder hacerlo directamente:

- **Comparar posturas relativas**: La posición de las articulaciones en los árboles cinemáticos, como ya hemos dicho en otros puntos, son relativas. Esto es, toman como origen de coordenadas el nodo padre, y teniendo cada articulación en su eje de coordenadas, el eje Z alineado al hueso de la extremidad que forma, la posición relativa es siempre la misma se encuentre en la postura que se encuentre.

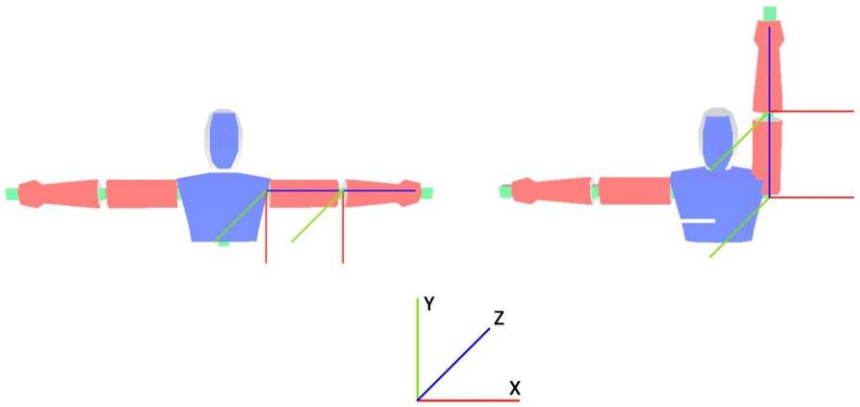


*Figura 55: Problema para comparar posiciones relativas*

En la imagen vemos que el brazo esté en vertical u horizontal, la posición relativa del codo por ejemplo, con respecto al hombro es siempre la misma, cero en los ejes X e Y y un valor determinado en el eje Z.

- **Comparar rotaciones relativas**: La orientación de las articulaciones en los árboles cinemáticos, como las posiciones, son relativas. Estos datos son los que realmente definen la postura, por lo que son las que habría que comparar. El problema principal reside en que con distintas rotaciones podemos llegar a la misma posición, por lo que puede darse el caso de que el usuario esté en la posición que buscamos, pero con rotaciones distintas a las definidas en el modelo.

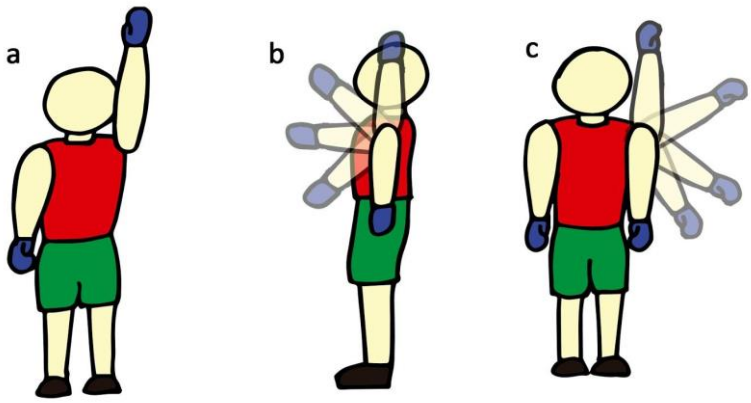


*Figura 56: Problema para comparar rotaciones relativas*

En la imagen vemos un claro ejemplo de este problema. Para llegar a la postura 56.a teniendo el brazo hacia abajo podemos hacerlo con dos movimientos distintos, definidos en las figuras 56.b y 56.c. Estos dos movimientos implican las rotaciones de dos ejes distintos.

Como hemos visto, los datos del árbol no se pueden comparar directamente, por lo que la mejor opción es hacer una transformación con ellos y comparar esta, las opciones son varias, unas válidas y otras no:

- **Comparar posiciones siendo el dispositivo el origen de coordenadas**: Existe la posibilidad de, con la función *transform()* de InnerModel, obtener la posición de un nodo desde el punto de vista de otro. Es decir, una posible solución podría ser obtener las posiciones de cada articulación desde el punto de vista de la cámara, tanto del modelo como del usuario, y compararlas. El gran problema de esta idea es que para que la comparación funcione el usuario debe estar colocado exactamente en la misma posición que el modelo. Es decir, si el modelo se encuentra a dos metros de la cámara, y el usuario a tres, la comparación no funcionaría. Dicho de otra manera, dos usuarios colocados uno al lado del otro en la misma postura estarían en posiciones distintas, y su comparación sería errónea.

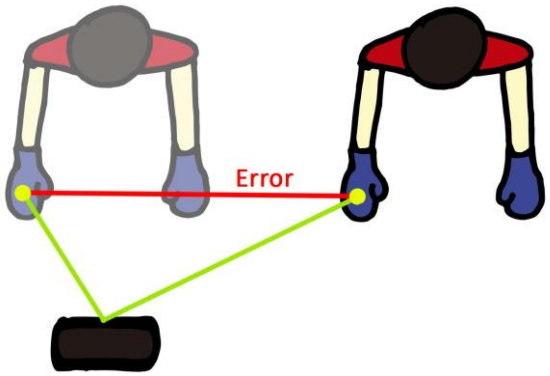


*Figura 57: Error de posición frente a la cámara RGBD*

La solución elegida es **comparar posiciones siendo el torso el origen de coordenadas**. Haciendo esta transformación se resuelve el problema anterior. De esta manera, dados el modelo de postura y el árbol de usuario, haciendo la transformación, por ejemplo, de la posición de la mano vista desde el torso de uno y otro árbol, si la postura es la misma, la posición también. Procediendo de la misma manera con cada una de las articulaciones, podemos decidir si las posturas definidas en ambos árboles son semejantes. Como ilustra la figura:

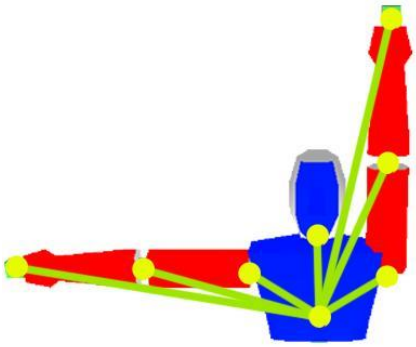

*Figura 58: Posiciones relativas desde el torso*

Este método tiene algunos problemas que comentaremos a continuación, y explicaremos las soluciones que se han planteado para tratar de resolverlas y cuál de ellas ha sido la elegida y por qué.

#### 5.4.1.2.1. DIFERENCIA DE TAMAÑO USUARIO-MODELO

Como ya dijimos anteriormente, tenemos un modelo sintético con la postura ideal que queremos detectar, ese modelo tiene un tamaño de extremidades determinado según las tablas de Contini [16], pero el usuario que esté frente a la cámara no tiene por qué tener ese tamaño, puede ser más grande o más pequeño, derivando en errores al comparar árboles que efectivamente se encuentran en la misma postura. Con el objetivo de explicarlo mejor observemos la siguiente imagen:

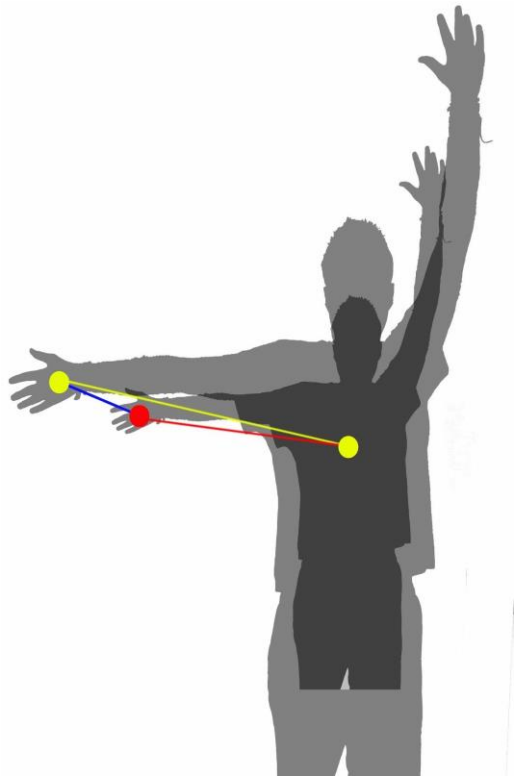

*Figura 59: Error de comparación provocado por la diferencia de tamaño*

Pensemos que el modelo sintético viene dado por la silueta más grande, y el usuario por la silueta pequeña. El vector amarillo determina la distancia de la mano derecha del modelo a su torso, y el vector rojo la distancia de la mano derecha del usuario a su torso. Como podemos comprobar están exactamente en la misma posición, pero son de distintos tamaños, esto hace que haya un error, expresado en el vector azul. El error viene cuantificado por la ecuación:

$$E = norma(\overrightarrow{TJ}_M - \overrightarrow{TJ}_U) = norma(\overrightarrow{J_M J_U}) = euclideanDistance(transform(Torso_M, Joint_M), transform(Torso_U, Joint_U))$$

*Ecuación 6: Error entre dos posiciones homónimas de distintos árboles cinemáticos*

Para solucionar este problema decidimos escalar el tamaño del usuario al tamaño del modelo Para escalarlo necesitamos saber cuánto más grande o más pequeño es el usuario que el modelo, es decir la relación entre ambos. Y

posteriormente cambiar el tamaño dado ese porcentaje. ¿Pero cómo obtenemos dicha relación?

A continuación se presenta dos caminos para hacerlo:

**ESCALAR CON ESTATURAS CORPORALES**

La primera opción y quizás más intuitiva sería pensar escalarlo comparando las estaturas de ambos cuerpos. Para obtener la estatura podríamos—obtener la transformación desde un pie a la cabeza, extraer el vector de translación, y quedarnos con el eje Z del vector que obtenemos. Pero esto tiene algunos inconvenientes:

- La persona necesita estar de pie. En ciertas posturas, como puede ser, sentada esta solución no es factible.
- Sólo funcionaría en el caso de que el usuario se muestre completo frente a la cámara, ya que OpenNI2 calcula unos valores imprecisos de las articulaciones que no ve, por lo que la estatura calculada muy posiblemente sería errónea.

Estos inconvenientes tienen el suficiente peso como para descartar esta opción.

**ESCALAR CON ARTICULACIONES FIJAS**

Esta opción soluciona los problemas anteriores. Cuando hablamos de articulaciones fijas hablamos de articulaciones que prácticamente no varían respecto a su articulación padre. Articulaciones de este tipo serían los hombros o el cuello respecto al torso. Podemos verlo mejor en la siguiente imagen:

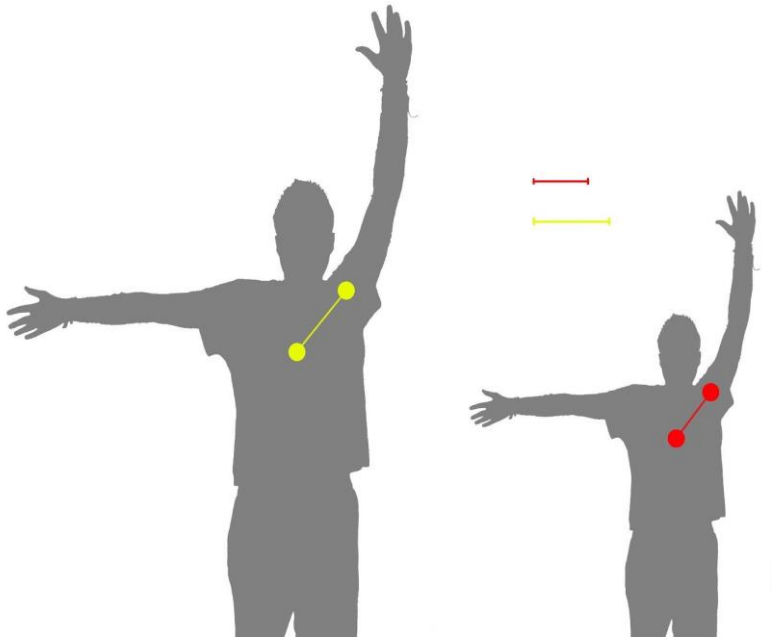

*Figura 60: Ejemplo de articulaciones fijas*

Para hacer esto realizamos una transformación relativa entre dicha articulación (trabajaremos con un hombro) con respecto al torso, tanto del modelo como del usuario, obteniendo dos posiciones $TH_M$ y $TH_U$ respectivamente. Este vector se obtiene fácilmente gracias a la primitiva de InnerModel, *transform()*, que devuelve el vector relativo entre dos nodos del árbol cualesquiera, pudiendo incluso para el nodo destino, expresar un vector relativo a este. Una vez calculadas estas posiciones, calculamos la distancia de ambas desde el origen (el torso) a cada una de ellas, y dividimos una distancia entre la otra. Esta división es el coeficiente que nos dice cuánto más grande (si es mayor que 1) o más pequeño (si es menor que 1) es un cuerpo de otro.

La fórmula con la que obtenemos el coeficiente es la siguiente:

$$C = \frac{norma(\overrightarrow{TS_M})}{norma(\overrightarrow{TS_U})} = \frac{euclideanDistance((0,0,0), transform(Torso_M, Shoulder_M))}{euclideanDistance((0,0,0), transform(Torso_U, Shoulder_U))}$$

*Ecuación 7: Cálculo del coeficiente de escalado*

Para aplicar el cambio de tamaño, basta con multiplicar cada posición relativa de cada articulación desde el torso del usuario por dicho coeficiente. Esta solución es la primera implementada pero se han detectado algunos errores.

- No todas las articulaciones tiene por qué ser igual de grandes o pequeñas que las del modelo. Es decir, puede darse que los brazos del modelo sean un 10% más grandes que los del usuario y que la distancia del cuello al torso sea un 5% menor.
- Este coeficiente se aplica a los tres ejes (X, Y, Z) de la posición relativa de cada articulación respecto al torso y esto es un error, puesto que la posición se mueve en la dirección que se genera al unir el torso con la articulación en cuestión, y no en la dirección que debería ser, que es la que une dicha articulación con el nodo padre (mano con codo, codo con hombro…). Para ver esto con mayor claridad fijémonos en la siguiente imagen.

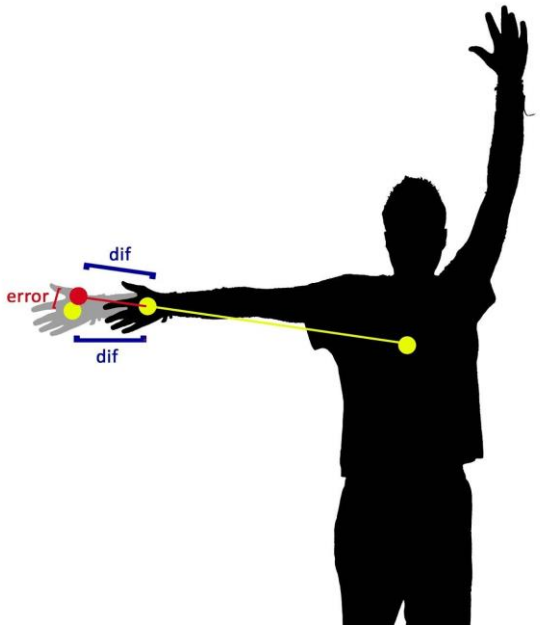


*Figura 61: Problema de escalado por posiciones fijas*

En el dibujo vemos la situación en la que el brazo del usuario es más pequeño que el del modelo y debemos escalarlo tanto como marca la distancia etiquetada por

*dif*. Si le aplicamos el coeficiente a la posición relativa de la mano desde el torso, no crece en el sentido en el que debería. Aun mejorando el error en muchos casos, genera otro que debemos solucionar.

El siguiente paso parece claro, calcular el coeficiente de crecimiento por cada una de las articulaciones, y mover la posición de la articulación en el ángulo que marca el padre. Para ello habría que cambiar la posición relativa de cada articulación con el padre, la propia que almacena el árbol InnerModel, por lo que habría que modificarlo (o crear una copia y modificar ésta para no perder los valores originales).

Implementando esta solución y comprobando las posiciones que deberían ser correctas, se llegó a la conclusión de que la mejor solución y más correcta es mucho más sencilla y no necesita añadir tanta matemática.

**FUSIÓN DE ÁRBOLES INNERMODEL**

La solución definitiva elegida no necesita escalados ni cálculos, es directa y sencilla de implementar. En las soluciones anteriores nos centrábamos en calcular cuánto más grande o más pequeño era el modelo sintético con respecto al usuario para cambiar el tamaño del árbol InnerModel del usuario y poder comparar sus posiciones.

Pues bien, este planteamiento no era el más acertado. No necesitamos saber cuánto más grande es uno que otro para poder cambiar el tamaño, pues sabemos el tamaño final que debe tener el árbol que comparemos con el modelo. Es decir, lo que realmente necesitamos es un árbol del tamaño del modelo pero con la postura del usuario. Se conocen los datos necesarios, por lo tanto la solución reside en crear un nuevo árbol InnerModel con las traslaciones del modelo y las rotaciones del usuario.

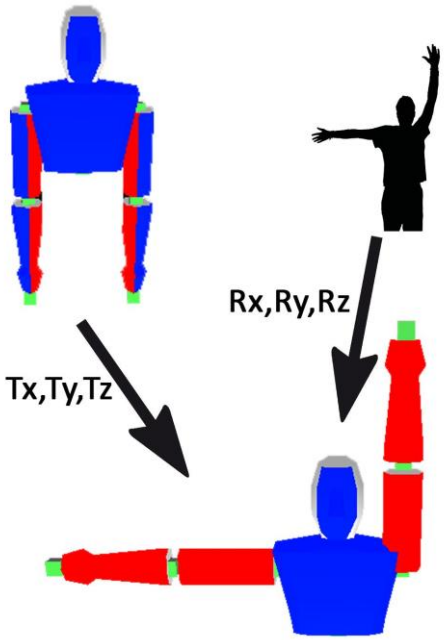


*Figura 62: Fusión de árboles InnerModel*

Con esto tendríamos un árbol escalado de un tamaño exactamente igual al del modelo con la postura del usuario, ya que simplemente con las rotaciones de las articulaciones se define una postura. Éste árbol escalado será el que se compare con el modelo.

Es la solución óptima, pues el tamaño del usuario es totalmente transparente en la comparación, ya que en ningún momento lo utilizamos. Con esto conseguimos también, que cualquier usuario que se presente frente a la cámara RGBD se mueva en la comparación en el mismo rango del espacio, haciendo válido el umbral que se determinará en los experimentos para cualquier tamaño de usuario. Esto no sería posible de otra forma, ya que la misma diferencia de posición en una articulación de un usuario de 110 cm, afecta más a la postura que la misma diferencia en un usuario de 190 cm, por lo que el umbral no debería ser el mismo, y deberían ser calculados en proporción, lo que haría el proceso bastante más complejo e inexacto.

Además la construcción del árbol de comparación es un proceso rápido y eficiente, exento de cálculos matemáticos, y que define perfectamente un árbol con la postura del usuario y tamaño exactamente igual al del modelo sintético, ideal para su comparación.

#### 5.4.1.2.2. DIFERENCIA DE ROTACIÓN USUARIO-MODELO

Además del error del tamaño se han detectado dos más, que tienen que ver con que el usuario no esté justamente frente a la cámara:

- Puede ocurrir que la cámara no esté a la altura del usuario, y que para encuadrarlo completamente tenga que estar girada hacia arriba o abajo. Esto provoca que la percepción de la cámara hacia el usuario sea que éste se encuentre ligeramente "tumbado", ya que la cámara se presupone en el eje de coordenadas y sin ningún tipo de rotación. Para entender este problema se representa gráficamente en la siguiente imagen:

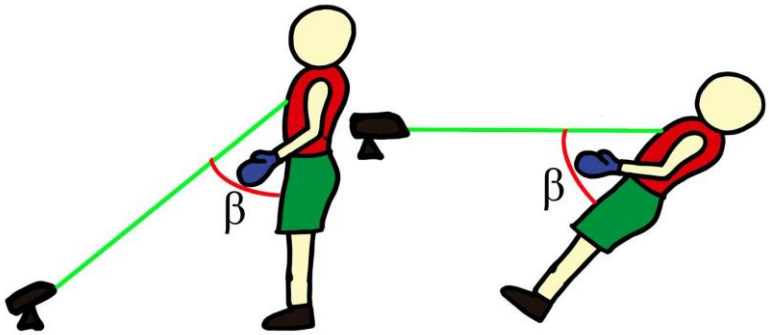


*Figura 63: Error provocado por la altura de la cámara RGBD*

- El segundo caso que tratamos en este apartado consiste en que el usuario no esté frente a la cámara, sino que se encuentre girado y la cámara lo vea desde una vista lateral. En este caso, si no se hace nada al respecto, la detección de posturas no se realiza correctamente, ya que el modelo con el que

comparamos se encuentra justamente de frente, y las posiciones de las articulaciones no coincidirían. Este problema lo podemos representar así:

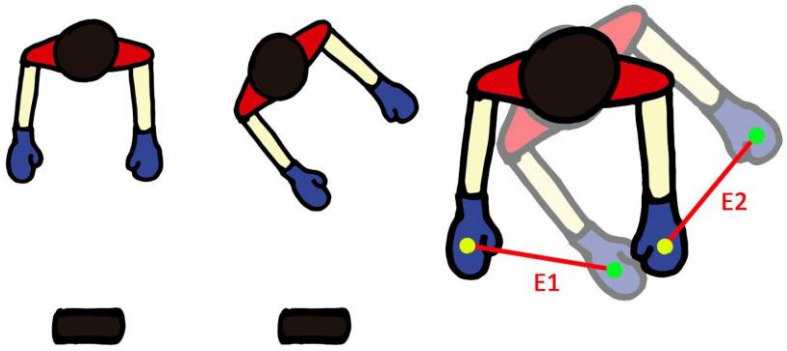


*Figura 64: Error provocado por la rotación del usuario frente a la cámara*

En la imagen podemos ver cómo el usuario tiene la misma postura en las dos primeras figuras, pero en la segunda está girada frente a la cámara. Este hecho induce a un error representado en la tercera figura, en la que la mano derecha tiene un error E1 y la mano izquierda un error E2, ocurriendo lo mismo en todas las articulaciones. Tales errores no deberían existir ya que la postura no varía.

La solución de ambos problemas es la misma y análoga a la del escalado. Consiste en sustituir la rotación del torso que tiene el árbol de comparación, que es la del usuario, por la del modelo. Es decir, para compararlo interesa colocar el árbol que vamos a comparar con respecto a la cámara igual al modelo con el que comparamos. Esto se encuentra en la rotación del torso del modelo, y podemos obtenerla directamente.

#### 5.4.1.2.3. MÉTRICAS

Una vez tenemos ambos árboles cinemáticos normalizados para su comparación, hay que decidir la métrica con la que comparar ambos árboles, qué distancias tienen sentido en éste ámbito y descartar las que no, y dentro de las posibles comprobar cuál ofrece mejor resultado. Para el estudio de la comparación de árboles cinemáticos hemos trabajado con las siguientes distancias, obteniendo los resultados que pueden verse en el punto 6.

- Distancia de Manhattan [17]: También denominada métrica taxicab, distancia rectilínea o norma 1. En esta geometría se define la distancia entre dos puntos como la suma de las diferencias absolutas de sus coordenadas. La fórmula que la define es la siguiente

$$d_1(\mathbf{p},\mathbf{q}) = \|\mathbf{p}-\mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

donde p y q son vectores de n dimensiones.

*Ecuación 8: Distancia Manhattan*

En nuestro caso, de 3 dimensiones, la Distancia Manhattan entre un punto $P=(p_x,p_y,p_z)$ y otro $Q=(q_x,q_y,q_z)$ es $d(P,Q) = |p_x-q_x| + |p_y-q_y| + |p_z-q_z|$.

Gráficamente, la Distancia Manhattan se describe en la siguiente imagen.

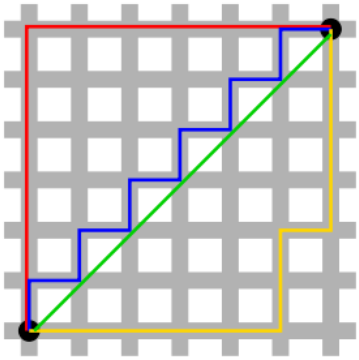

*Figura 65: Distancia Manhattan frente a Distancia Euclídea*

En la imagen tanto las líneas azul, amarilla y roja definen la Distancia Manhattan, cuyo valor es 12. Por otro lado la línea verde identifica a la distancia euclídea, que es la más común y que definiremos en el siguiente punto, cuyo valor es $6\times\sqrt{2} \approx 8.48$.

- Distancia Euclídea [18]: Es la distancia "ordinaria" entre dos puntos de un espacio euclídeo, la cual se deduce a partir del Teorema de Pitágoras [19]. La fórmula que la define es:

$$d_E(P,Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

*Ecuación 9: Distancia euclídea*

En nuestro caso, de 3 dimensiones, la Distancia Euclídea entre un punto $P=(p_x,p_y,p_z)$ y otro $Q=(q_x,q_y,q_z)$ es $d(P,Q) = \sqrt{[(p_x-q_x)^2 + (p_y-q_y)^2 + (p_z-q_z)^2]}$.

#### 5.4.1.2.4. IMPLEMENTACIÓN

En cuanto a la implementación de la comparación de árboles cinemáticos tenemos que tener en cuenta que no se hace de todas las articulaciones del cuerpo, sino únicamente de la parte superior del cuerpo, por dos motivos principales:

- El seguimiento de las piernas de NITE2 no funciona aún de una manera fiable, necesita algunos requisitos que no siempre se dan, como decíamos en el punto en el que hablábamos de esta librería.
- RoboLab no posee ningún robot con piernas.

Para hacer esto tenemos una lista de cadenas que definen las articulaciones que se quieren comparar, en el caso futuro que se quieran comparar más articulaciones sería una tarea muy sencilla, basta con añadir elementos a esta lista.

Además de la lista con las articulaciones a comparar, tenemos un diccionario para definir el nombre que le queremos dar a las articulaciones en el árbol cinemático.

La implementación de la comparación se ha hecho recorriendo la lista de articulaciones a comparar y utilizando una función *getNode(QString id)* de InnerModel que dándole la cadena que define su identificador devuelve un nodo completo. De esta manera obtenemos los dos nodos semejantes de ambos árboles y hacemos la transformación que comentábamos antes. A estas dos posiciones les aplicamos alguna de las métricas del punto anterior, y si la distancia obtenida está por debajo de un umbral óptimo obtenido en las pruebas, se considera que ambas articulaciones están en la misma posición, si no, se descarta esa postura.

Este proceso se realiza sobre cada una de las poses definidas. En el caso que dos poses puedan darse por buenas, se elige la que menor diferencia de distancias ha dado al aplicarle la métrica, teniendo en cuenta todas las articulaciones comparadas. Es decir, la más parecida a la pose del usuario.

La comparación se ha hecho recorriendo la lista, a pesar de que podría haberse implementado mediante un recorrido de árboles, quedando una solución más elegante. Pero teniendo en cuenta el número de nodos del árbol (la mayoría de ellos no se comparan) el recorrido de la lista es más eficiente que el de árboles, y considerando que este proceso se hace varias veces por frame, era necesario que fuera lo más rápido posible. Los recorridos de árboles los hemos implementado en otros puntos del sistema menos críticos, como veremos más adelante.

### 5.4.2. VISUALIZACIÓN DE LOS USUARIOS EN RCIS

Este componente además de detectar la postura en la que se encuentran los usuarios de la escena, inyecta, modifica y borra los modelos en el simulador de RoboComp, RC InnerModel Simulator, para que podamos comprobar gráficamente

qué información tiene el robot en su mente. Vamos a comentar cómo se ha implementado cada una de estas operaciones.

#### *5.4.2.1. INSERCIÓN DE USUARIO*

El componente, en su inicialización carga un InnerModel con el modelo de una persona. El modelo podemos encontrarlo en robocomp/Files/InnerModel/person.xml.

Por otro lado tenemos una lista interna de usuarios que se actualiza cada frame dependiendo de los datos ofrecidos por el componente OpenNI2Comp a través de la interfaz HumanTracker.

En el momento en el que se detecta una persona en la lista obtenida con HumanTracker que no está en la lista interna se inserta en ella y se inyecta al simulador. Para ello se hace un recorrido en anchura del árbol cinemático del modelo de la persona cargado al arrancar el componente insertando cada nodo leído en el simulador con las funciones que ofrece InnerModelManager. Es necesario hacer un casting de los nodos previo a la inserción para comprobar el tipo de nodo (joint, transform, mesh…) e insertarlo con el tipo y datos correctos.

Para hacer el recorrido en anchura se ha usado una cola auxiliar, y hemos tenido que definir una estructura para conocer el padre de cada nodo, ya que es necesario para la inserción en el simulador al nivel correcto, veamos el pseudocódigo para explicarlo:

```
estructura PadreHijo{
      cadena padre;
      cadena hijo;
};

funcion insertarNodo{
      PadreHijo ph;
```

```
      Cola<Padrehijo> cola;
      InnerModelNode nodo;

     ph.padre = "world"
      ph.hijo = "torso"
      cola.encolar(ph);

      mientras(!cola.vacia){
            ph = cola.desencolar;
            nodo = obtenerNodo(ph.hijo);
            insertarNodo(nodo, ph.padre);

            para cada (nodo.obtenerHijos)
                  encolar hijo;
      }
}
```

El proceso consiste en encolar el primer nodo de una persona (el torso). Posteriormente se entra en un bucle que mientras la cola no esté vacía, o dicho de otro modo, mientras haya más nodos que insertar, se desencola uno, se inserta en el simulador, y se encolan los hijos de dicho nodo. De esta forma se va "creando" la persona dentro del simulador por niveles.

Hemos elegido un recorrido en anchura en lugar de un recorrido en profundidad por motivos de eficiencia. Con esta implementación tenemos un proceso iterativo, que es más rápido que un proceso recursivo, necesario si nos hubiésemos decantado por el recorrido en profundidad.

A la hora de insertar un nodo se inserta con el nombre definido en el diccionario nombrado anteriormente seguido del identificador de la persona. De esta manera conseguimos la inserción de todas las personas que haya en la escena. Por ejemplo, si en el diccionario definimos para la articulación "JOINT_TORSO" el nombre "TORSO", esta articulación para el usuario de identificador 1 se llamará "TORSO_1".

#### 5.4.2.2. BORRADO DE USUARIO

El borrado de usuarios se realiza en el momento en el que llega la lista de usuarios de HumanTracker, se va recorriendo la lista interna, y en el caso de que no esté en la lista que ha llegado de la interfaz, se borra, y se borra el nodo correspondiente.

El InnerModel del simulador puede tener en un momento dado varios subárboles colgados del nodo "world" que representen a distintas personas. Para borrar a una persona con identificador X porque ya no está en la escena, basta con borrar el nodo principal, es decir, el torso, ya que así se borran todos los descendientes. Esto lo haremos creando el nombre del torso de dicha persona, que será, siguiendo con el ejemplo del punto anterior, "TORSO_X", y posteriormente lanzando la función *removeNode()* de InnerModelManager, con el nombre creado. De esta manera se borra el subárbol que define a esa persona, dejando intactos los demás subárboles.

#### 5.4.2.3. ACTUALIZACIÓN DE POSICIONES

El modo de actualizar las posiciones es similar al de la comparación de árboles. En este caso, recorremos el diccionario mencionado en otros puntos, el cual no es más que un mapa de todas las articulaciones con sus nombres para InnerModel, y actualizamos el nodo de identificador que nos da el diccionario con los datos ofrecidos por OpenNI2Comp a través de la interfaz HumanTracker, con la función de InnerModelManager updateTransformValues().

Como en el borrado y la inserción, para la actualización es necesario crear el nombre del nodo concatenando el nombre que ofrece el diccionario con el identificador del usuario, para modificar el subárbol correcto.

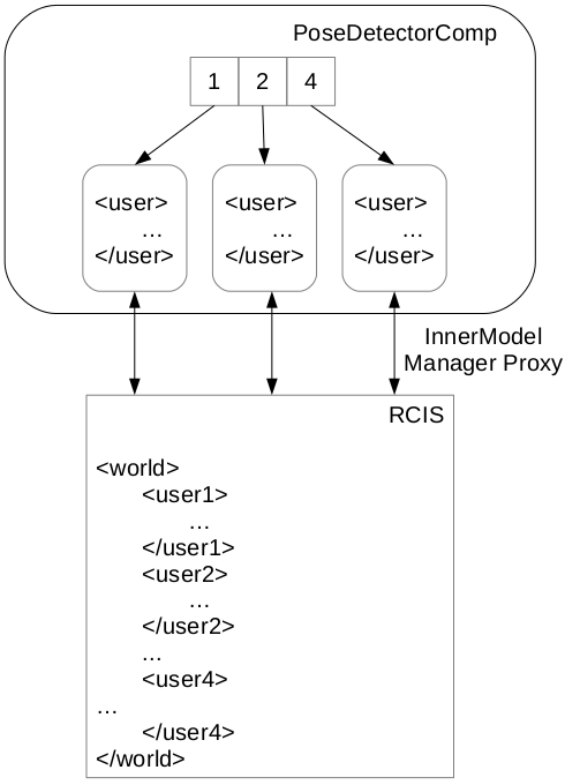


*Figura 66: Esquema de comunicación de PoseDetectorComp con RCIS*

Básicamente manejamos una estructura dinámica que puede contener hasta 16 personas, (limitación implícita como vimos por NITE/OPENI2), con sus respectivos árboles cinemáticos, esta estructura se actualiza en cada periodo con la información de los sensores y a su vez propaga los cambios al RCIS.

# 6. EXPERIMENTOS

En este punto pondremos a prueba el sistema que hemos implementado, para ver si el trabajo ha dado sus frutos o por el contrario habría que cambiar de táctica en futuros proyectos. Se definirán una serie de experimentos que darán unos resultados en ficheros *.CSV. En los componentes se han implementado unas funciones que exportan los datos que nos interesan a ficheros de este formato, para posteriormente poder tratarlos en hojas de cálculo y analizar los datos obteniendo una serie de conclusiones que pongan en relieve la mucha o poca fiabilidad del proyecto, los puntos fuertes y débiles, y los posibles caminos de mejora o ampliación.

## 6.1. DETECCIÓN DE POSES

El primer experimento consistirá en la grabación de diferentes personas realizando las posturas definidas. Sobre ellos se compararán los modelos analizando las diferencias entre articulaciones iguales de ambos árboles cinemáticos. A partir de estas diferencias calcularemos el mejor umbral para cada métrica. Y extraeremos las conclusiones oportunas que podamos deducir de este estudio.

Se han definido once posturas, y las pruebas se han hecho sobre ocho personas con dos métricas diferentes. El grupo de usuarios elegido hemos intentado que fuera lo más heterogéneo posible, para poner a prueba la comparación. Consta tanto de hombres como de mujeres, que van desde los 8 años hasta los 28 y con estaturas de entre 110 cm y 192 cm.

Las métricas que utilizaremos son las definidas en puntos anteriores: la distancia Euclídea y la distancia Manhattan.

A continuación mostramos las gráficas con las diferencias resultantes al comparar los árboles de los usuarios con las posturas definidas. Hay una gráfica por cada

postura y métrica. En ellas podemos ver los resultados de comparar cada articulación de cada persona con el modelo sintético. Las diferencias están en unidades milimétricas.

**Distancia Euclídea**

Pose Despedida

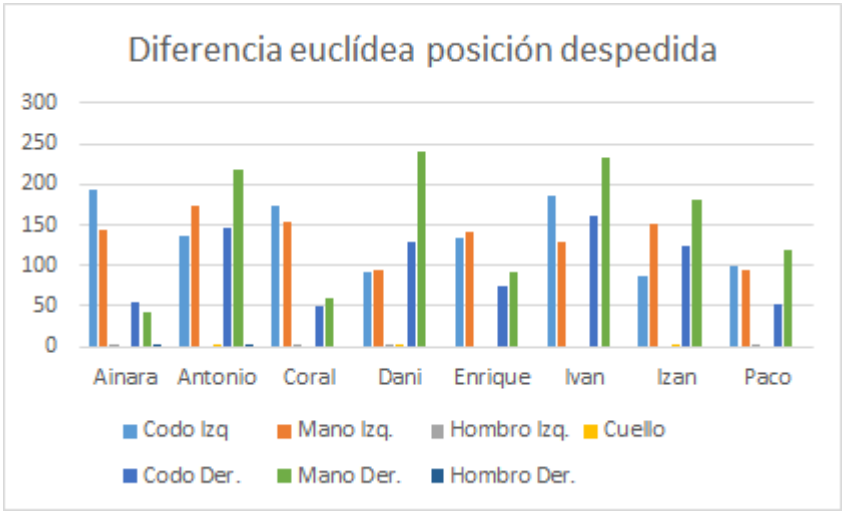


*Figura 67: Diferencia euclídea de la posición "Despedida"*

- Diferencia máxima de una articulación: Mano Derecha Dani, 240.669 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 148.424788 mm.
- Diferencia máxima total: Iván, 708.552202 mm.
- Diferencia media total: 520.49711 mm.

En esta gráfica están definidas las diferencias que se han producido al comparar el árbol de la pose "despedida" del modelo sintético con cada uno de los árboles de los usuarios haciendo la misma postura. Para entenderlo mejor fijémonos en el primer usuario, Ainara:

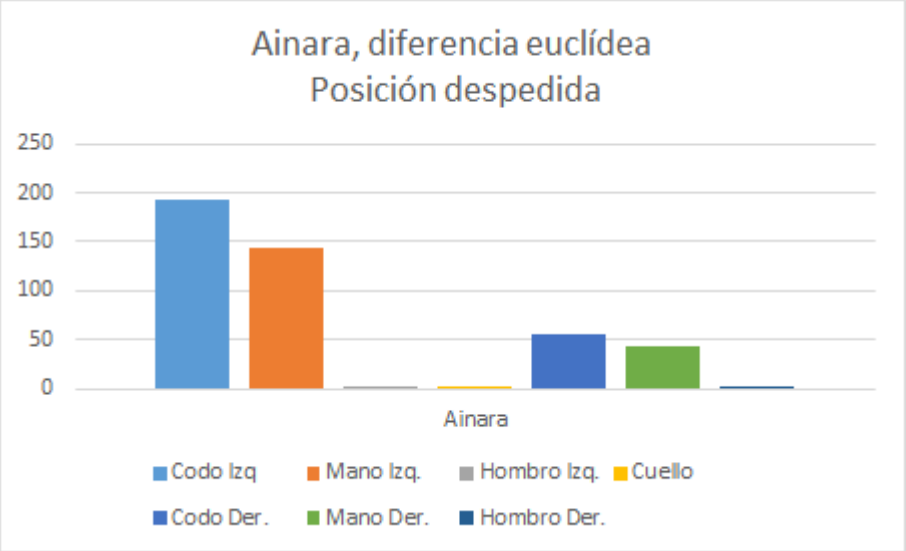


*Figura 68: Diferencia euclídea de la posición "Despedida" de Ainara*

Esta gráfica corresponde al primer bloque de barras de la gráfica anterior. En ella se muestran las diferencias obtenidas al comparar cada articulación de Ainara con el modelo sintético. Así, podemos ver que la posición con respecto al torso del codo de Ainara comparado con el modelo sintético se encuentra aplicando la distancia euclídea a algo menos de 20 cm, la mano izquierda a unos 15 cm, y el codo y mano derecha a unos 5 cm. Las demás articulaciones comparadas están prácticamente en el mismo sitio, pues la diferencia es casi 0 cm. Con esta gráfica podemos concluir que se encuentra en la posición despedida, y que el brazo peor colocado es el izquierdo, pues las mayores diferencias se encuentran en las articulaciones que lo forman.

La gráfica [Figura 67] representa lo explicado con Ainara, para cada uno de los usuarios. Así, las siguientes gráficas serán de la misma forma con cada una de las posturas, y posteriormente se hace el mismo proceso con la métrica de Manhattan.

Pose Neutral

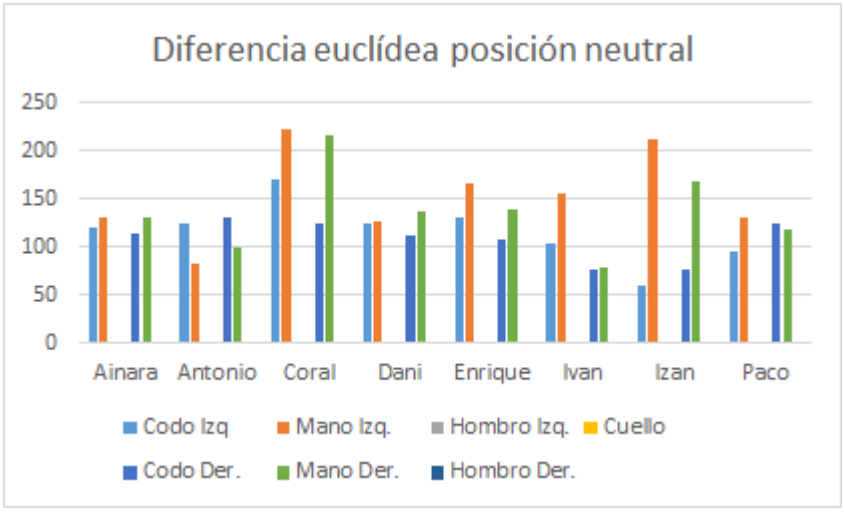


*Figura 69: Diferencia euclídea de la posición "Neutral"*

- Diferencia máxima de una articulación: Mano Izquierda Coral, 223.379 mm.
- Diferencia máxima media de las articulaciones: Mano Izquierda, 153.353 mm.
- Diferencia máxima total: Coral, 735.373244 mm.
- Diferencia media total: 514.582863 mm.

Pose Saludo

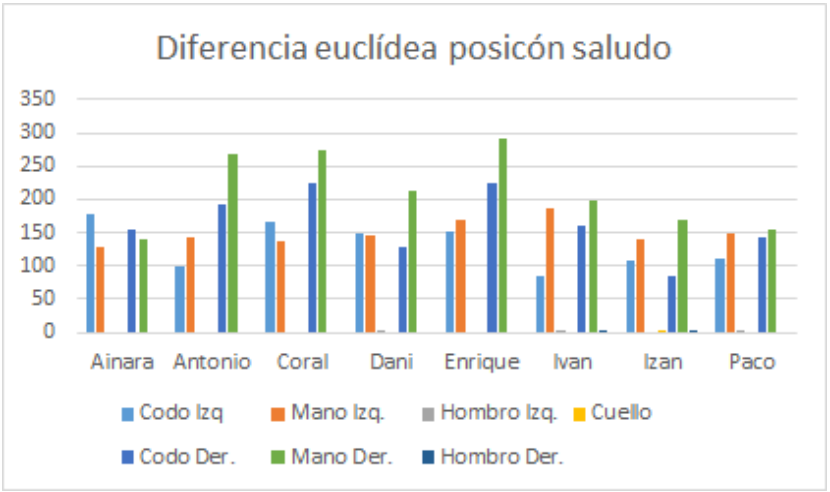


*Figura 70: Diferencia euclídea de la posición "Saludo"*

- Diferencia máxima de una articulación: Mano Derecha Enrique, 291.65 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 213.623625 mm.
- Diferencia máxima total: Enrique, 837.483381 mm.
- Diferencia media total: 658.223436 mm.

Pose Modelo

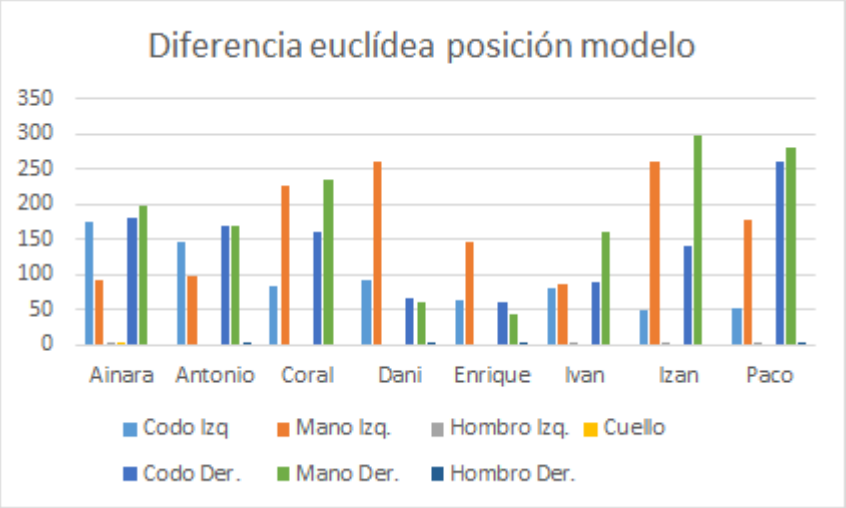


*Figura 71: Diferencia euclídea de la posición "Modelo"*

- Diferencia máxima de una articulación: Mano Derecha Izan, 298.103 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 180.829925 mm.
- Diferencia máxima total: Paco, 771.213069 mm.
- Diferencia media total: 583.71759 mm.

Pose Abrazo

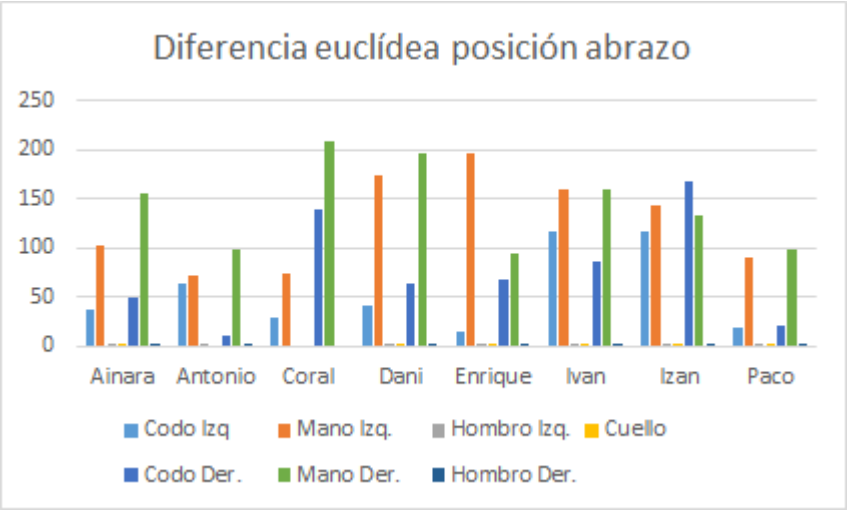


*Figura 72: Diferencia euclídea de la posición "Abrazo"*

- Diferencia máxima de una articulación: Mano Derecha Coral, 208.161 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 143.249025 mm.
- Diferencia máxima total: Izan, 560.304377 mm.
- Diferencia media total: 400.376482 mm.

Pose Protección

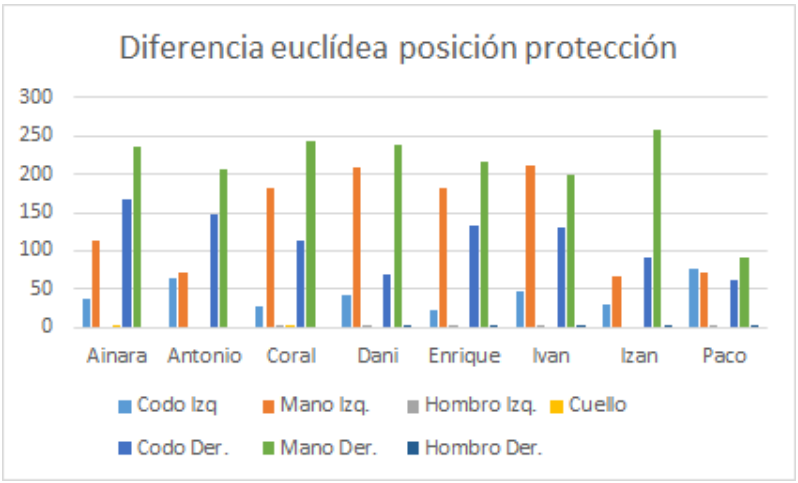


*Figura 73: Diferencia euclídea de la posición "Protección"*

- Diferencia máxima de una articulación: Mano Derecha Izan, 257.679 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 210.745338 mm.
- Diferencia máxima total: Iván, 588.589439 mm.
- Diferencia media total: 506.666687 mm.

Pose Inocente

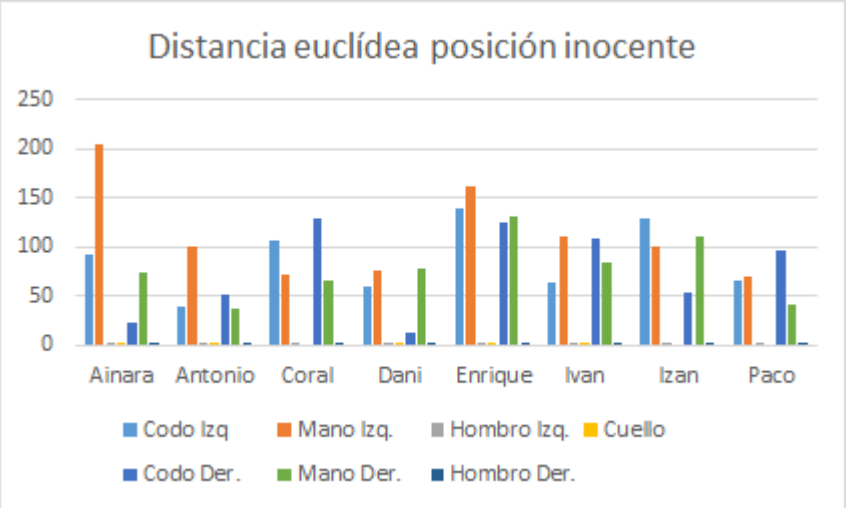


*Figura 74: Diferencia euclídea de la posición "Inocente"*

- Diferencia máxima de una articulación: Mano Izquierda Ainara, 204.149 mm.
- Diferencia máxima media de las articulaciones: Mano Izquierda, 111.819113 mm.
- Diferencia máxima total: Enrique, 557.091523 mm.
- Diferencia media total: 352.36281 mm.

Pose Preocupación

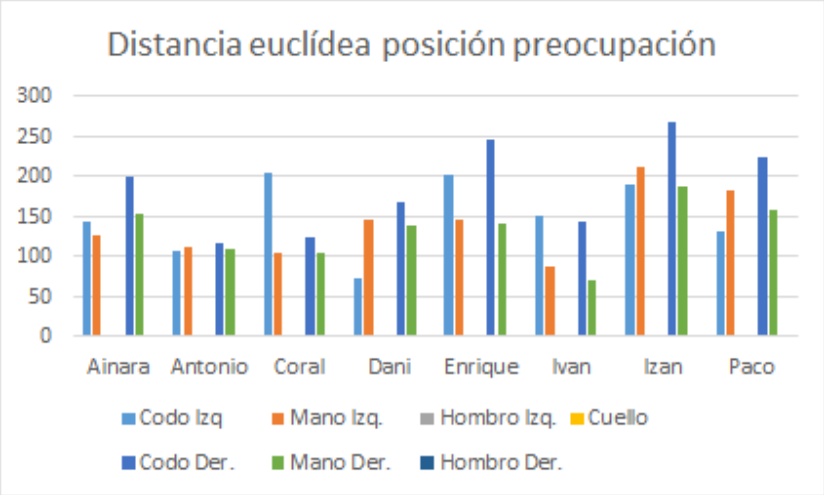


*Figura 75: Diferencia euclídea de la posición "Preocupación"*

- Diferencia máxima de una articulación: Codo Derecho Izan, 268.799 mm.
- Diferencia máxima media de las articulaciones: Codo Derecho. 187.125875 mm.
- Diferencia máxima total: Izan, 860.103403 mm.
- Diferencia media total: 610.792092 mm.

Pose Espera

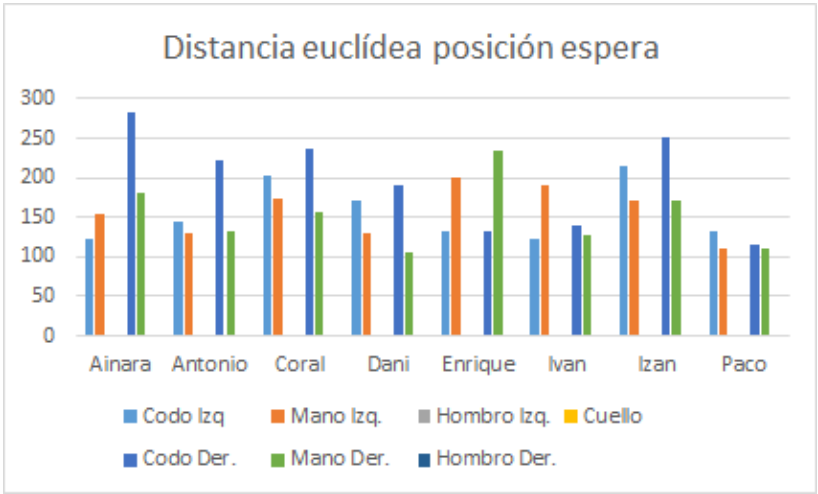


*Figura 76: Diferencia euclídea de la posición "Espera"*

- Diferencia máxima de una articulación: Codo Derecho Ainara, 282.883 mm.
- Diferencia máxima media de las articulaciones: Codo Derecho, 196.665125 mm.
- Diferencia máxima total: Izan, 808.948358 mm.
- Diferencia media total: 662.575135 mm.

Pose Atención

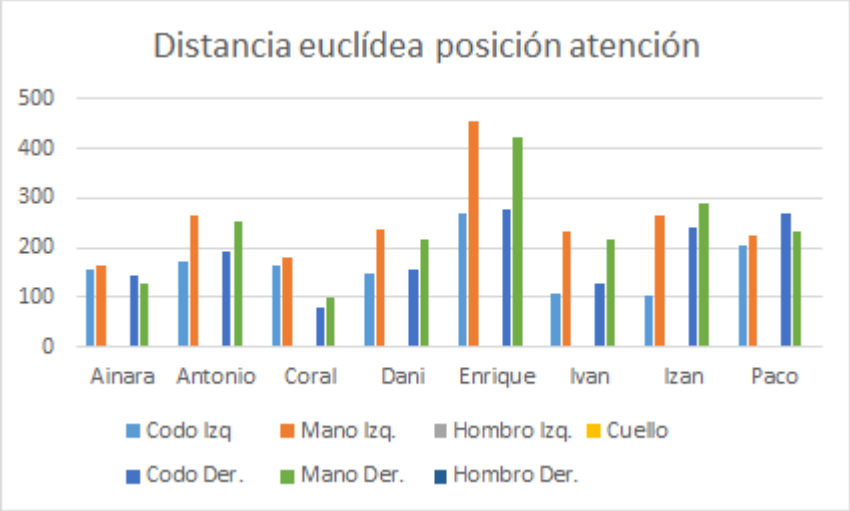


*Figura 77: Diferencia euclídea de la posición "Atención"*

- Diferencia máxima de una articulación: Mano Izquierda Enrique, 453.409 mm.
- Diferencia máxima media de las articulaciones: Mano Izquierda, 251.905375 mm.
- Diferencia máxima total: Enrique, 1419.79631 mm.
- Diferencia media total: 834.992624 mm.

Pose Pensando

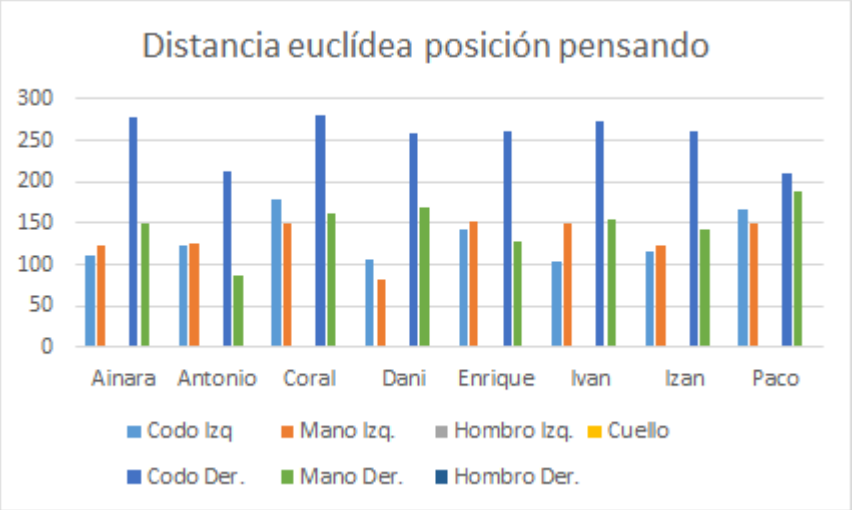


*Figura 78: Diferencia euclídea de la posición "Pensando"*

- Diferencia máxima de una articulación: Codo Derecho Coral, 279.858 mm.
- Diferencia máxima media de las articulaciones: Codo Derecho, 254.129375 mm.
- Diferencia máxima total: Coral, 770.32325 mm
- Diferencia media total: 665.071144 mm.

**Distancia Manhattan**

Pose Despedida

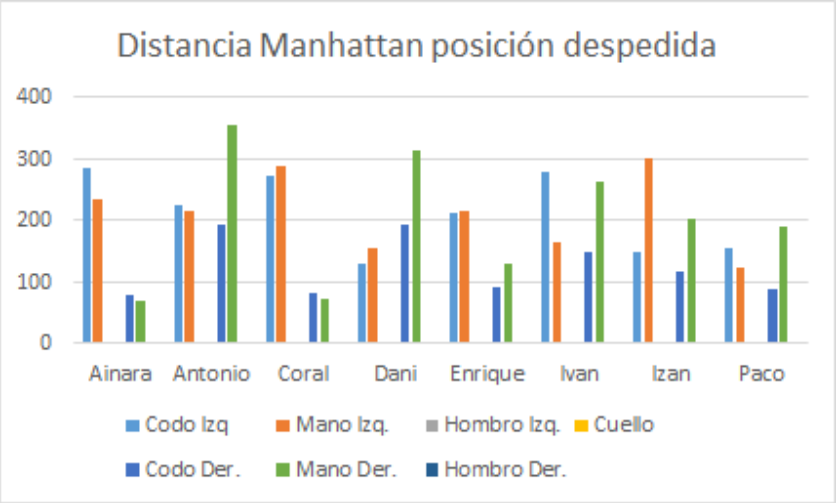


*Figura 79: Diferencia Manhattan de la posición "Despedida"*

- Diferencia máxima de una articulación: Mano Derecha Antonio, 356.956 mm.
- Diferencia máxima media de las articulaciones: Codo Izquierdo, 213.090125 mm.
- Diferencia máxima total: Antonio, 991.195443 mm.
- Diferencia media total: 750.497816 mm.

Pose Neutral

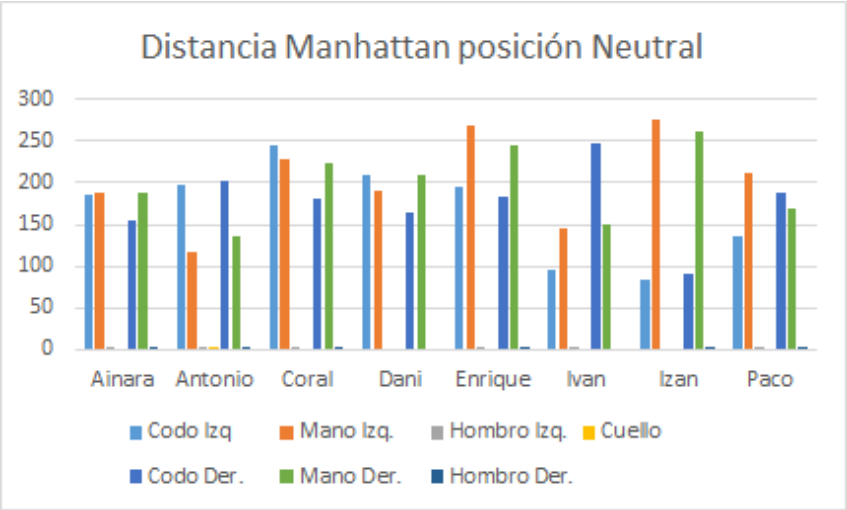


*Figura 80: Diferencia Manhattan de la posición "Neutral"*

- Diferencia máxima de una articulación: Mano Izquierda Izan, 275.098 mm.
- Diferencia máxima media de las articulaciones: Mano Izquierda, 203.453125 mm.
- Diferencia máxima total: Enrique, 892.227488 mm.
- Diferencia media total: 746.639769 mm.

Pose Saludo

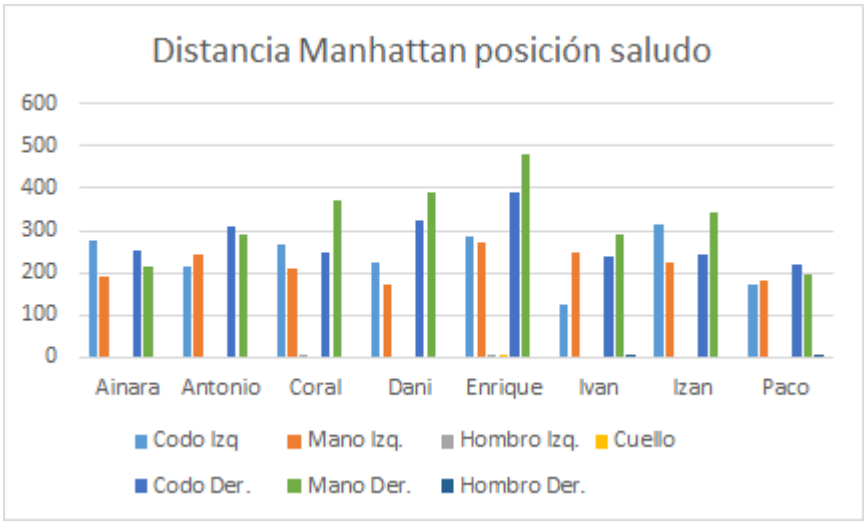


*Figura 81: Diferencia Manhattan de la posición "Saludo"*

- Diferencia máxima de una articulación: Mano Derecha Enrique, 483.217 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 321.451875 mm.
- Diferencia máxima total: Enrique, 1435.35567 mm.
- Diferencia media total: 1052.72134 mm.

Pose Modelo

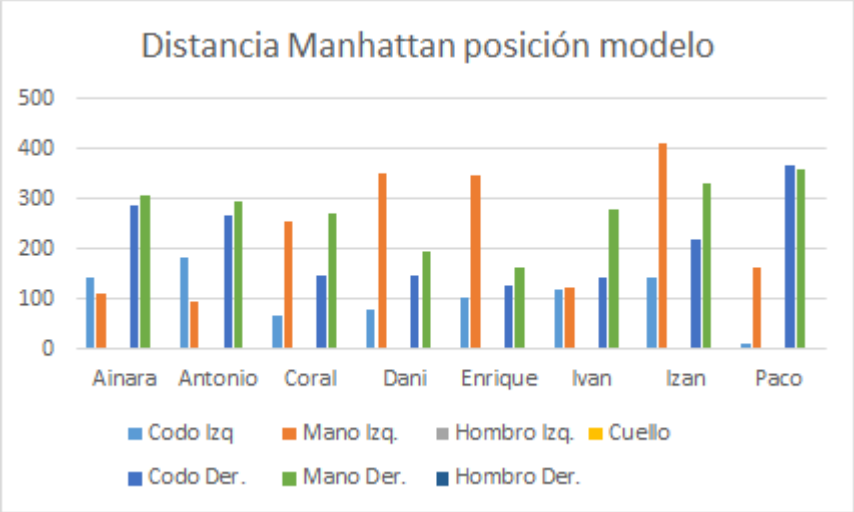


*Figura 82: Diferencia Manhattan de la posición "Modelo"*

- Diferencia máxima de una articulación: Mano Izquierda Izan, 410.796 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 275.06325 mm.
- Diferencia máxima total: Izan, 1105.63244 mm.
- Diferencia media total: 827.92449 mm1

Pose Abrazo

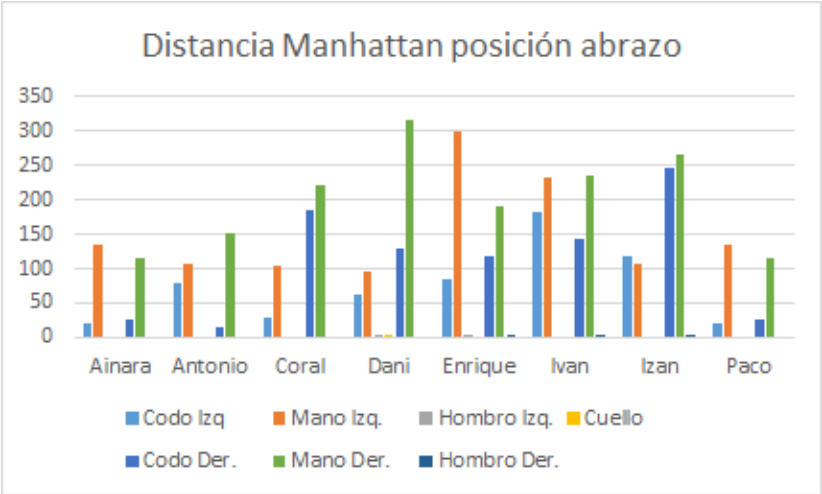


*Figura 83: Diferencia Manhattan de la posición "Abrazo"*

- Diferencia máxima de una articulación: Mano Derecha Dani, 315.872 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 202.18775 mm
- Diferencia máxima total: Iván, 793.662519 mm.
- Diferencia media total: 540.402257 mm.

Pose Protección

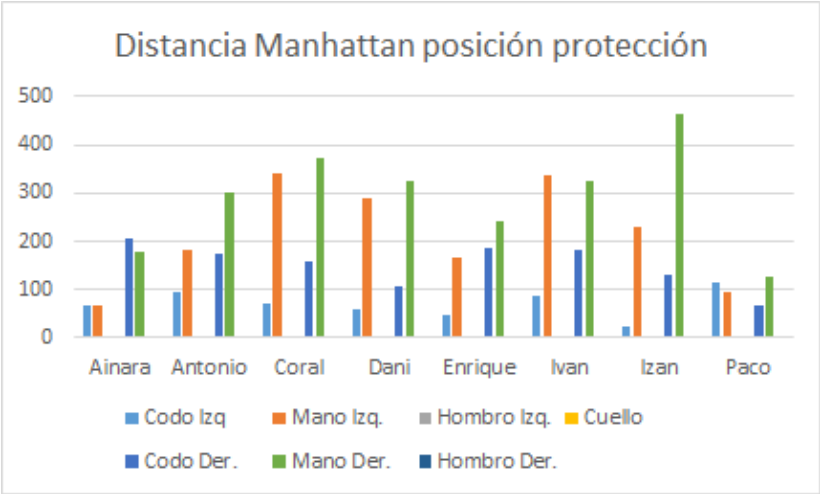


*Figura 84: Diferencia Manhattan de la posición "Protección"*

- Diferencia máxima de una articulación: Mano Derecha Izan, 463.727 mm.
- Diferencia máxima media de las articulaciones: Mano Derecha, 292.503125 mm.
- Diferencia máxima total: Coral, 944.111227 mm.
- Diferencia media total: 708.872733 mm.

Pose Inocente

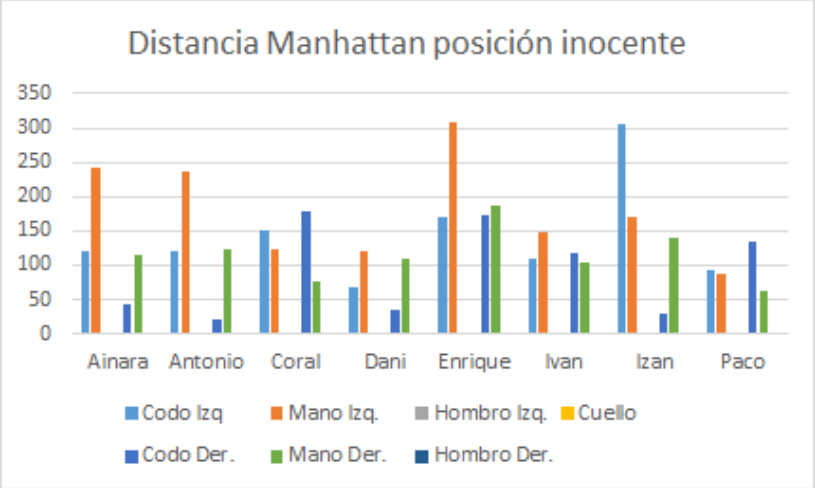


*Figura 85: Diferencia Manhattan de la posición "Inocente"*

- Diferencia máxima de una articulación: Mano Izquierda Enrique, 308.945 mm.
- Diferencia máxima media de las articulaciones: Mano Izquierda, 179.382788 mm.
- Diferencia máxima total: Enrique, 837.983412 mm.
- Diferencia media total: 528.150447 mm.

Pose Preocupación

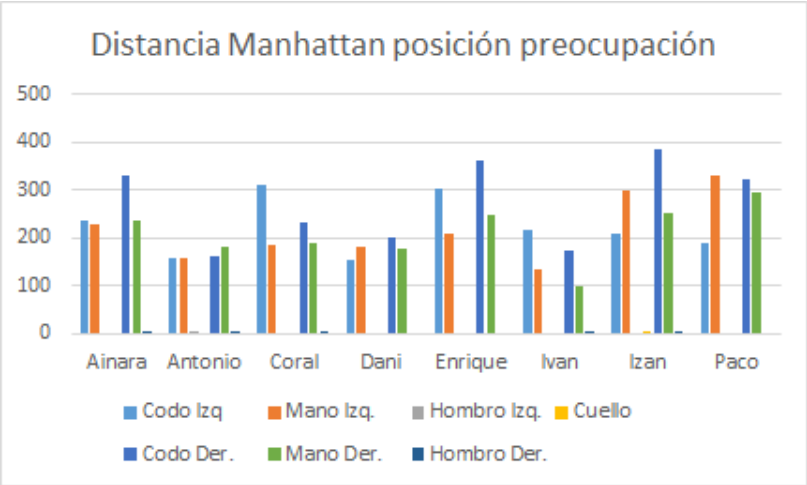


*Figura 86: Diferencia Manhattan de la posición "Preocupación"*

- Diferencia máxima de una articulación: Codo Derecho Izan, 387.311 mm.
- Diferencia máxima media de las articulaciones: Codo Derecho, 271.5615 mm.
- Diferencia máxima total: Izan, 1174.73866 mm.
- Diferencia media total: 919.657753 mm.

Pose Espera

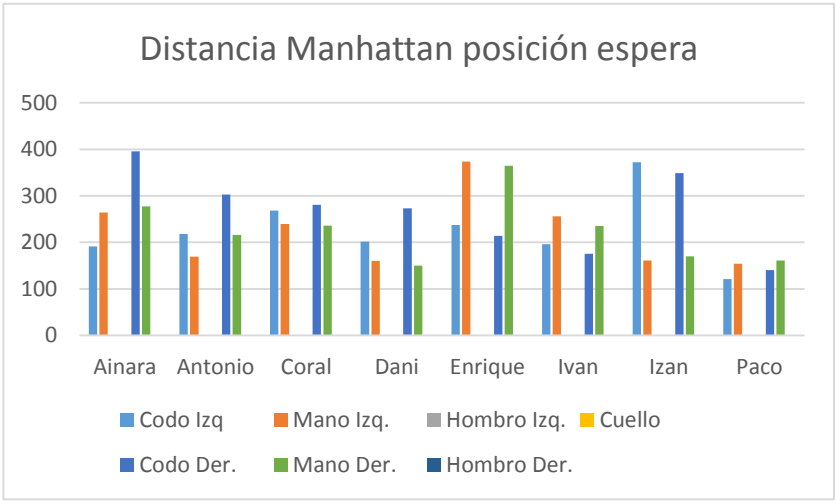


*Figura 87: Diferencia Manhattan de la posición "Espera"*

- Diferencia máxima de una articulación: Codo Derecho Ainara, 395.7 mm.
- Diferencia máxima media de las articulaciones: Codo Derecho, 266.42125 mm.
- Diferencia máxima total: Ainara, 1128.32137 mm.
- Diferencia media total: 941017217 mm.

Pose Atención

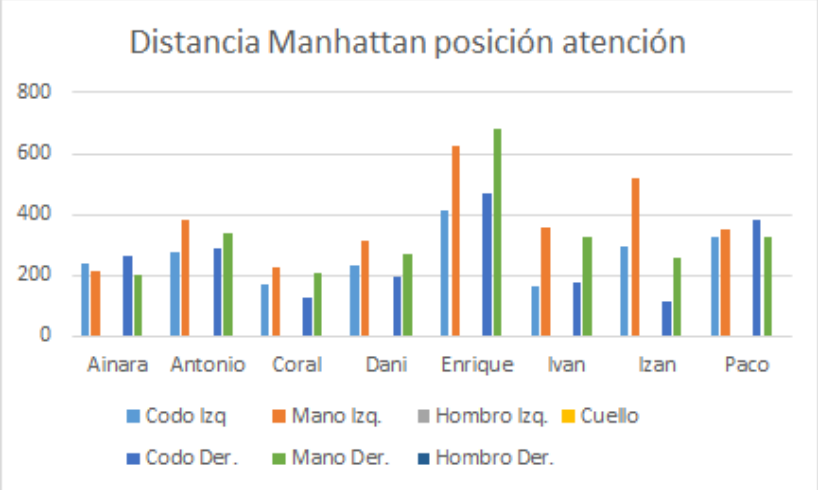


*Figura 88: Diferencia Manhattan de la posición "Atención"*

- Diferencia máxima de una articulación: Mano Derecha Enrique, 683.171 mm.
- Diferencia máxima media de las articulaciones: Mano Izquierda, 375.092375 mm.
- Diferencia máxima total: Enrique, 2196.03737 mm.
- Diferencia media total: 1219.09414 mm.

Pose Pensando

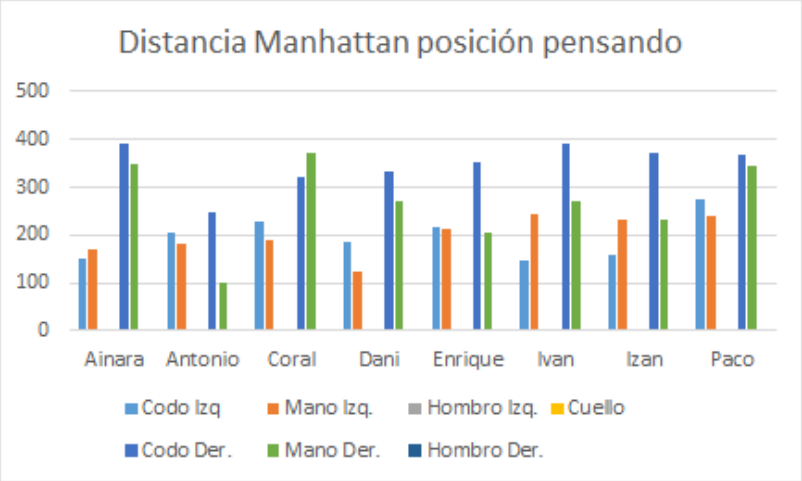


*Figura 89: Diferencia Manhattan de la posición "Pensando"*

- Diferencia máxima de una articulación: Codo Derecho Iván, 390.456 mm.
- Diferencia máxima media de las articulaciones: Codo Derecho, 346.662375 mm.
- Diferencia máxima total: Paco, 1227.37844 mm.
- Diferencia media total: 1010.64992 mm.

**Análisis de resultados**

Es necesario dar un margen de diferencia entre articulaciones en el cual se siga considerando dos posturas iguales. Con esto resolvemos los errores menores de cálculo que NITE2 pueda tener y a la vez flexibilizar las posturas, ya que es imposible que una persona pueda adoptar una postura milímetro a milímetro exactamente igual a la definida en los modelos. Una buena elección es muy importante para que la comparación funcione correctamente. Este margen o umbral, como lo llamaremos de ahora en adelante, debe ser lo suficientemente grande como para evitar falsos negativos. Esto es, dar como diferentes posturas dos que son iguales. Y lo

suficientemente ajustado como para no tener falsos positivos, es decir, dar como válidas posturas que no lo son, evitando de esta manera errores y ambigüedades.

Un ejemplo claro de falso positivo lo vemos en la siguiente imagen:

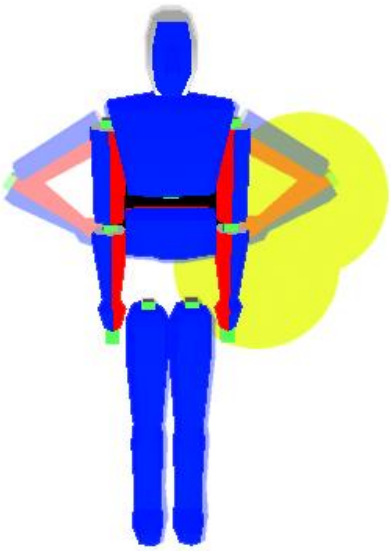

*Figura 90: Problema de falsos positivos provocados por un umbral alto*

En ella se observa cómo el umbral, representado por el círculo que envuelve las articulaciones, deja tanta libertad que la postura "Neutral" puede ser confundida con "Espera" y viceversa. Por lo que, como hemos dicho, es muy importante definir bien este punto para el correcto funcionamiento del sistema.

Analizando las gráficas se han probado por cada una de las dos métricas varios umbrales diferentes. Como en los vídeos de prueba (que se adjuntan al final del documento…) aparecen los usuarios realizando cada una de las posturas correctamente, nos centraremos en evitar falsos negativos, ajustando todo lo posible el umbral. Y posteriormente se comprobará que el umbral está ajustado para que no se den casos de falsos negativos. Esto sería comprobando que en pruebas que definen una postura no se da como buena otra. El conjunto de posturas es

suficientemente amplio, y con posturas lo bastante parecidas, como para poder probar el correcto ajuste del umbral. Los resultados obtenidos son los siguientes:

- Umbral para la distancia Euclídea: Se han probado multitud de umbrales pero cabe destacar los siguientes:
    - 200 mm: Ninguna de las posturas hubiera sido acertada en todos los usuarios, pues para que una postura se considere correcta, todas las articulaciones deben estar distanciadas por debajo del umbral. Y en las pruebas realizadas uno o varios usuarios que consideramos, estaba realizando la postura correctamente, lo rebasaba. Ha dado una media del 88,79% de aciertos de articulaciones correctamente, lo que consideramos no es suficiente. Con este umbral, las posturas que más articulaciones ofrecen por debajo del umbral son "Abrazo y "Protección" con un 98,2%. En cambio la postura "Atención" sólo se han tomado por correctas un 69,6%, por lo que debemos ampliar el umbral.
    - 250 mm: Como es de esperar todas las posturas han mejorado su porcentaje de articulaciones tomadas como buenas. En algunos casos se ha llegado al 100%. Lo que quiere decir, que estas posturas se han acertado siempre. En cambio, hay siete posturas que han dado falsos negativos, volviendo a ser la postura "Atención" la más errada, con un 16,1% de articulaciones falladas. Este umbral, con un 95,61% de articulaciones bien detectadas, queda demostrado que sigue estando por debajo del óptimo buscado.
    - 300 mm: Con este umbral prácticamente se eliminan los falsos negativos. Diez de las once posturas definidas se han detectado siempre, y tiene un porcentaje del 99,67% de articulaciones totales bien detectadas. Finalmente "Atención" queda demostrada como la más problemática, al menos en nuestro conjunto de pruebas, y tiene un porcentaje del 96,4%.

Finalmente, para la distancia euclídea hemos tomado un umbral de 300 mm, que aunque en nuestras pruebas no ha dado un funcionamiento perfecto, lo vemos lo bastante correcto como para elegirlo, teniendo de esta manera también un umbral lo bastante ajustado como para evitar falsos positivos. Podemos ver la gráfica con los resultados explicados anteriormente.

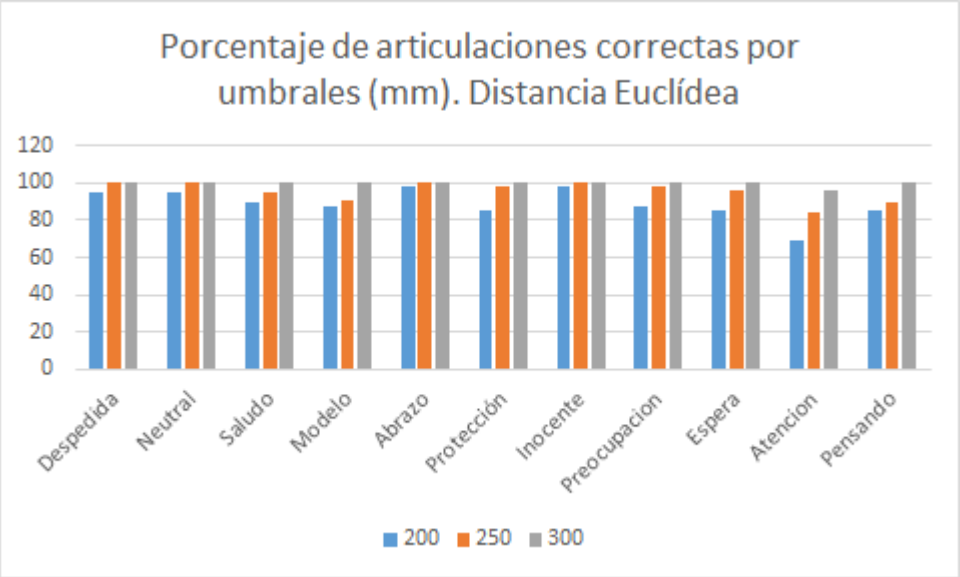


*Figura 91: Comparación de articulaciones acertadas con distintos umbrales en la distancia euclídea*

- Umbral para la distancia Manhattan: Los umbrales para esta métrica deben ser superiores a los elegidos para la distancia euclídea, pues las diferencias que resultan son mayores. Esto lo veíamos en el apartado [5.4.1.2.3], y gráficamente en la [Figura 65]. Destacamos los siguientes para ejemplificar el proceso, como hicimos antes:
    - 300 mm: Este umbral tiene un porcentaje de articulaciones correctamente colocadas detectadas del 89,6 %. Únicamente la posición neutral se ha detectado en todos los usuarios. Como no podía ser de otra manera, la postura Atención ha sido la que posee el menor número de articulaciones detectadas, con un 75%. Como vemos, es un umbral insuficiente, pues tiene demasiado falsos negativos.

- 350 mm: Este umbral mejora notablemente el total de aciertos, aumentándolo hasta un 94,96%. Cuatro de las once posturas han sido detectadas siempre, y las que no lo han sido tienen un porcentaje alto de acierto, por lo que sigue habiendo margen de mejora.
- 400 mm: El último umbral, el más alto probado nos ofrece buenos resultados. Tiene un 98,7% de articulaciones detectadas. Siete de las once posturas han sido acertadas siempre. De las cuatro posturas restantes que no fueron descubiertas por el algoritmo, en tres de ellas el error se produjo debido a que sólo una de las articulaciones superaba el umbral. Ampliando este umbral ampliaríamos el porcentaje de acierto, pero 40 cm de margen en cada articulación en distancia Manhattan lo consideramos como una distancia importante, que en el caso de seguir aumentándola podríamos caer en falsos positivos.

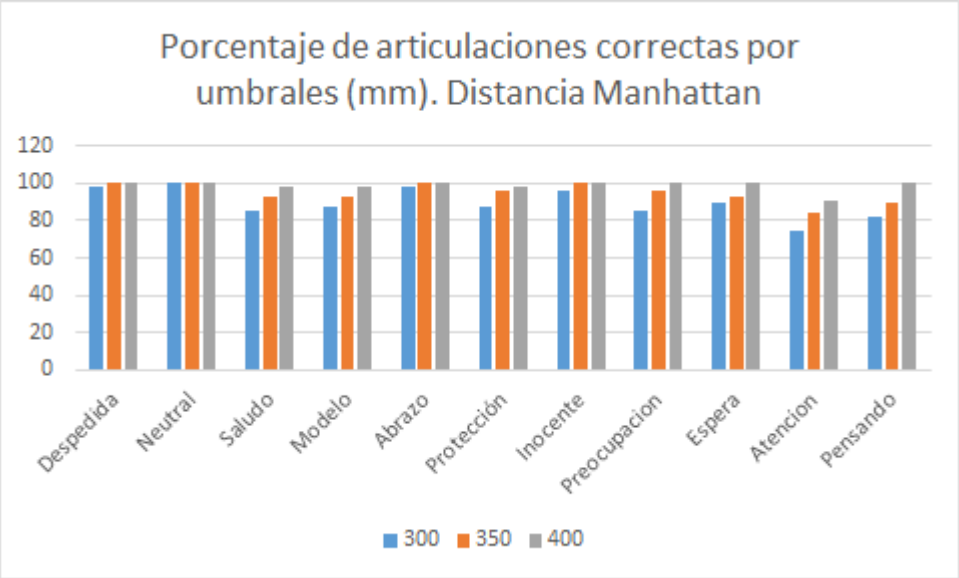


*Figura 92: Comparación de articulaciones acertadas con distintos umbrales en la distancia Manhattan*

Vistos los resultados con varios umbrales podemos concluir varios aspectos sobre la métrica:

- Comportamiento muy similar: Ambas métricas funcionan de una manera semejante, la forma de ambas gráficas son muy parecidas Esto era de suponer, pues son de la misma familia, pero con este estudio ha quedado demostrado.
- Rangos diferentes: Es la principal diferencia, y la que nos hará decantarnos por una u otra. La distancia Manhattan necesita rangos mayores de distancia para diferenciar las posturas correctas de las incorrectas. Mientras que para la distancia euclídea nos hemos decantado por un rango de 300 mm con una efectividad del 99,67%, para la distancia Manhattan necesitamos un rango mayor (400 mm) para una efectividad menor (98,7%). Dicho de otra manera, la distancia euclídea para los mismos resultados se mueve en rangos menores, por lo que es menos probable caer en falsos positivos. Por ello, nos decantaremos por la **distancia euclídea** como la métrica más válida **y 300 mm** para el umbral óptimo.

Articulaciones con más error:

En la siguiente gráfica podemos ver la distancia media resultante cada articulación con cada métrica. De un simple vistazo concluimos que cuanto más alejada está la articulación del torso, o más libertad de movimiento tiene, o más profundo está en el árbol de InnerModel, más diferencia existe. El cuello y los hombros tienen una diferencia despreciable, pues no tienen prácticamente libertad de movimiento. Las manos, seguidas de los codos, pueden considerarse las más problemáticas

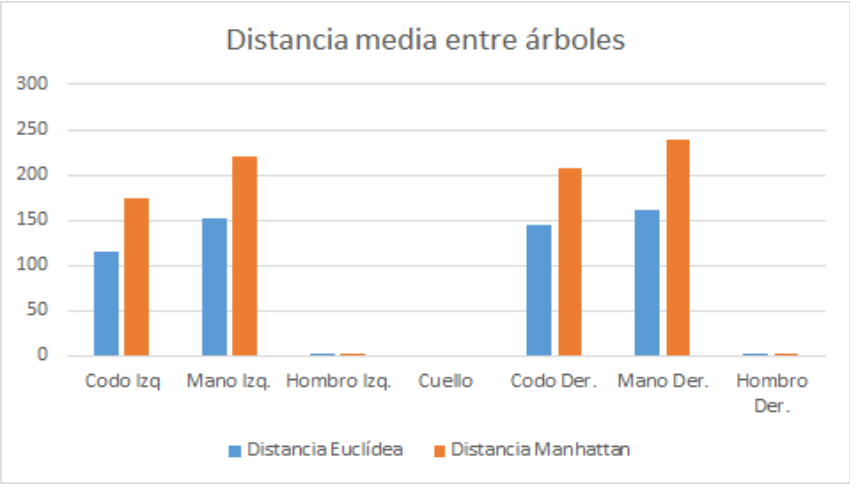


*Figura 93: Diferencia media de articulaciones homónimas entre dos árboles distintos*

Posturas con más error:

También hemos hecho un estudio de las posturas definidas. Hemos calculado el error total medio de cada una. Esto es, la media de la suma de las diferencias de las articulaciones por cada postura. Para así ver qué posturas son las que más problemas presentan e intentar descubrir por qué para futuros modelos que se quieran añadir al sistema, o para mejorar los que ya hay.

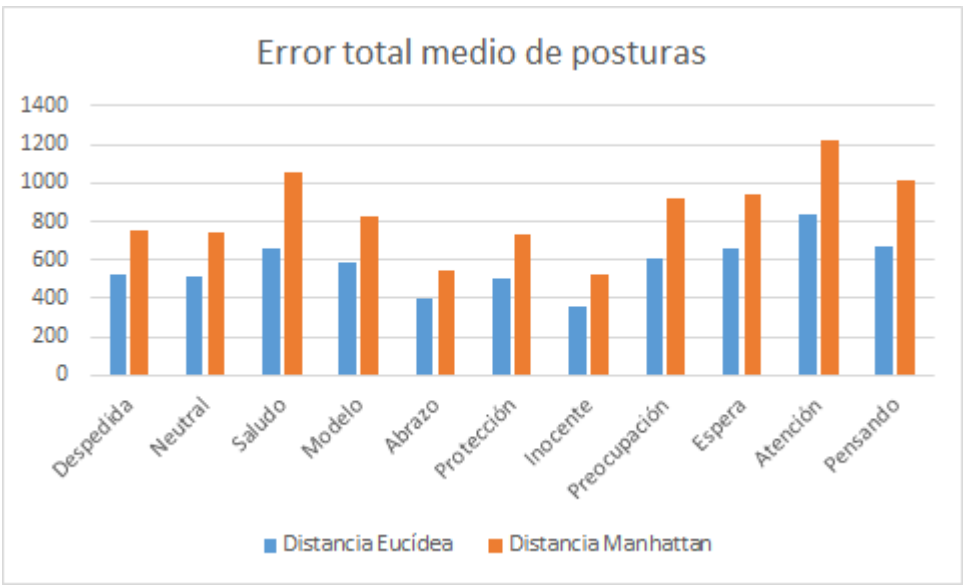


*Figura 94: Error total medio de las posturas*

Como comentamos anteriormente, la postura que más ha costado detectar ha sido “Atención”. Por eso vamos a reflexionar sobre ella probablemente el error sea inherente a su definición ya que es poco para natural para el ser humano.

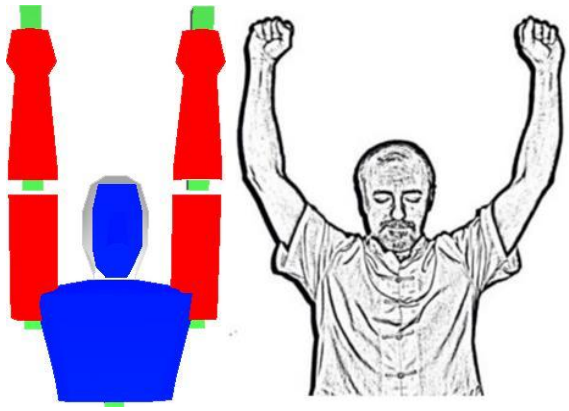

*Figura 95: Problema de definición de posturas poco humanas*

En la imagen podemos ver cómo está definida, demasiado extendida y rígida, y a su derecha cuál es la manera más cómoda del ser humano para hacer la pose. Hay diferencias notables que desembocarán en errores inevitables. Lo que nos hace darnos cuenta de la importancia de definir las poses de la manera más humana posible. Y no sólo fijándonos en el concepto de lo que buscamos, que en este caso serían los brazos levantados, o el coloquial: “manos arriba”.

Lo mismo creemos que ocurre con la postura “Pensando”, (quizás inspirada por el cine mudo o los dibujos) se busca tener una mano en la cabeza, pero la forma definida no es la más natural de hacerlo.

Por otro lado, el saludo es la segunda postura más errada. Creemos que la causa es la amplitud de posibilidades que pueden darse al tomar la postura.

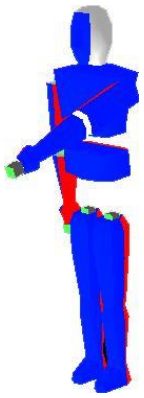

*Figura 96: Problema de definición de posturas ambiguas*

La mano en el modelo está definida a la altura que vemos en la imagen, pero depende de la persona y del momento, la altura de la misma puede variar notablemente. Induciendo los errores que hemos visto. Para casos como este, adoptar un umbral como hemos hecho es totalmente necesario, puesto que da cierta libertad de movimiento para adoptar la postura deseada. En cambio, posturas como "Protección", "Abrazo" e "Inocente" se ha comprobado que están bien definidas, pues dan resultados bastantes buenos.

En esta última gráfica se puede ver perfectamente lo que comentábamos antes, y es el comportamiento semejante de las métricas. Ambas crecen y decrecen de la misma manera y en los mismos puntos. En la gráfica [Figura 94] presentan la misma forma pero a distintas alturas (en umbrales distintos).

## 6.2. DETECCIÓN MÚLTIPLE DE POSES

En este experimento se pretende poner a prueba el reconocimiento múltiple de posturas. Esto consiste en que el sistema analiza simultáneamente a cada una de las personas que aparecen en la escena, detectando cada una de las poses de forma individual.

Para las pruebas se han grabado cinco vídeos en los que aparecen dos personas realizando poses simultáneamente. En cada video cada persona realiza una postura diferente. Por lo que el grueso total está formado por diez posturas realizadas en cinco vídeos de dos personas cada uno. Se ha dejado fuera del experimento la pose neutral.

Las gráficas que vemos a continuación se muestran las diferencias entre articulaciones obtenidas al comparar los árboles cinemáticos en cada una de las poses en los vídeos de detección múltiple junto con las distancias de la misma persona en la misma postura en la detección simple, y la media de distancias de todos los usuarios en la misma pose en el experimento anterior. De esta manera comprobaremos si influye de alguna manera la aparición de varias personas en la misma escena.

**Distancia euclídea**

**Vídeo 1**

Pose Atención

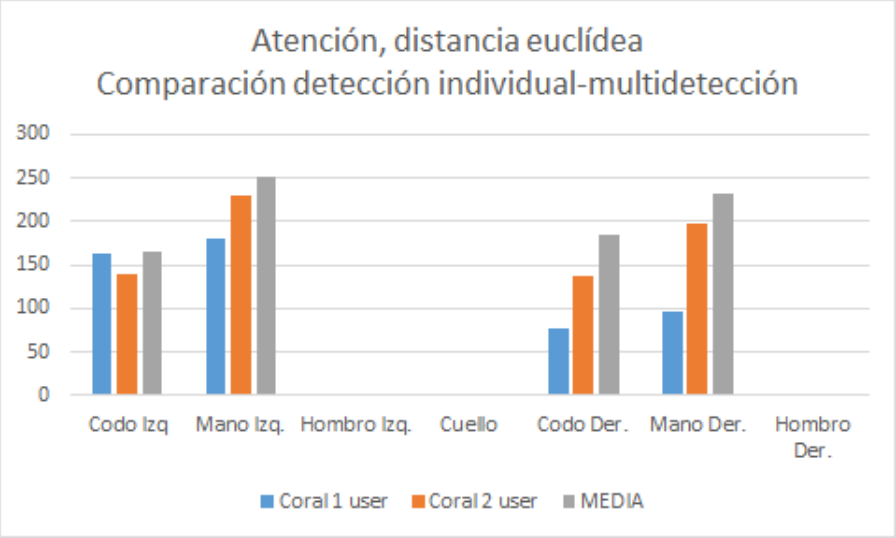


*Figura 97: Comparación detección simple vs multi-deteccion, pose "Atención", distancia euclídea*

En esta gráfica podemos ver que la pose "Atención" realizada por Coral en el video de dos personas. Se pueden ver las diferencias euclídeas de la comparación generadas en este vídeo, las diferencias euclídeas del video individual de Coral realizando esta misma pose y la media de las diferencias de las articulaciones de los ocho usuarios del experimento anterior realizando esta pose. Vemos como por ejemplo, en el codo, la diferencia en el vídeo múltiple está a la par con la media, mientras que en el video individual estaba un poco por debajo. Pero en este caso las diferencias son mínimas, no olvidemos que las distancias están en milímetros. En el caso de la mano izquierda el codo derecho y la mano derecha la distancia calculada en este experimento está entre la diferencia del video individual y la media.

A continuación mostramos los resultados de las posturas y métricas restantes y posteriormente las analizaremos para extraer algunas conclusiones.

Pose Modelo

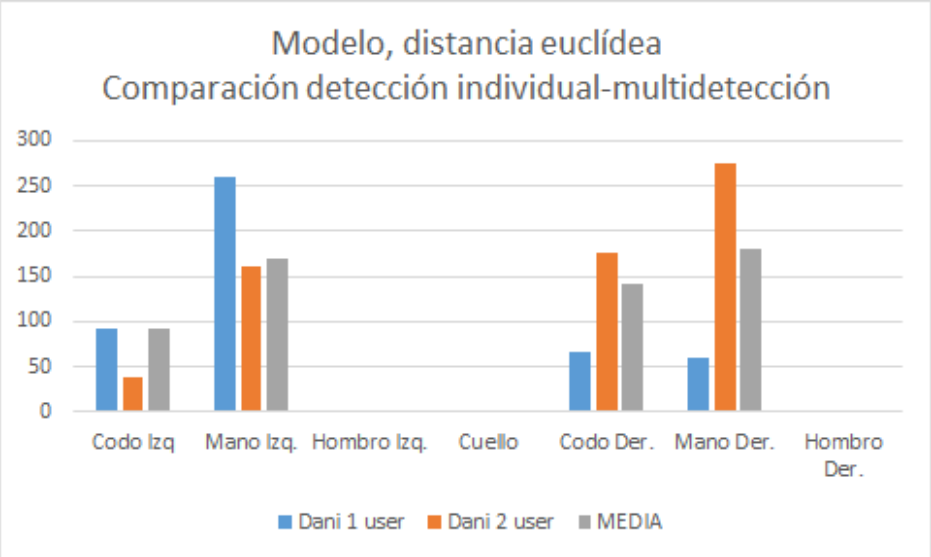


*Figura 98: Comparación detección simple vs multi-deteccion, pose "Modelo", distancia euclídea*

**Vídeo 2**

Pose Despedida

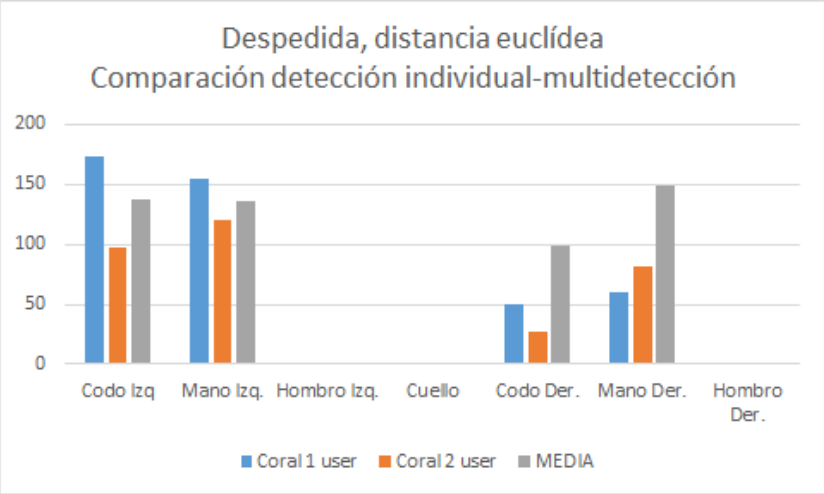


*Figura 99: Comparación detección simple vs multi-deteccion, pose "Despedida", distancia euclídea*

Pose Protección

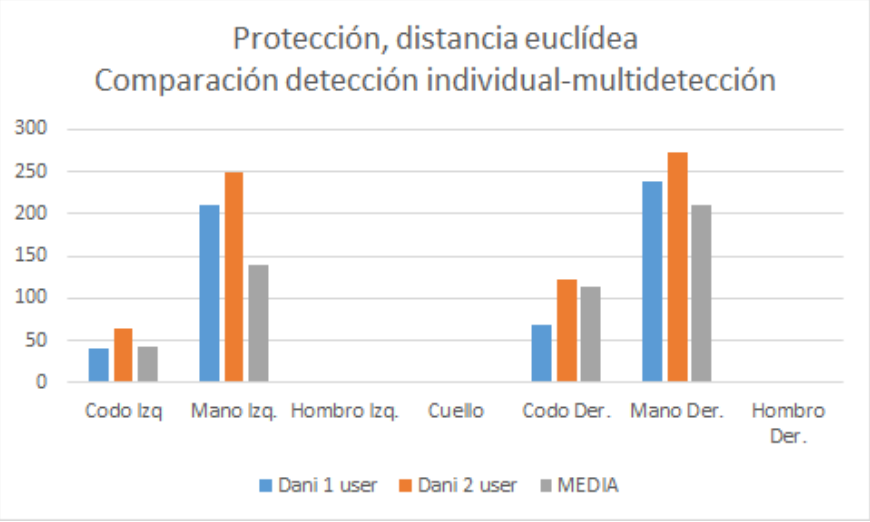


*Figura 100: Comparación detección simple vs multi-deteccion, pose "Protección", distancia euclídea*

**Vídeo 3**

Pose Espera

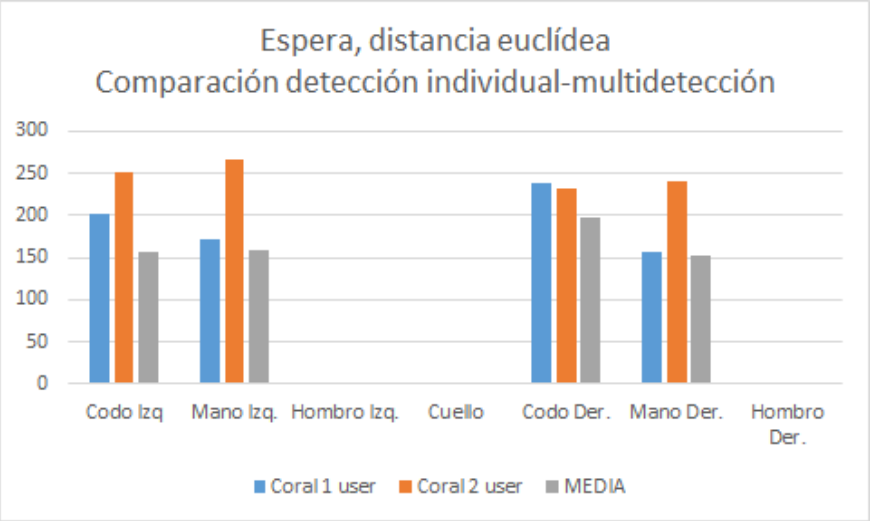


*Figura 101: Comparación detección simple vs multi-deteccion, pose "Espera", distancia euclídea*

Pose Inocente

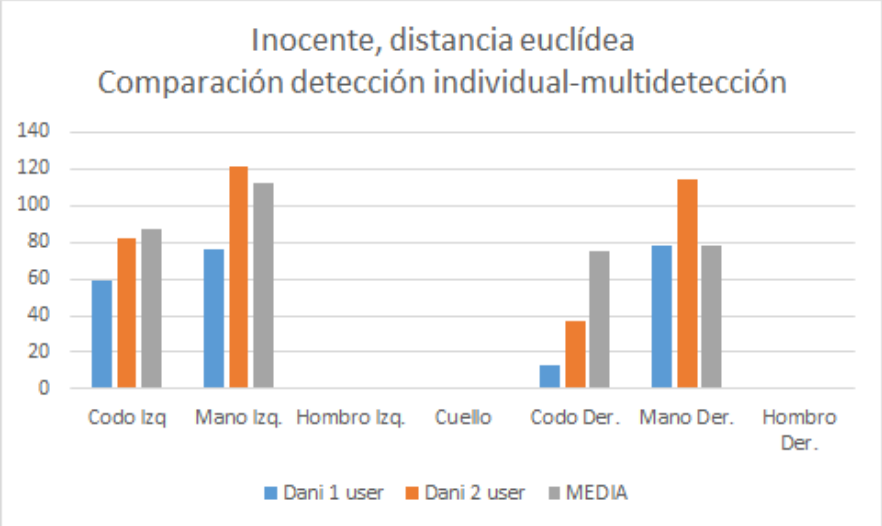


*Figura 102: Comparación detección simple vs multi-deteccion, pose "Inocente", distancia euclídea*

**Vídeo 4**

Pose Preocupación

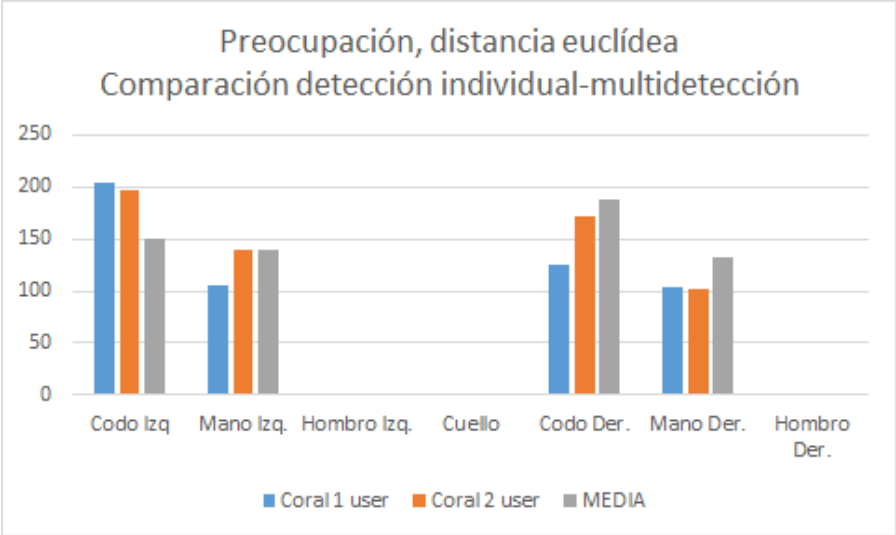


*Figura 103: Comparación detección simple vs multi-deteccion, pose "Preocupación", distancia euclídea*

Pose Pensando

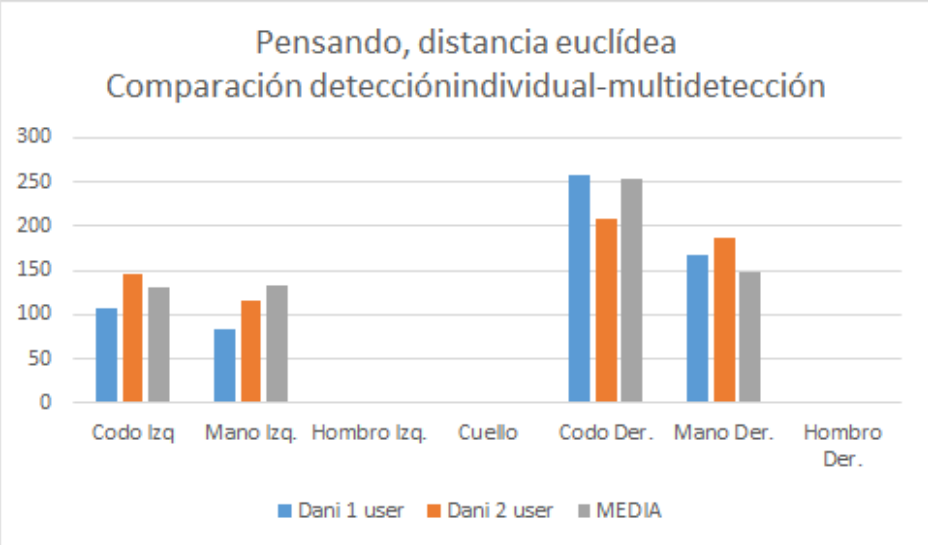


*Figura 104: Comparación detección simple vs multi-deteccion, pose "Pensando", distancia euclídea*

**Vídeo 5**

Pose Abrazo

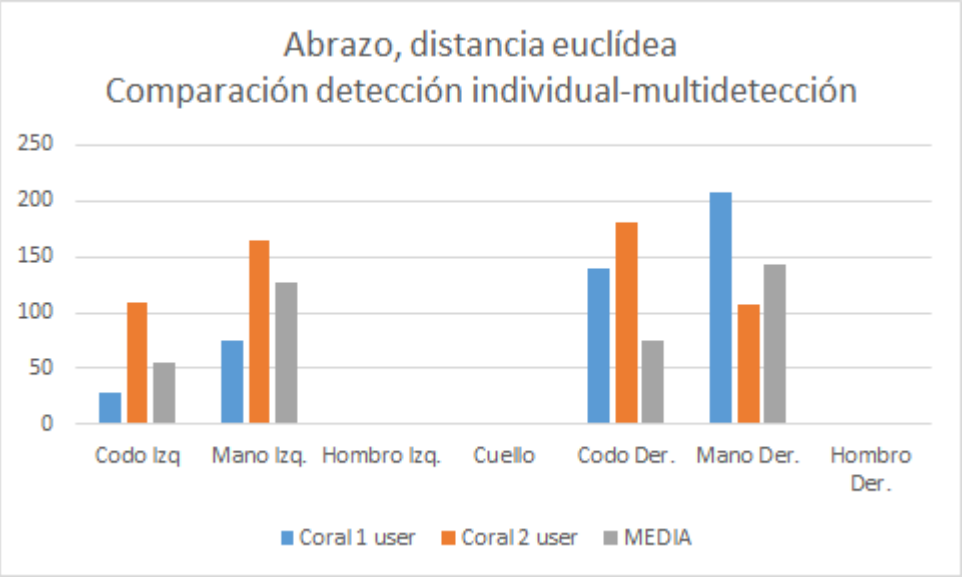


*Figura 105: Comparación detección simple vs multi-deteccion, pose "Abrazo", distancia euclídea*

Pose Saludo

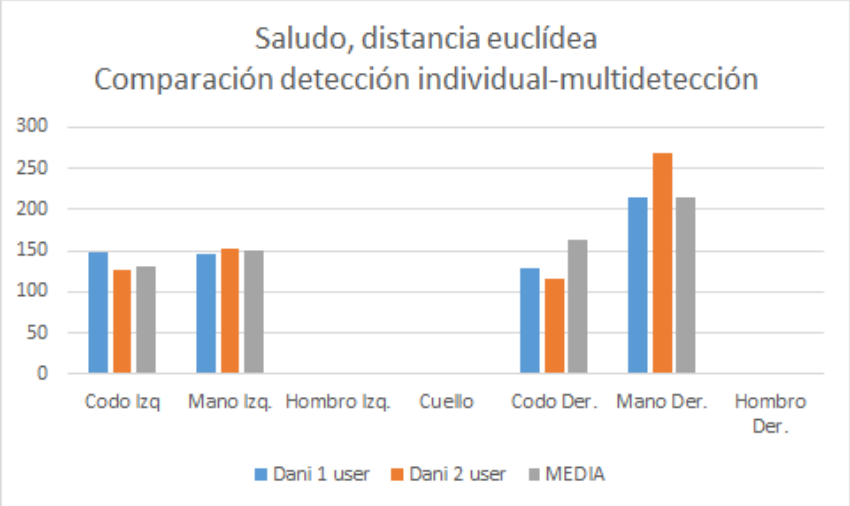


*Figura 106: Comparación detección simple vs multi-deteccion, pose "Saludo", distancia euclídea*

## Distancia Manhattan

### Vídeo 1

Pose Atención

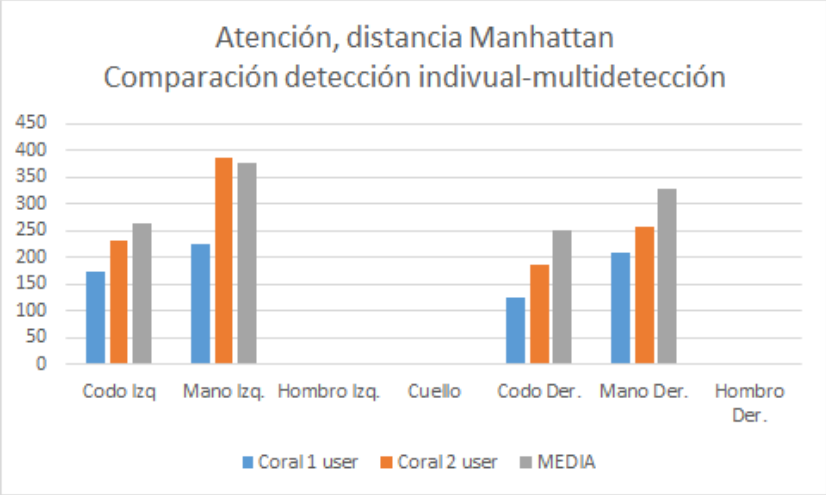


*Figura 107: Comparación detección simple vs multi-deteccion, pose "Atención", distancia Manhattan*

Pose Modelo

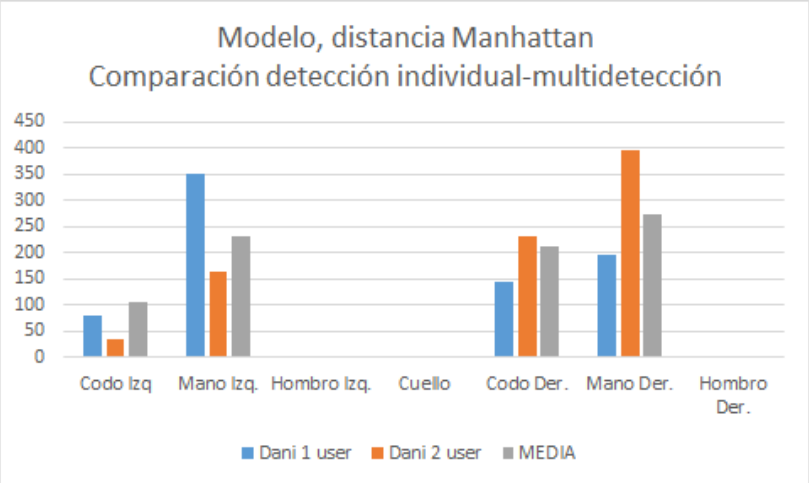


*Figura 108: Comparación detección simple vs multi-deteccion, pose "Modelo", distancia Manhattan*

**Vídeo 2**

Pose Despedida

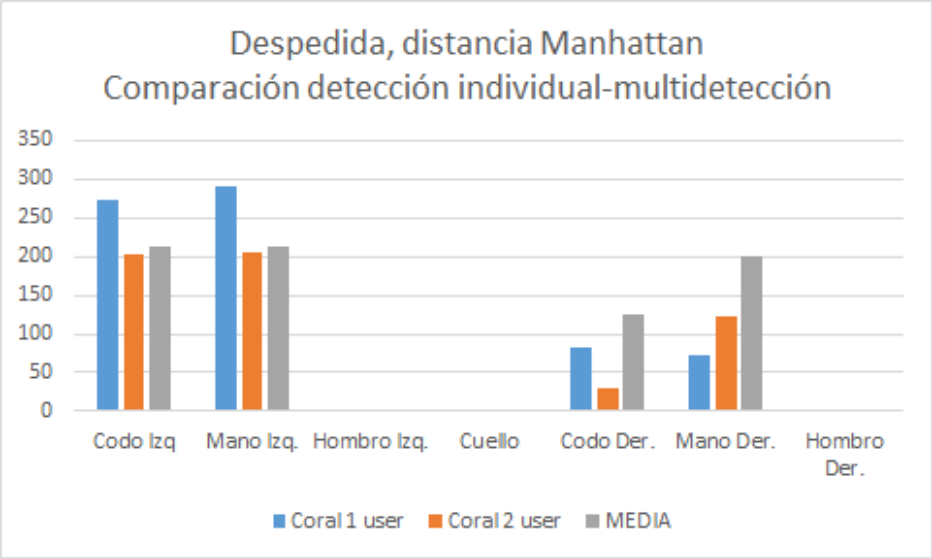


*Figura 109: Comparación detección simple vs multi-deteccion, pose "Despedida", distancia Manhattan*

Pose Protección

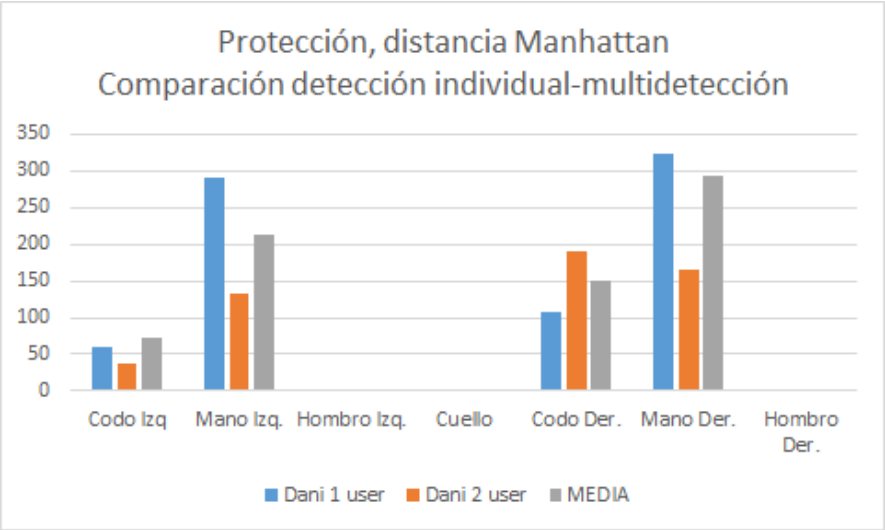


*Figura 110: Comparación detección simple vs multi-deteccion, pose "Protección", distancia Manhattan*

**Vídeo 3**

Pose Espera

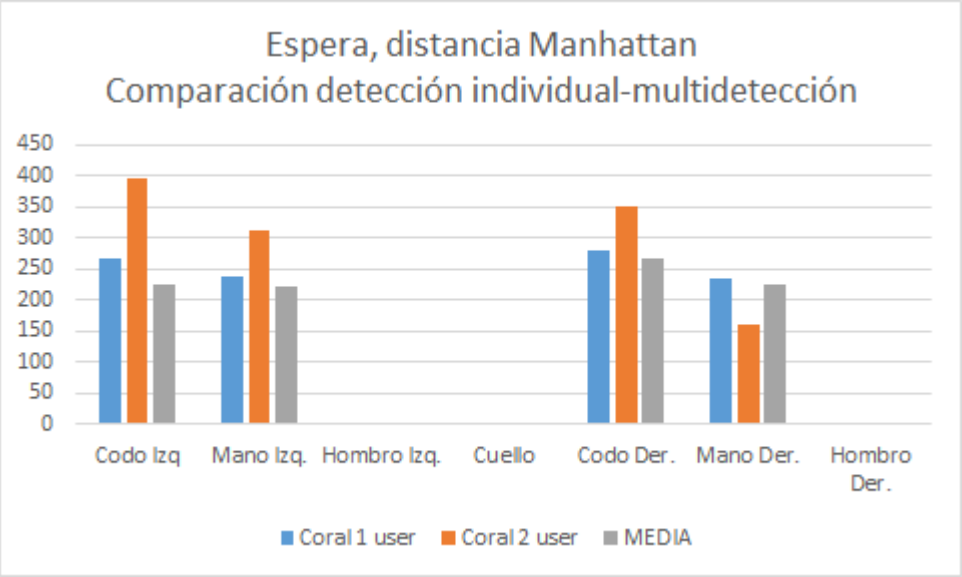


*Figura 111: Comparación detección simple vs multi-deteccion, pose "Espera", distancia Manhattan*

Pose Inocente

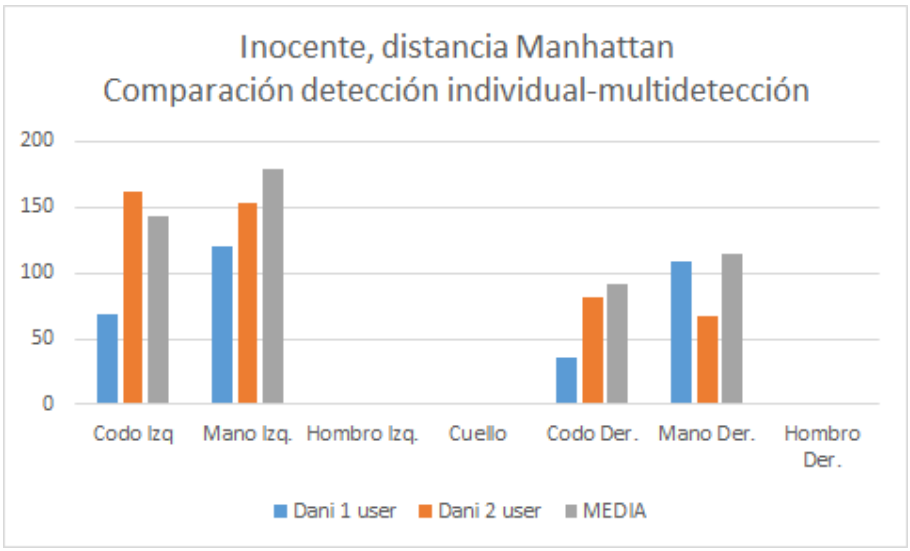


*Figura 112: Comparación detección simple vs multi-deteccion, pose "Inocente", distancia Manhattan*

**Vídeo 4**

Pose Preocupación

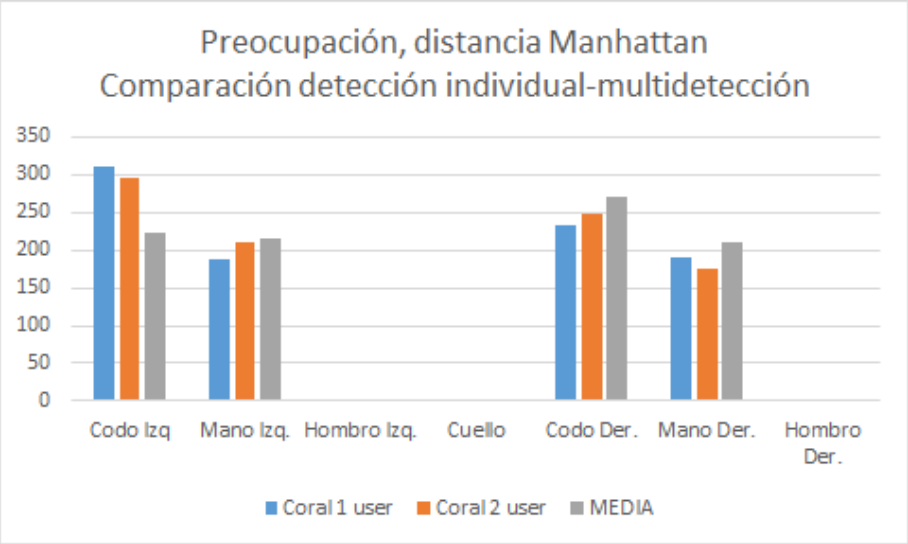


*Figura 113: Comparación detección simple vs multi-deteccion, pose "Preocupación", distancia Manhattan*

Pose Pensando

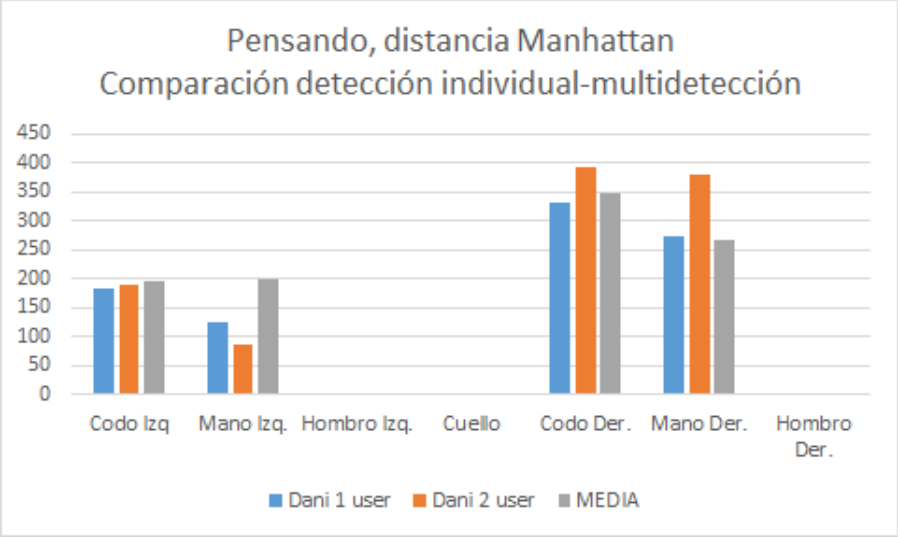


*Figura 114: Comparación detección simple vs multi-deteccion, pose "Pensando", distancia Manhattan*

**Vídeo 5**

Pose Abrazo

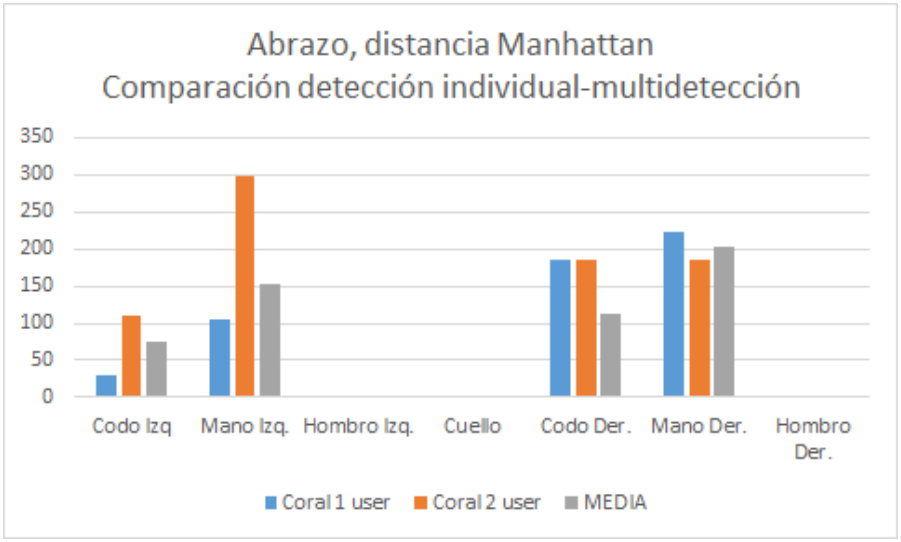


*Figura 115: Comparación detección simple vs multi-deteccion, pose "Abrazo", distancia Manhattan*

Pose Saludo

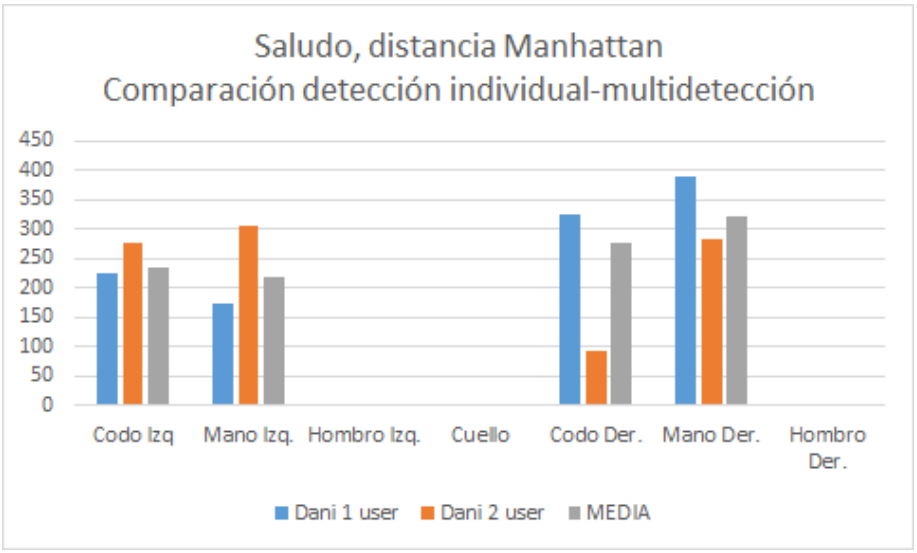


*Figura 116: Comparación detección simple vs multi-deteccion, pose "Saludo", distancia Manhattan*

**Análisis de los resultados**

- En las gráficas anteriores no existe un patrón claro que nos ayude a definir el mejor o peor funcionamiento de la multidetección de poses. En ocasiones las diferencias de la multidetección son menores que la media y la detección simple, en otras mayores, en otras están a la par... repartido estos casos en partes iguales. Esto hace pensar que no importa si el algoritmo analiza a una persona o varias a la vez, pues las diferencias únicamente dependen de la precisión con la que se tome la postura y no influye el número de personas que estén en escena.
- Articulaciones con más error: Con las gráficas anteriores se puede intuir que la multidetección funciona de igual manera que la detección simple, pero para confirmarlo veamos la diferencia media de cada articulación en cada uno de los casos y métricas.

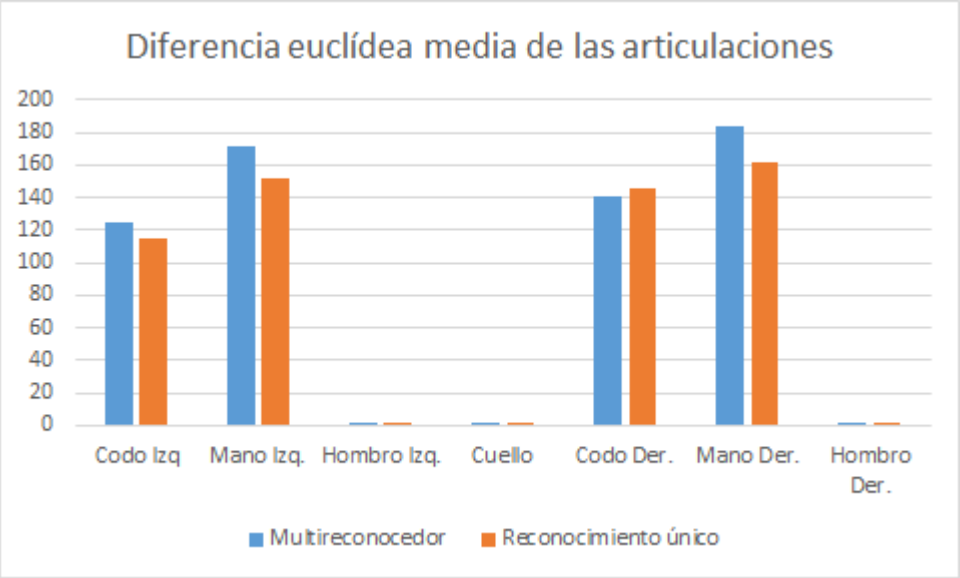


*Figura 117: Diferencia euclídea media de las articulaciones, detección simple vs multi-detección*

En la gráfica superior podemos ver la distancia euclídea media de cada articulación tanto en la multidetección como en la detección individual. De un simple vistazo se intuye un comportamiento similar de ambos. En tres de los siete casos la diferencia de la multidetección es superior, pero no más de unos 2 cm. Diferencia despreciable en el espacio en el que nos movemos.

Existe una gran diferencia entre la cantidad de poses probadas en ambos casos, mientras que en el reconocimiento único se han analizado un total de 88 poses, en la multidetección únicamente contamos con 10, por lo tanto creemos que aumentando el número de pruebas, las diferencia de distancias que hay entre el caso multi e individual se irían minimizando, llegando a ser prácticamente iguales en cuanto el número de pruebas fuera suficientemente alto.

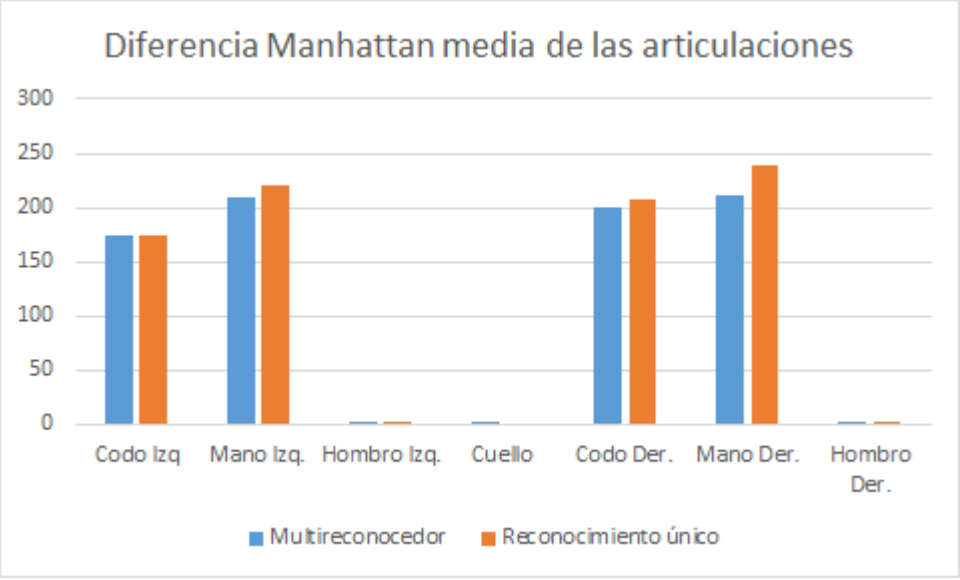


*Figura 118: Diferencia Manhattan media de las articulaciones, detección simple vs multi-detección*

En el caso de la distancia Manhattan ocurre lo mismo que con la distancia euclídea, la diferencia entre el multi-reconocedor y el reconocimiento simple es mínima. Las diferencias medias varían de 0 cm a 3 cm, distancia totalmente despreciable en un espacio tres dimensiones y con un rango como en el que nos movemos.

- Posturas con más error: En este caso vamos a hacer un análisis de las posturas definidas, viendo cómo han funcionado en el multi-reconocedor en comparación con el reconocimiento simple. En las siguientes gráficas se muestran las distancias totales medias de cada pose, es decir la media de la suma de los errores de las articulaciones para cada postura. En realidad en el caso del multi-reconocimiento no es una media propiamente dicha, pues sólo contábamos con un caso para cada postura, siendo la media el resultado de ese único caso.

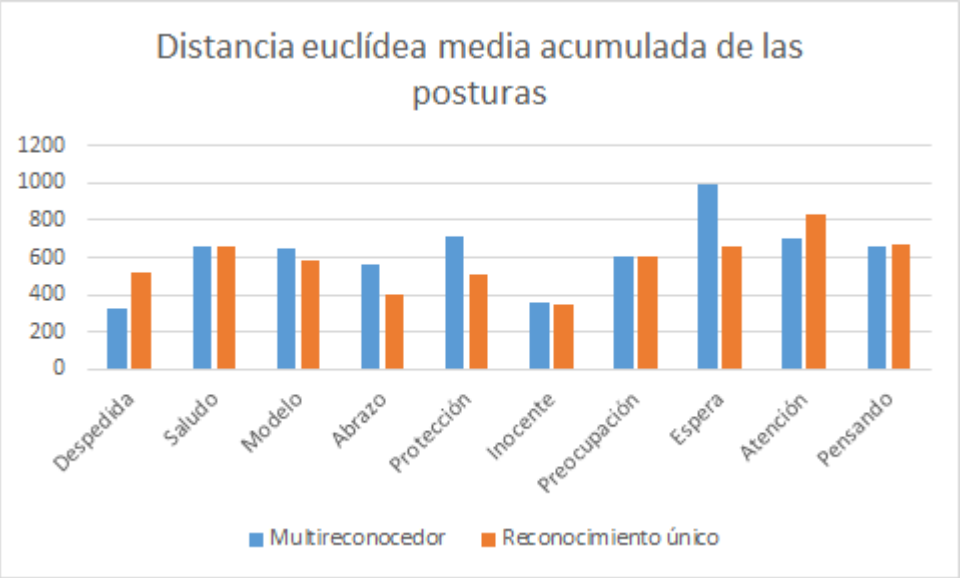


*Figura 119: Distancia euclídea media total de las posturas, detección simple vs multi-detección*

Fijándonos en la gráfica superior, y analizando la distancia euclídea total media de cada postura vemos que en algunos casos son muy similares (“Pensando”, “Preocupación”, “Saludo”, ”Modelo”, ”Inocente”) y en otro casos varía en mayor o menor medida. La diferencia más notable la tenemos en “Espera”, donde hay una diferencia de unos 40 cm totales, que equivale a una diferencia media de error entre articulaciones de unos 5,7 cm.

Esta diferencia viene provocada por los pocos casos de prueba por postura con los que cuenta este experimento. Contamos con un caso por postura, mientras que en el reconocimiento único había ocho pruebas. En cualquier caso, en casi la mitad de las posturas se obtienen resultados muy parejos, y las posturas más difíciles de descubrir coinciden en ambos tipos de reconocimiento. Las diferencias que existen en los casos más dispares estamos seguros que se suavizarían conforme aumenten los casos de prueba para la multidetección.

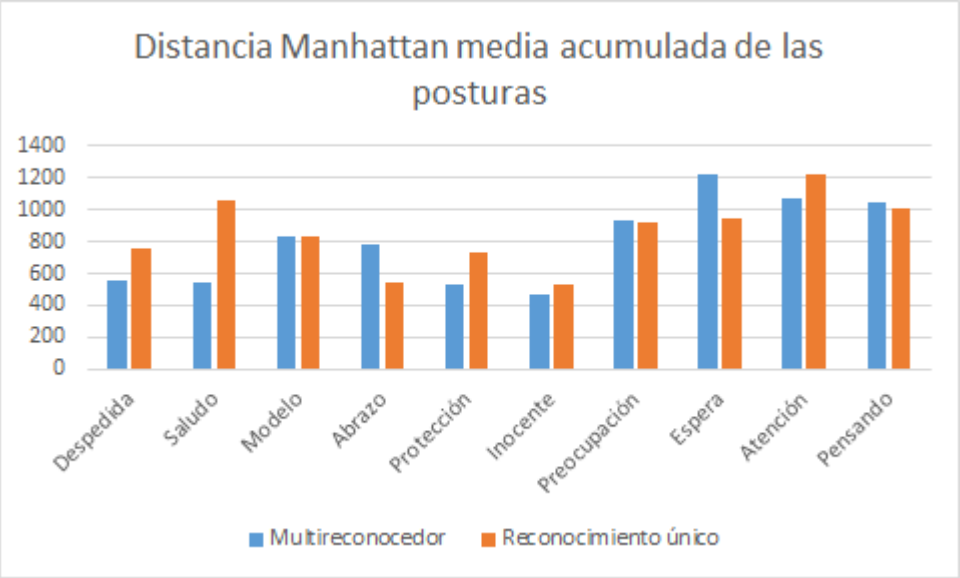


*Figura 120: Distancia Manhattan media total de las posturas, detección simple vs multi-detección*

Con la distancia Manhattan ocurre exactamente lo mismo, aunque en este caso las diferencias son menores, salvo en el caso de la pose “Saludo” cuya diferencia en la multidetección es mucho menor a la media de la detección simple, provocado igualmente por los pocos casos por postura probados en el experimento. Igualmente, las posturas no reconocidas coinciden casi en su totalidad en ambos casos, por lo que como hemos dicho antes, también se nivelarían las diferencias existentes con un mayor número de pruebas.

Los datos generales extraídos en este segundo experimento han demostrado algunas diferencias en los resultados en cuanto al estudio de las poses de la detección múltiple y simple, llegando a la conclusión que han sido provocados por la cantidad de casos estudiados de uno y otro. Para una mayor fiabilidad hubieran sido necesarias más pruebas por cada pose. Aun con este hándicap, viendo la multitud de similitudes que existen entre ambos reconocimientos en las poses y sobre todo en el estudio de las diferencias entre articulaciones, podemos concluir un comportamiento de la multidetección igual al del reconocimiento simple.

La conclusión obtenida del correcto funcionamiento de la multidetección para dos personas, demuestra el correcto funcionamiento para la multidetección de N personas dentro de la escena, siendo N un valor limitado por las restricciones de la librería NITE2.

# 7. CONCLUSIÓN Y TRABAJOS FUTUROS

## 7.1. CONCLUSIÓN DEL TRABAJO REALIZADO

Después del desarrollo del proyecto es importante sacar conclusiones sobre los resultados obtenidos, sobre el camino recorrido y por recorrer y también sobre las sensaciones personales durante y después de su realización.

Creemos que se han logrado unos resultados palpables y demostrados. Si bien no se ha alcanzado finalmente todo lo planeado, básicamente debido a la complejidad de aunar directamente posiciones corporales y *affordances*, quizás demasiado novedoso y ambicioso para un proyecto fin de carrera, si se han puesto los cimientos para un estudio más desarrollado y complejo de los *affordances* en la robótica. Además la solución implementada ramifica el camino en varias sendas de investigación pudiendo, incluso, ser el soporte o la base para profundizar en distintos temas relacionados con la Human Robot Interaction (HRI).

Como todos los proyectos, este también tiene debilidades y fortalezas que merece la pena señalar, para tenerlas presentes en su uso y en futuros desarrollos.

Como puntos débiles destacamos los siguientes:

- En la detección de poses no se han introducido las piernas y caderas, sí en el seguimiento general. Esto es debido principalmente a la poca precisión que ofrece NiTE2 de las mismas. En cualquier caso, la detección de poses se ha implementado de una manera que hace que añadir nuevas articulaciones a la comparación sea una tarea casi trivial.

- La comparación puede no funcionar correctamente en casos de oclusión. Esto es, en casos en los que no exista línea visual directa entre alguna articulación y el sensor. Cuando esto ocurre, NITE2 estima la posición de la

articulación con una escasa fiabilidad, ya que no utiliza información temporal. Si la estimación no es buena, aunque la pose esté correctamente ejecutada, el sistema puede no detectarla.

- Los vídeos producidos con el componente OpenNI2Comp aunque son válidos para su reproducción no lo son para hacer el seguimiento de personas sobre ellos. Provoca un error de programación. Para realizar las pruebas, los videos han sido grabados con la herramienta de OpenNI2, NiViewer. Aunque es un error menor, sería conveniente solucionarlo para tener una funcionalidad completa al 100% de OpenNI2 en RoboComp.

Como puntos fuertes del sistema se encuentran;

- La comparación funciona independientemente del tamaño del usuario y de su orientación frente a la cámara. Esto es importante pues hace bastante general el algoritmo de comparación. No es necesario modificar ningún tipo de parámetro para cada funcionalidad nueva que se añada. Todos los tipos de personas se mueven en la comparación sobre los mismos rangos, por lo que los umbrales definidos son generales para todos.

- Es multi-reconocedor, define las poses de las personas que haya en la escena de forma simultánea. Esto es muy útil, ya que de no ser así restringiría mucho su uso. Además abre una posibilidad muy interesante, que es la de, una vez definida la pose de cada una de las personas de una escena, tiene la posibilidad de analizar lo que ocurre en ella, infiriendo conclusiones que de otra manera no podría. Es decir, por ejemplo, si dos personas se saludan, puede deducir que tienen algún tipo de relación. O si en lugar de saludarse están abrazándose, deduce que la relación es más estrecha, incluso de cariño.

- Es fácilmente ampliable a nuevas poses. La implementación se ha hecho de tal manera que resulte sencillo añadir nuevos modelos sintéticos, basta con una simple asignación por cada nueva pose.

- El código ha sido diseñado para que sea fácilmente reutilizable y extensible, pensando en la importancia de estas características en el mundo del software, y especialmente dentro del framework de RoboComp donde es de vital importancia.

El fin con el que se planteó este proyecto, quizás demasiado ambicioso, era dotar al robot con la capacidad de detectar los *affordances* que existieran en su entorno, obteniendo información de las personas que se encontraran a su alrededor. Se ha logrado detectar las posturas de las personas y con esta, definir de una manera muy básica qué está haciendo. Esta solución puede seguir investigándose, para que conociendo con un cierto grado de fiabilidad qué está haciendo una persona, el robot pueda decidir si debe interactuar con ella o no, y de qué manera.

Otra idea más compleja que requiere además un análisis de las características físicas de los objetos que le rodean sería, conociendo qué está haciendo una persona, analizar cómo es el objeto que usa, si es que tiene alguno, para saber qué acciones puede realizar con objetos de esas características. Ponemos un ejemplo para entender mejor esta idea:

*Si un robot detecta que una persona está sentada sobre un objeto plano de unas medidas determinadas, podría deducir que objetos planos de medidas semejantes pueden servir para sentarse. O si un robot detecta a una persona con un cubo en el que hay objetos dentro, puede deducir que objetos cóncavos pueden valer para transportar otros objetos.*

Como vemos son ideas muy lejanas, pero que requieren de pequeños pasos como el de este proyecto para llegar a conseguirse. En los puntos venideros comentaremos dos ideas que han surgido como posible continuación inmediata a este proyecto y que van enfocadas en el camino de investigación comentado.

En cuanto a las conclusiones que puedo sacar en el plano personal, debo decir que he quedado muy satisfecho con el trabajo realizado, pues comencé el proyecto sin haber escuchado siquiera qué era un CMake o un árbol cinemático..., y acabo entendiendo no sólo eso sino también en qué consiste RoboComp (aunque queda mucho por aprender), cómo funciona InnerModel con sus matrices de transformación, rotaciones, etc.; cómo funcionan las cámaras RGBD, la visión estereoscópica, los sistemas MoCap y conociendo un poquito más de otros muchos temas.

Además este proyecto me ha hecho sentir una especie de admiración por la robótica y su software al conocer sus dificultades y también valorar mucho más, algunos avances que antes no valoraba. Otro punto a favor de la robótica y que me ha encantado es el enfoque psicológico y filosófico que se le da a la manera de diseñar sistemas robóticos, el cual debería estar más presente en la informática general. Todo esto me ha hecho tener hambre de seguir desarrollando y formándome en estos temas, por lo que espero y deseo que este final sea el comienzo de un largo recorrido.

## 7.2. IMITACIÓN DE UN ROBOT A UNA PERSONA

Uno de los retos que surgen a partir del desarrollo que tenemos es intentar saber si el robot que observa a una persona sería capaz de imitar sus posiciones. La pregunta planteada es: “Yo siendo robot, conociendo como soy, observo y analizo a una persona, ¿sería capaz de imitarle?”.

Es una pregunta más compleja de lo que parece, pues el robot no tiene por qué ser antropomórfico, ni tener las mismas articulaciones… A la pregunta ¿una persona puede imitar la postura de un perro? ¿Y a un pulpo? Para nosotros la respuesta es sencilla, pero ¿en qué nos basamos para decidirlo? No nos fijamos sólo en el número de extremidades, como se puede pensar en un principio, porque una persona también puede imitar a un pez, por ejemplo. ¿A la forma del cuerpo? Tampoco es la solución, y no tenemos una respuesta concreta, en este capítulo propondremos una idea básica y propia que entendemos puede ser el comienzo de una solución real.

### 7.2.1. DEFINICIÓN GENÉRICA DE POSTURAS

Hasta ahora, las posturas humanas las hemos definido a partir de un árbol cinemático en el que guardamos las rotaciones y traslaciones de cada articulación con respecto al padre. Esta representación no sirve para saber si un robot con una morfología distinta a la humana podría imitar dicha postura.

Lo primero que necesitamos en este reto, es definir una forma general de postura para todos y cada uno de los seres vivos o máquinas que se puedan representar con un árbol cinemático. Para que posteriormente un robot en este caso, pueda decidir a partir de la definición general de la postura, si sería posible adaptarse a ella y cómo.

Pensando la forma de hacerlo, llegamos a una conclusión con una imagen como la siguiente:

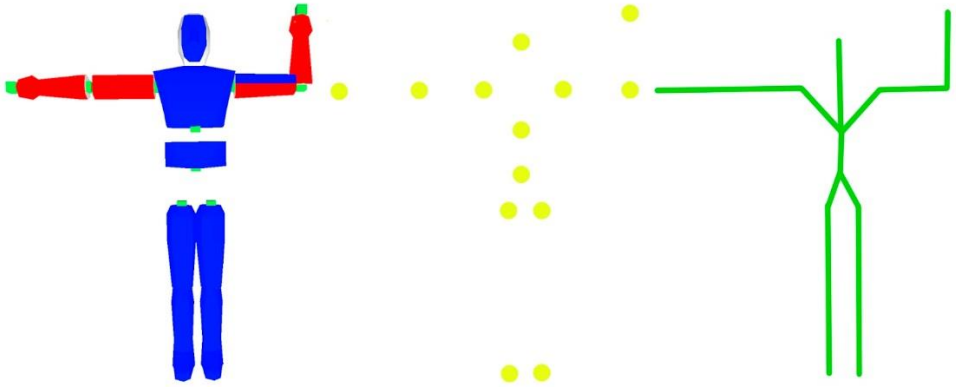

*Figura 121: ¿Qué características determinan una postura?*

La sensación de postura no la da la posición en la que se encuentren las articulaciones, si no el esqueleto que forma. Es más, cuando un dibujante empieza a pintar una persona o cualquier otro ser vivo, comienza pintando el esqueleto del mismo, dibujando sobre él posteriormente los demás detalles, ya que es esto lo que define una postura.

*Figura 122: El dibujo desde cero*

Por lo tanto, intuimos que la forma de simbolizar una postura de forma general es definiendo de alguna manera su esqueleto. Pensamos que una buena forma de hacerlo es a partir de los segmentos y ángulos que lo forman. Nuestra idea es que un robot pueda saber si puede imitar a otro sujeto a partir de una representación de su postura independiente de la morfología de ambos.

Es decir, se pretende que un robot conociendo el árbol cinemático de una persona o animal, sepa extraer la información relevante necesaria para construir otro árbol que defina su postura de una forma general, basándose en segmentos y ángulos. Como hemos dicho, esta definición debe ser independiente de la morfología del árbol cinemático de origen. Posteriormente, el robot mediante un procesado de su árbol cinemático propio y el construido para definir la postura, decidiría si puede imitarla y cómo. El árbol construido debe ser lo más simple posible, y con una jerarquía de nodos que indique el orden en el que surgen estos "segmentos" o extremidades a partir del nodo raíz.

Veamos un ejemplo:

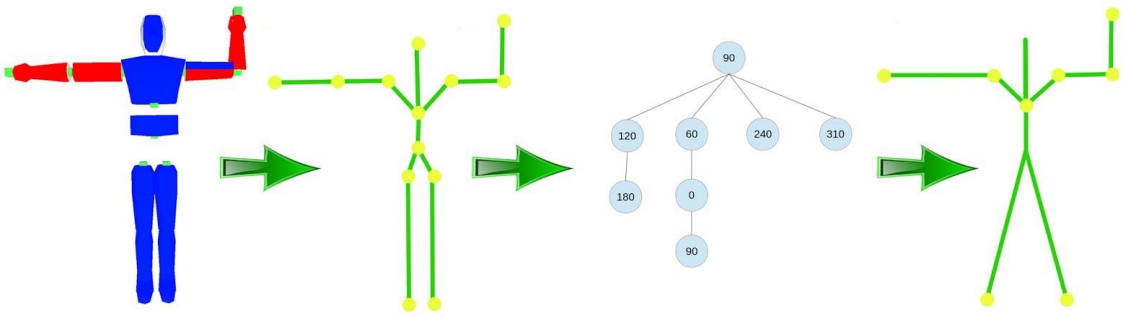


*Figura 123: Pasos para la determinación de una postura general*

En la imagen vemos a una persona con una pose determinada, la representación esqueletal de su árbol cinemático, y el árbol que surgiría a partir de este y llevaría a definir la postura que tiene de una forma general, y no basándose en las articulaciones de la persona. En el ejemplo propuesto, el nodo raíz de este nuevo

árbol es el torso, que como vemos tiene un ángulo de 90º, a partir de él los nodos hijos son segmentos que surgen de él con el ángulo absoluto que marca el contenido del nodo. Los nodos serán colocados de izquierda a derecha según la altura del tronco en la que surgen, de esta manera, por ejemplo los brazos al estar más a la izquierda están por encima de las piernas evitando soluciones erróneas.

El proceso de construcción del árbol de la postura genérica consistiría en recorrer recursivamente el árbol que define la postura de la persona insertando en el árbol de la postura genérica los segmentos que tienen una rotación respecto al padre superior a un umbral a definir, y obviando los que no tienen una rotación destacable, pues no formaría ningún segmento nuevo.

Insertar los nodos de forma ordenada sería posible, pues se conocen las posiciones de las articulaciones, por lo que se puede calcular qué segmentos están por encima de otros, o en el caso de que el torso se encuentre en horizontal, cuáles están a la izquierda o a la derecha.

Los pasos para la construcción del árbol de la imagen anterior son:

1) Creación del árbol, con el nodo raíz de 90º.

2) Obtención de los nodos hijos y procesamiento de arriba a abajo (esta ordenación se haría para cada nodo nuevo con sus hijos, se omite de aquí en adelante por simpleza)

3) Cuello, ángulo de 90º = 90º => Se ignora el segmento

4) Cabeza, ángulo de 90º = 90º => Se ignora el segmento

5) Hombro izquierdo, ángulo de 120º != 90º => Inserción en el árbol de postura genérica

6) Codo izquierdo, ángulo de 180º != 120º => Inserción en el árbol de postura genérica

7) Mano izquierda, ángulo de 180º = 180º => Se ignora el segmento

8) Hombro derecho, ángulo de 70º != 90º => Inserción en el árbol de postura genérica

9) Codo derecho, ángulo de 0º != 70º => Inserción en el árbol de postura genérica

10) Mano izquierda, ángulo de 90º != 0º => Inserción en el árbol de postura genérica

11) Cadera rodilla, ángulo de 240º != 90 => Inserción en el árbol de postura genérica

12) Cadera izquierda, ángulo de 240º = 240º => Se ignora el segmento

13) Pie izquierdo, ángulo de 240º = 240º => Se ignora el segmento

14) Cadera derecha, ángulo de 310º!= 90 => Inserción en el árbol de postura genérica

15) Cadera derecha, ángulo de 310º= 310º=> Se ignora el segmento

16) Pie derecho, ángulo de 310º= 310º=> Se ignora el segmento

Una vez definida la postura genérica con el árbol generado el siguiente paso será, dado un árbol cinemático cualquiera, decidir si es posible que adopte dicha postura.

### 7.2.2. ADAPTACIÓN DE UN ÁRBOL 'A' EN OTRO 'B'

Para comprobar si un árbol cinemático puede adaptarse en otro pasaremos como enlace entre ambos por la postura genérica que hemos definido en el punto anterior.

Una vez tenemos la postura genérica queda comprobar si el árbol que forma el robot puede adoptarla. Para ello, se realizaría un recorrido del árbol, realizando un algoritmo de backtracking, buscando que las articulaciones que el robot tiene puedan adaptarse a los ángulos que se definen en el árbol de la postura genérica, de esta forma se iría definiendo qué articulaciones tienen que adoptar qué ángulos, para formar los segmentos definidos en la postura genérica.

En el momento en el que en una extremidad no haya más segmentos que definir, pero el árbol del robot si tenga más articulaciones, los sucesivos segmentos se alinearán con el segmento de la última articulación definida.

- Con esto tenemos que un árbol A (que describe a un robot) con menos articulaciones que otro árbol B podría imitarlo siempre que éste no haga uso de más articulaciones de las que posea A. Y siempre y cuando las articulaciones de A puedan adoptar los ángulos que adopta B.
- También solucionaríamos el problema que pudiera surgir utilizando este método, de que se quisiera imitar una articulación con otra de distinto tamaño, pues no deberían importar sus tamaños. Una extremidad de 40 cm puede adoptar la forma de otra de 1.5m, y viceversa.

Como vemos esta definición recrea de forma muy general una postura. Esto no aseguramos que se hayan definido todos los casos y el método funcione siempre. Es posible que durante su implementación surjan problemas, pero puede ser un buen inicio para, tras una evolución lograr que un árbol cinemático pueda adaptarse a otro. Dicho de otra manera, lograr que un robot cualquiera pueda saber si puede imitar y cómo a una persona o a otro robot de igual o distinta morfología. Sería un interesante inicio para mi futuro trabajo fin de máster.

## 7.3. GESTOS COMO SECUENCIA DE POSTURAS

La segunda idea por la que puede seguir evolucionando el proyecto consiste en lograr detectar gestos de una forma distinta a la que se hace generalmente. El segundo flanco abierto por donde me hubiera gustado seguir avanzando, consiste en lograr detectar gestos de una forma distinta a la que se hace generalmente.

Actualmente para detectar un gesto se trabaja con las posiciones de las articulaciones que interesan, integrándolas en el tiempo para detectar si coincide con el patrón sintético del gesto.

Nosotros proponemos una vuelta de tuerca, y pretendemos basarnos en una idea clásica para ofrecer algo nuevo.

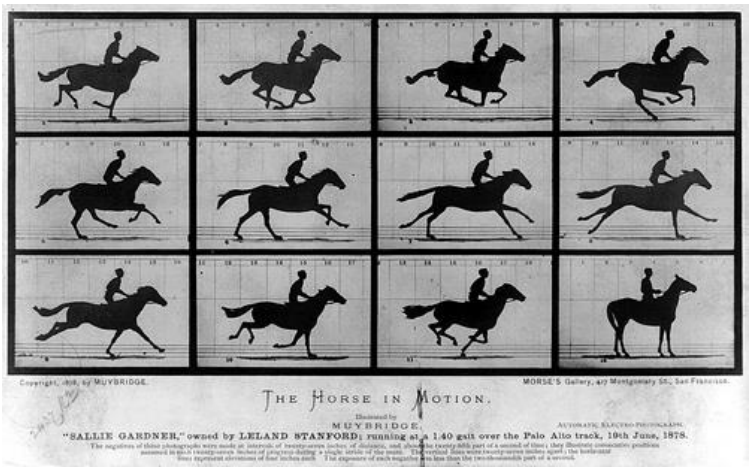


*Figura 124: The horse in motion*

Desde el inicio del cine hasta nuestros días, el movimiento de una imagen se ha generado a partir de un conjunto de imágenes que varían ligeramente de una a otra su aspecto, y que se intercambiaban secuencialmente dando la impresión de movimiento. Una idea clásica, sencilla, y que da muy buenos resultados en este ámbito.

De la misma forma, nosotros planteamos definir un gesto como una secuencia de poses discretas, de esta manera, detectando las poses en un orden determinado podríamos concluir a qué gesto pertenece. Veámoslo con algún ejemplo:

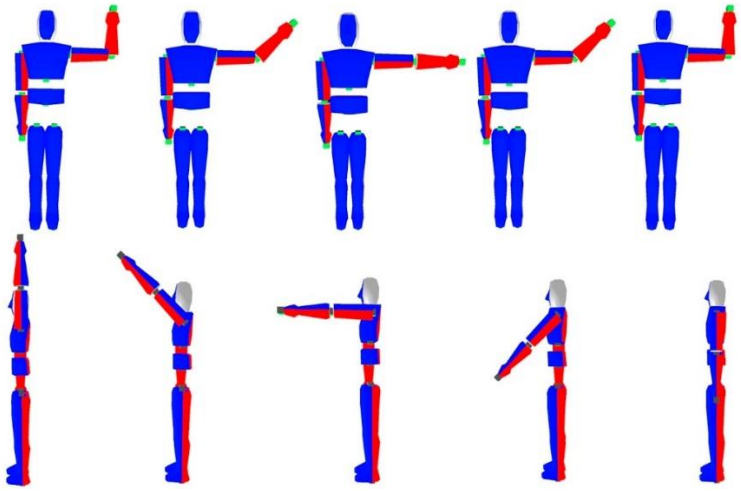

*Figura 125: Gestos como secuencia de posturas*

En la imagen podemos ver dos gestos definidos a partir de secuencia de posturas. Son sus "fotogramas" si pensamos en el ejemplo inspirador.

La primera secuencia podría ser un gesto de saludo o despedida, y la segunda un gesto que simboliza cualquier significado que deseemos añadir. Con la implementación de este sistema que detecta gestos como una secuencia de poses sería muy fácil definir gestos propios, ya que sólo deberíamos crear las posturas intermedias y añadirlas al sistema, encargándose éste de buscar la secuencia introducida.

Para implementar este sistema se podría comenzar, como ya hemos dicho, a partir de este proyecto. Es decir, podríamos añadir nuevas funciones a los métodos implementados o simplemente que los utilizaran y combinaran para obtener nuevos resultados. Hecha la detección de poses, el siguiente paso a seguir sería definir una secuencia de posturas (de árboles InnerModel) con significado de gesto e implementar un algoritmo que descubra de qué secuencia se trata. Es una idea

sencilla de entender, pero lejos de lo que pudiera parecer, nada fácil de implementar. Para la detección de gestos entra en juego el tiempo, ¿cuándo empieza y termina un gesto? ¿Cuánto debe durar para darlo como válido? ¿Cada cuánto se debería pasar de un "fotograma" de la secuencia a otro? Además es posible que detectara posturas intermedias no deseadas, aunque fueran momentáneas, que deberían ser descartadas de alguna manera. Todos estos son problemas que surgirían y que, por supuesto, habría que buscar soluciones que no parecen nada sencillas. Pero creemos que trabajar sobre ello podría dar sus frutos. Y una vez implementado este sistema se nos facilitaría mucho las tareas en detecciones gestuales, pues pasarían de ser un problema complejo a un problema de definición de árboles InnerModel.

Como hemos visto en estas páginas, el proyecto ofrece una funcionalidad sencilla de entender como es la detección de poses humanas, pero a la vez tan amplia y versátil y que expande el abanico de posibilidades hasta donde nuestra mente sea capaz. En cualquier caso, no debemos olvidar las raíces de este proyecto, las cuáles marcan el camino principal por el que nos gustaría que se desarrollase, que no es otro que el de la robótica ecológica. Se ha pretendido asentar una base sobre la que trabajar para dotar al robot de una especie de "conciencia" que le permita sacar conclusiones sobre lo que hay a su alrededor. Y con esta información concederle un pequeño "poder de decisión" para elegir qué hacer en cada momento, buscando siempre la forma más natural de interactuar con el medio.

Creemos firmemente que es el camino por el que debe avanzar la robótica, pues hemos enseñado a los robots a jugar al ajedrez antes que a caminar. Vemos necesario este cambio de mentalidad para una mayor integración de ésta en nuestra forma de vida, y esperamos que este proyecto haya servido para dar un pequeño paso en este largo camino.

# 8. BIBLIOGRAFÍA

[1] Luis J. Manso. Navegación visual en robots móviles. Proyecto fin de carrera. 2009.

[2] Luis Joyanes., Fundamentos de programación. Algoritmos, estructuras de datos y objetos. 3ª edición 2003.

[3] Michi Henning and Mark Spruiell. Distributed Programming with Ice., revision 3.3.0. 2008.

[4] Robolab. RoboComp project - http://robolab.unex.es/index.php?option=com_content&view=article&id=10&Itemid=26. 2014.

[5] Web. RoboComp project. - http://robocomp.sourceforge.net/wordpress/. 2014.

[6] Wiki. RoboComp project - http://wiki.robocomp.org. 2014.

[7] Code. RoboComp project - http://sourceforge.net/projects/robocomp/. 2014.

[8] Juan P. Bandera. Vision-based gesture recognition in a robot learning by imitation framework. Tesis doctoral. 2010

[9] Web oficial OpenNI2 - http://structure.io/openni. 2014.

[10] Downlod NITE2 - https://www.dropbox.com/s/fds3icp17d8602d/NiTE-Linux-x64-2.2.tar1.zip. 2014.

[11] Regla de la mano izquierda - http://es.wikipedia.org/wiki/Regla_de_la_mano_izquierda. 2014

[12] Regla de la mano derecha - http://es.wikipedia.org/wiki/Regla_de_la_mano_derecha. 2014.

[13] Definición de cuaterniones - http://es.wikipedia.org/wiki/Cuaterni%C3%B3n

[14] Ivan Dryanovski, William Morris, Ravi Kaushik and Jizhong Xiao. Real-Time Pose Estimation with RGB-D Camera

[15] Prof. Dongheui Lee, Ph.D.Robust Hand Pose Estimation with a RGBD Camera Using Large Database

[16] Renato Contini. Engineering in medicine.

[17] Distancia de Manhattan - http://es.wikipedia.org/wiki/Geometr%C3%ADa_del_taxista. 2014.

[18] Distancia euclídea - http://es.wikipedia.org/wiki/Distancia_euclidiana. 2014.

[19] Teorema de Pitágoras - http://es.wikipedia.org/wiki/Teorema_de_Pit%C3%A1goras. 2014.

[20] Manuel Heras Escribano. Comprender la realidad sin representaciones. Affordances y psicología ecológica.

[21] James J. Gibson. The Ecological Approach to visual perception. 1986.

[22] María Muñoz Serrano y Alex Diaz. Inteligencia Artificial ecológica para una nueva generación de robots. Revista Factotum. 2010.

[23] Thomas E. Horton, Arpan Chakraborty, and Robert St. Amant. Affordances for robots: a brief survey

# 9. ÍNDICE DE FIGURAS