Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Universidad de Valladolid

Escuela de Ingenierías Industriales

Grado en Ingeniería Electrónica Industrial y Automática

Interacción con un robot colaborativo mediante un interfaz háptico

Autor:
Rodríguez Acebrón, Jorge

Tutor:
Pérez Turiel, Javier
Ingeniería de Sistemas y Automática

Valladolid, Julio 2021

Escuela de Ingenierías Industriales

Universidad de Valladolid

Grado en Ingeniería Electrónica Industrial y Automática

Interacción con un robot colaborativo mediante un interfaz háptico

Autor:
Jorge Rodríguez Acebrón

Agradecimientos

Con este trabajo de fin de grado pongo punto final a esta etapa de mi vida en la Universidad de Valladolid. Han sido unos años de intenso aprendizaje tanto en lo personal como en lo académico, que han pasado mucho más rápido de lo que hubiese querido.

Como siempre, ha habido buenos y malos momentos, y es aquí donde quiero acordarme de todos aquellos que han estado conmigo tanto en los primeros como especialmente en los segundos.

Por ello quiero comenzar agradeciendo a mi tutor Javier Pérez Turiel y a los trabajadores del ITAP por la ayuda y medios que me han prestado para la realización de este TFG, en particular Ana Cisnal de la Rica, Carlos Fonturbiel Mediavilla y David Sierra Rodríguez por su colaboración a lo largo de la elaboración del trabajo.

Quiero dar también las gracias a todos los amigos y compañeros que me han acompañado a lo largo de esta etapa y que en tantas ocasiones me han servido de soporte para seguir adelante.

Por último, quiero hacer una mención especial a mi familia, por todo el tiempo empleado para que esto haya sido posible, por haberme sabido aconsejar cuando lo he necesitado y sobre todo por su apoyo en los momentos en los que los estudios se pusieron cuesta arriba o en los que la salud no acompañó.

Gracias a todos.

Resumen

El objetivo de este trabajo de fin de grado consiste en determinar de qué modo se pueden conjugar las capacidades hápticas, dinámicas y cinemáticas de un robot colaborativo *Universal Robots UR3,* con las de un sistema háptico *3D Systems Touch.*

Siendo más preciso, se propone el desarrollo de un software que nos permita, mediante la interacción de un usuario con el mencionado dispositivo háptico, la realización de un control manual de la posición y orientación del *TCP* del robot previamente mencionado.

Se realizará un análisis del funcionamiento y capacidades tanto del sistema háptico como del robot colaborativo, haciendo mayor hincapié en el primero ya que no existe experiencia previa de su uso en la Universidad de Valladolid. De esta manera, se podrán delimitar tanto las funciones que pueden realizar de manera conjunta, como los medios a través de los cuales efectuar esa conexión.

Posteriormente se efectuará una verificación de la operatividad conjunta de los sistemas (mediante simulaciones primero y experiencia real una vez que consiga un funcionamiento satisfactorio en el entorno virtual), para finalizar valorando las posibles vías futuras de estudio.

Palabras clave

Dispositivo háptico, cobot, interacción, control, *HIP*.

Abstract

The objective of this end-of-studies project is to find a way to combine the haptic, kinematic and dynamic capabilities of a *Universal Robots UR3* collaborative robot with those of a *3D Systems Touch* haptic device*.*

Taking a deeper look, it is intended to develop a software which allows a user to manually determine, through the use of the previously mentioned haptic device, the position and orientation of the tool center point of an UR3 robot.

First of all, an analysis of the functioning and capabilities of the haptic interface and collaborative robot will be conducted. Taking into account the haptic interface has not been studied previously in the Universidad de Valladolid, special attention will be paid in this task. Once this objective has been accomplished, a steppingstone will be possessed in order to begin with the works of the combined operation.

Finally, an examination process will be required (using simulations, and real-life operation) in order to determine whether the system is working properly, to be followed by a search of future study pathways.

Keywords

Haptic device, cobot, interaction, control, HIP.

Índice

Índice de figuras

1.- INTRODUCCIÓN Y OBJETIVOS

1.1.- Introducción.

El trabajo de fin de grado que aquí se presenta, se realiza como la última etapa del plan de estudios del Grado en Ingeniería Electrónica Industrial y Automática impartido por la Universidad de Valladolid. Se han utilizado para su realización medios proporcionados por el Instituto de las Tecnologías Avanzadas de la Producción, institución perteneciente a la UVa.

El proyecto que se plantea tiene como finalidad el desarrollo de un mecanismo de conexión y control, el cual permita a un usuario establecer manualmente la posición y orientación de un robot colaborativo *UR3* desarrollado por la compañía *Universal Robots*, mediante el uso del dispositivo háptico *3D Systems Touch*, producido por la empresa *3D Systems*.

La utilización de los robots colaborativos ha aumentado notablemente en los últimos años, y se prevé que esta tendencia no se frene en un futuro próximo. En concreto según se indica en [1], los ingresos anuales de las empresas del sector crecerán de los 22,2 mil millones de dólares en 2020 a los 38,3 mil millones en el año 2024. A pesar de que los valores de 2020 puedan haber sido adulterados por los efectos de la pandemia que aún seguimos sufriendo, la tendencia alcista es clara.

Estos sistemas han permitido abandonar el paradigma del robot enjaulado, que debe permanecer en condiciones completamente controladas y aisladas del exterior tanto para garantizar su correcto funcionamiento, como para evitar riesgos laborales. Ha sido sustituido por otro en el que estos sistemas automatizados son capaces de relacionarse con el medio que les rodea y responder a estímulos externos, haciendo desaparecer barreras en lo concerniente a la seguridad y permitiendo su implementación en ámbitos que hasta su aparición quedaban vetados a la robótica debido a la naturaleza cambiante del entorno.

Figura 1. Robot colaborativo operando junto a trabajador humano [2].

Por otra parte, la aparición (que como veremos posteriormente está muy cercanamente ligada a la de los robots colaborativos) y posterior desarrollo de los interfaces hápticos, supone un importante avance tanto en el diseño de mecanismos de simulación como de teleoperación.

Los mencionados mecanismos generalmente constan de un medio visual acompañado por otro auditivo, que, mediante el uso de un elemento de control, véase mando, *joystick*, volante, etc… permiten operar dentro de un entorno determinado. La implementación de un interfaz háptico supone la transformación del elemento de control en una nueva herramienta, a través de la cual el usuario recibe información en forma táctil. Este nuevo elemento otorga al usuario una mayor inmersión, más aún si se opta por la introducción de gafas de realidad virtual en lugar de pantalla y altavoces, lo cual juega en su favor a la hora de la toma de decisiones necesarias para su actividad.

Figura 2. Ejemplo de setup para simulación con uso de interfaz háptico y realidad virtual [3].

La labor que se presenta en este trabajo de fin de grado, es el desarrollo de una herramienta *software* que permita comunicar ambos sistemas a fin de poder combinar las características que sean más valoradas de cada uno. Para lograrlo, se deberá primero realizar una tarea de investigación acerca del funcionamiento de los dispositivos hápticos, en concreto del modelo *3D Systems Touch*, ya que no han existido hasta la fecha labores de investigación al respecto en la Universidad de Valladolid.

Una vez se cuente con ese conocimiento, se implementará en *ROS* (*Robot Operating System*) un mecanismo de control y comunicación entre interfaz háptico y robot colaborativo, utilizándose la herramienta "*MoveIt*" para la implementación y verificación de la solución alcanzada, primero en un medio simulado y posteriormente en un entorno real.

1.2.- Objetivos.

Tras esta presentación, paso a concretar los objetivos fijados para la realización del presente TFG.

- Búsqueda y análisis de información concerniente al funcionamiento, características, limitaciones y usos tanto habituales como experimentales de los interfaces hápticos, robots colaborativos y *Robot Operating System*, a fin de establecer el estado del arte.

- Puesta a punto de un dispositivo de interfaz háptico *3D Systems Touch* y estudio del entorno de desarrollo *OpenHaptics Toolkit* proporcionado con el dispositivo.

- Desarrollo e implantación de los canales de comunicación entre el interfaz háptico y robot colaborativo.

- Establecimiento de un método de control adecuado a las posibilidades y limitaciones de ambos sistemas.

- Validación de la solución alcanzada.

2.- Estado del arte

En este capítulo se llevará a cabo una descripción general de las tecnologías que he utilizado para la realización del trabajo de fin de grado, así como una explicación de los conceptos y principios en los que se basa su funcionamiento, aportando ejemplos de su aplicación en la industria o en entornos experimentales.

2.1.- Dispositivos hápticos.

Un dispositivo háptico se define como un sistema que interacciona con un usuario humano por medio del sentido del tacto. Si bien se incluyen dentro de este concepto el estudio de las percepciones tanto de presión como temperatura y dolor, los dispositivos hápticos (en su gran mayoría) solo pueden generar y recibir información a través de la interacción mecánica.

Existen múltiples clases de dispositivos hápticos debido al amplio rango de opciones que ofrece el sentido del tacto. Para entender un poco mejor este abanico de opciones y concretar en aquellas que son de nuestro interés, comenzaré por una breve explicación de dicho sentido.

2.1.1.- Sentido del tacto.

"Los sentidos son un conjunto de procesos y mecanismos fisiológicos que nos permiten captar estímulos del exterior" [4] siendo el más distribuido en el cuerpo humano el sentido del tacto. El órgano encargado de su percepción es la piel, a través de la cual somos capaces de recibir información que agrupamos en las tres clases que se nombraban anteriormente: temperatura, presión y dolor.

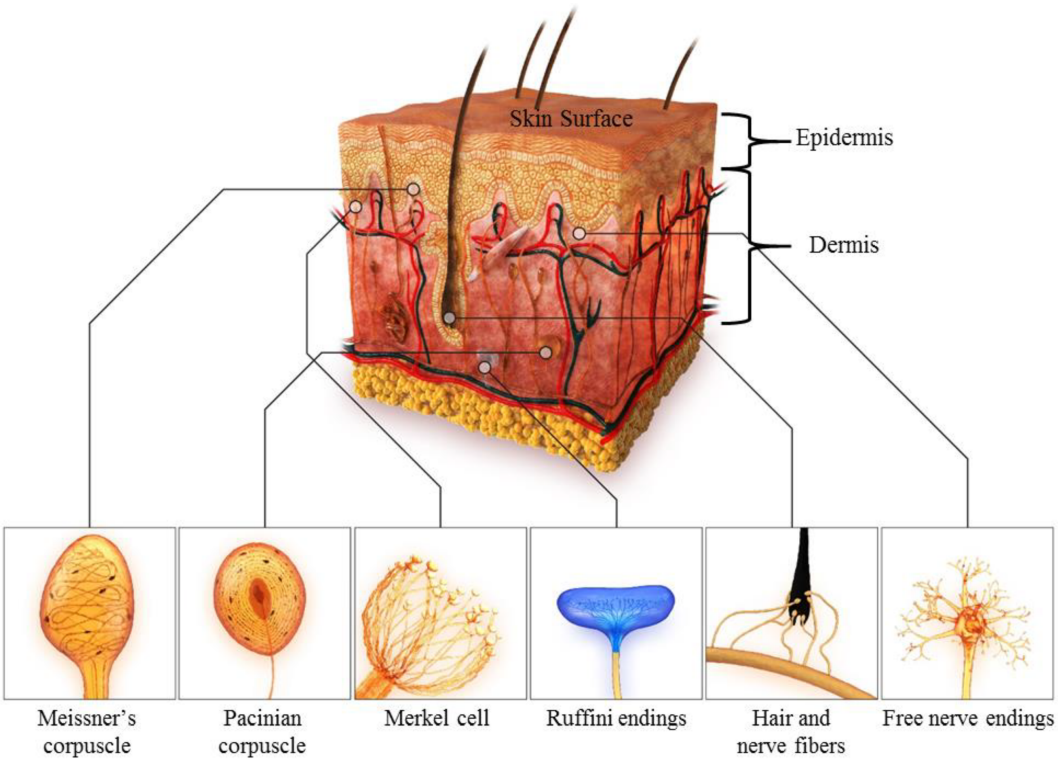

Figura 3. Descripción de la piel y receptores táctiles [5].

La piel consta de tres capas, siendo la más externa la epidermis. Cuenta con un espesor cercano a los 0,1 mm y está conformada por células epiteliales muertas conocidas como queratinocitos, los cuales se encuentran en constante proceso de renovación. Sirve como barrera ante patógenos o sustancias externas que puedan ser perjudiciales para el individuo, a la vez que limita la pérdida de fluidos.

En el extremo interno de la piel encontramos la hipodermis, formada en su vasta mayoría por células adiposas, las cuales contienen un 95% de lípidos, y que reciben el nombre de adipocitos. Entre sus funciones se encuentra el mantenimiento de la temperatura corporal, sirviendo además como reservorio de energía y sistema de amortiguación ante impactos.

La capa intermedia se conoce como dermis, la más gruesa de las tres, siendo aquí donde se encuentran los mecanismos receptores del sentido del tacto. Estos mecanismos son agrupaciones de células nerviosas especializadas que aportan diferentes clases de información, por ejemplo:

- Corpúsculo de Meissner: percibe contactos ligeros, textura de los objetos y deslizamiento sobre la piel.
- Corpúsculo de Pacini: responden ante vibraciones rápidas y presión mecánica profunda.
- Discos de Merkel: son receptores capaces de percibir fuerzas estáticas con gran precisión, permitiendo detectar las formas de los objetos que entran en contacto con la piel.
- Corpúsculos de Ruffini: perciben la variación de temperatura además de deformaciones en la piel (estiramientos).

Existen además una serie de receptores situados en las articulaciones y en sus inmediaciones, que aportan información acerca del estado (fuerza ejercida, posición o velocidad) de cada una de ellas. Reciben el nombre propioceptores.

La información aportada por todos ellos se transmitirá en forma de impulsos nerviosos, de tal manera que podamos generar respuestas adecuadas ante los estímulos exteriores con los que entremos en contacto.

2.1.2.- Interfaz háptico.

Un dispositivo háptico hace referencia a cualquier sistema electrónico o mecánico el cual, mediante el uso del tacto se comunica con un operario. Si el sistema cuenta con medios que permitan al usuario obtener información a través del sentido del tacto, pero no posee mecanismos sensitivos que otorguen la capacidad de modificar el estímulo háptico producido, estaríamos hablando de un "*display* háptico". Por el contrario, si el dispositivo cuenta con mecanismos sensitivos, pero el usuario no recibe ninguna reacción háptica por parte del sistema, en función de la información recibida, estaríamos hablando de un "sistema de input general".

Un ejemplo de *display* háptico podría ser la vibración del *joystick* de control en un avión al encontrarse éste en peligro de entrada en pérdida.

Si la velocidad es demasiado baja o el ángulo de ataque (ángulo que forman las alas con respecto a la dirección del aire incidente) es demasiado grande, podría alcanzarse un estado en el cual la sustentación del avión no fuese la suficiente como para mantener la aeronave bajo una situación de vuelo controlado.

El piloto puede llevar a cabo una serie de acciones para poner fin a esa situación, por ejemplo, aumentar la velocidad o disminuir el ángulo de ataque, pero no son los comandos del piloto los que ponen fin al efecto, sino el estado de la nave al cual conducen estas acciones.

Por otro lado, las teclas de un piano conformarían un sistema de input general. Dependiendo de la fuerza, velocidad o firmeza con la que se pulsen se obtiene una sonoridad determinada, sin embargo, en ningún caso la resistencia ofrecida se modificará en función de la clase de contacto o el número de teclas pulsadas.

Si un dispositivo es capaz de al mismo tiempo, percibir una serie de estímulos táctiles, interpretarlos y generar una respuesta acorde mediante la ejecución de una fuerza o la modificación de la posición de algún elemento del dispositivo, decimos que se trata de un interfaz háptico.

En esta categoría se incluyen por ejemplo los sistemas *Phantom* desarrollados por la compañía *3D Systems*, ya que el usuario puede determinar la posición y orientación del sistema en función de las fuerzas que ejerza sobre él, al mismo tiempo que recibirá información háptica dependiendo del punto del escenario virtual o teleoperado en el que se sitúe.

Figura 4. Dispositivos hápticos Phantom Premium. [6]

Desde un punto de vista físico, los dispositivos hápticos pueden agruparse en dos clases en función de las características que presenta la recepción y aportación de información por parte del usuario: controlados por impedancia o por admitancia.

En el primer caso, la aportación de información por parte del usuario se realizará mediante desplazamiento (posición y/o velocidad), y la recepción por medio de fuerzas, de tal manera que las propias fuerzas generadas por el dispositivo interfieran con las producidas por el usuario.

De manera general se utiliza un único punto de interacción que se conoce como "punto de interfaz háptico" o *HIP* por sus siglas en inglés. Dado que la concentración en un único punto del sentido del tacto resulta contraintuitivo, los interfaces hápticos acostumbran a presentar algún tipo de mecanismo que mejore la ergonomía. Pueden presentarse soluciones que se asemejen a bolígrafos, como es el caso de los dispositivos desarrollados por la compañía *3D Systems*, o a utensilios específicos como tijeras en el caso de que se quiera utilizar el interfaz con un fin determinado y preestablecido, situándose el *HIP* en la zona de trabajo habitual del objeto o utensilio que represente.

En la mayor parte de las aplicaciones son utilizados en conjunto con mecanismos de audio y vídeo, de tal manera que pueda complementarse la experiencia envolvente que proporcionan estos dispositivos.

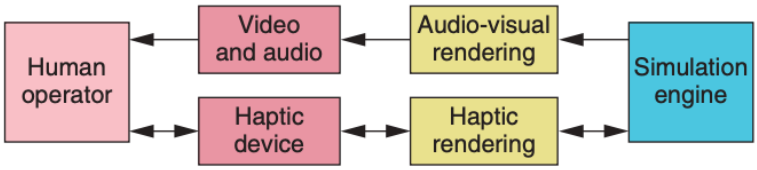

Figura 5. Arquitectura genérica de la aplicación de un dispositivo háptico controlado por impedancia [7].

En cuanto a los sistemas controlados por admitancia, la información aportada por el usuario se realiza en forma de fuerzas o torques, mientras que el dispositivo aporta una respuesta en forma de posición, velocidad o aceleración. Un lector en lenguaje braille es un ejemplo de estos sistemas: mediante el uso de botones la persona puede desplazarse a lo largo de un texto, modificándose la posición de los puntos en cada uno de los caracteres que sea necesario.

Figura 6. Instrumento electrónico de lectura en lenguaje Braille. [8]

2.1.3.- Modelos de contacto.

En el proceso de diseño de un modelo, se deberá atender a las características y necesidades prefijadas a las que se busque atender. Mientras que en algunas aplicaciones, la capacidad del dispositivo para renderizar contornos de escenarios o piezas será suficiente, para otras, la posibilidad de generar rugosidades o superficies elásticas, por ejemplo en una simulación dirigida a la formación en medicina, puede mejorar la experiencia formativa del usuario.

En el desarrollo de estos modelos, los programadores se enfrentan a una serie de desafíos. En primer lugar, los interfaces hápticos cuentan con un área de trabajo nominal, en la que el fabricante garantiza un funcionamiento equitativo y preciso en todas direcciones, y un área de trabajo real, determinada por las limitaciones físicas del dispositivo y donde pueden producirse comportamientos indeseados. Deberá valorarse la conveniencia de realizar un programa capaz de presentar un funcionamiento correcto en la totalidad del área de trabajo real, o buscar una forma de restringir el movimiento del interfaz a la zona de trabajo nominal, a fin de evitar comportamientos inaceptables.

Figura 7. Zona de trabajo nominal (izq) y real (dcha) de un dispositivo Phantom Omni [9].

En segundo lugar, un dispositivo háptico ideal no debería generar rozamientos al desplazarse a través de un entorno carente de obstáculos, sin embargo, es inevitable que existan rozamientos internos, por lo que se podría utilizar el modelo para contrarrestarlos.

Por último, los procesos necesarios para la renderización háptica suponen un uso importante de recursos por parte de los sistemas informáticos, en consecuencia, en la fase de diseño se deberá plantear un compromiso entre la complejidad del modelo de contactos y los recursos necesarios para su funcionamiento. Un modelo óptimo debería contar con renderización háptica en seis grados de libertad (fuerzas y momentos en las tres direcciones cartesianas), ya que este es el modelo observable en la vida real, pero ya sea con el objetivo de reducir la carga computacional o el de lograr un modelo simplemente más sencillo, en ocasiones se utilizan únicamente tres grados de libertad, eliminando los momentos torsores y reduciendo a un único punto la interacción usuario-entorno virtual, al anteriormente mencionado HIP.

Una de las filosofías que utilizan tres grados de libertad, es el llamado método *proxy*. Consiste en la generación de un sistema formado por dos puntos: uno que en todo momento representará la posición determinada por el interfaz háptico, y otro, llamado punto *proxy*, el cual estará sujeto a las restricciones de movimiento que se presenten dentro de las simulaciones.

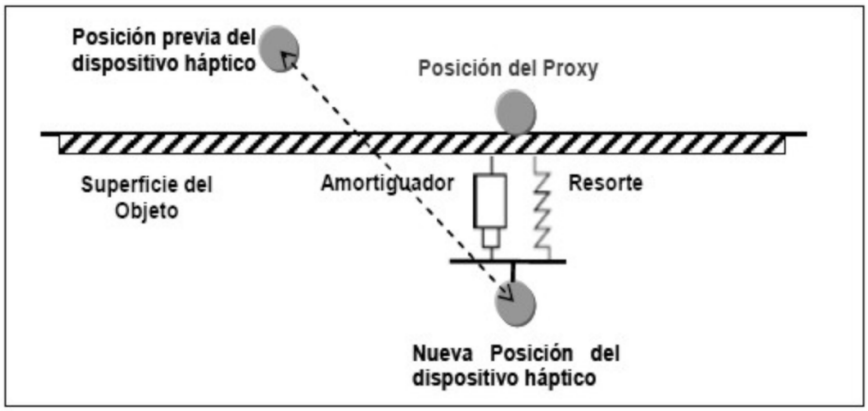

Figura 8. Esquema de modelado mediante método proxy [10].

Cuando no existe contacto en la simulación, estos puntos son coincidentes, sin embargo, al producirse una colisión, el punto de interfaz háptico continúa representando la posición real del sistema físico, mientras que la posición proxy representa la posición del entorno virtual, sin poder ésta atravesar ningún elemento sólido de la simulación. Cuando se produzca una disparidad en la posición de ambos, se crea entre ellos un sistema muelle-amortiguador, que producirá una fuerza mayor a medida que la distancia entre ellos aumente, en dirección y sentido normales a la superficie de contacto, oponiéndose así a la intención del usuario de atravesar la restricción virtual.

Las fuerzas generadas responden a la siguiente ecuación en cada una de las tres dimensiones:

$$F\vec{\imath} = (-K \cdot \Delta x)\vec{\imath} + \left(-b \cdot \frac{\partial \Delta x}{\partial t}\right)\vec{\imath}$$

En donde *F* es la fuerza en una dirección genérica $\vec{\imath}$ (X, Y o Z), *K* la constante elástica del muelle, Δx la distancia en dirección $\vec{\imath}$ entre el punto de interfaz háptico y el punto *proxy*, tomando *b* como la constante viscosa del amortiguador.

En condiciones normales, la distancia entre ambos puntos será aquella que resulte en la ejecución de una menor fuerza, sin embargo, si se produce un cambio de posición, y el punto proxy ha de transitar por una región con mayor energía potencial en comparación con la que posee en ese momento debido a la existencia de un obstáculo, no podrá continuar avanzando en esa dirección.

Por otra parte, si lo que se busca es un modelo con seis grados de libertad, no se puede reducir la interacción a un único punto, ya que no existe forma de aplicar un momento torsor a un elemento unidimensional, es aquí donde entra en juego el concepto del "*god object*". Este *god object* hace referencia a un objeto con el cual el usuario se relaciona con el medio, no pudiendo atravesar el contorno de ese objeto las restricciones de movimiento producidas por otros objetos o cualquier otra clase de limitación.

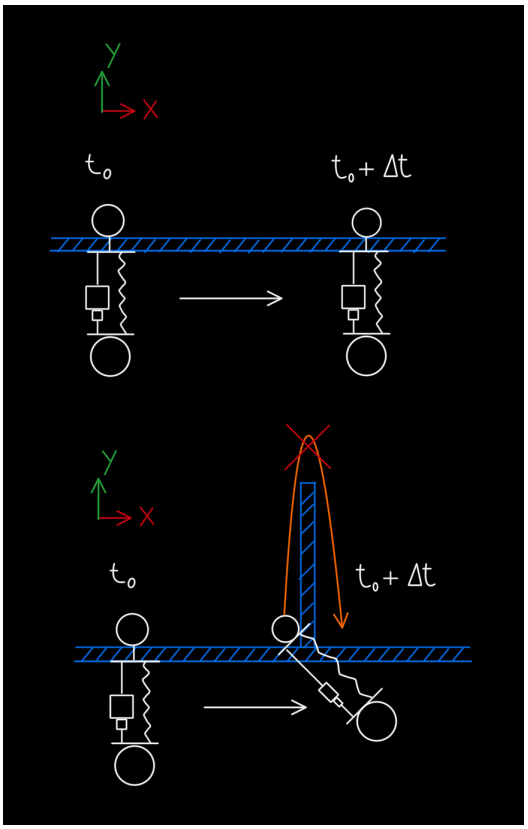

Figura 9. Comportamiento del modelo proxy en presencia de un obstáculo físico.

Se puede encontrar en [11] una explicación pormenorizada del funcionamiento de un modelo diseñado a partir de este concepto.

2.1.4.- Utilización de los interfaces hápticos.

Dentro de los usos de los interfaces hápticos, podemos encontrar el diseño asistido por ordenador. En un primer momento, tal y como se indica en [12], los interfaces solo se utilizaban como medio de comprobación de los diseños generados por otros programas de *CAD*, ampliándose progresivamente su uso al diseño en tres dimensiones, a medida que la tecnología necesaria para su uso se iba desarrollando o abaratando, de tal manera que el desembolso fuera económicamente rentable para las empresas.

Uno de los problemas que presentaba la implementación en conjunto de los programas de *CAD* y renderización háptica, se debía a que los modelos de representación en los que se basaban los programas de *CAD* eran distintos de los utilizados en programas dirigidos a dispositivos hápticos. En el primer caso utilizaban modelos de representación de fronteras o *B-Rep*, mientras que en el segundo se servían de modelos de representación geométrica simplificada como la geometría poligonal, *volxel* o *NURBS*. Otro de los aspectos a tener en cuenta, es la gran carga computacional que requieren los interfaces hápticos, sumada a la para nada desdeñable carga que suponen los programas de *CAD*, que hubiesen obligado a la compra de elementos informáticos de mayores prestaciones capaces de ejecutar ambas tareas simultáneamente.

La mejora de los sistemas informáticos y el descenso de su coste, asociado al desarrollo de *software* dedicado como *Geomagic Sculpt* o *Geomagic Freeform*, ha permitido una mayor implantación de estos sistemas en entornos tanto profesionales como de investigación.

El método de uso de estos dispositivos en el campo del diseño consiste en tomar un bloque de material y moldear sus superficies mediante la interacción de un punto de interfaz háptico, presionando o arrastrando las superficies preexistentes. La principal ventaja que presentan es que permiten la generación de formas muy complejas de manera más rápida que otros programas de *CAD*, pudiendo actuar en todo momento sobre las tres dimensiones del objeto.

En contrapartida, suponen un mayor desembolso inicial en los equipos requeridos para su utilización, partiendo de un coste de 600 dólares en los modelos de interfaz háptico más básicos y de más de 2000 en modelos más avanzados, superando con creces esa cifra modelos capaces de realizar una renderización háptica con seis grados de libertad.

Figura 10. Ejemplo de imagen diseñada con interfaz háptico [13].

Dentro del campo de la medicina existen numerosos ejemplos del uso de interfaces hápticos como medio de formación. Uno de ellos es el desarrollo de un entorno simulado en el que ensayar la extirpación de la glándula submandibular [14].

Se utiliza un interfaz háptico *Geomagic Touch*, al que se le realiza una modificación para asemejarlo al material quirúrgico que se emplearía en las actuaciones reales, en concreto se busca que se asemeje a unas tijeras.

Figura 11. Geomagic Touch con modificación para herramienta quirúrgica [14].

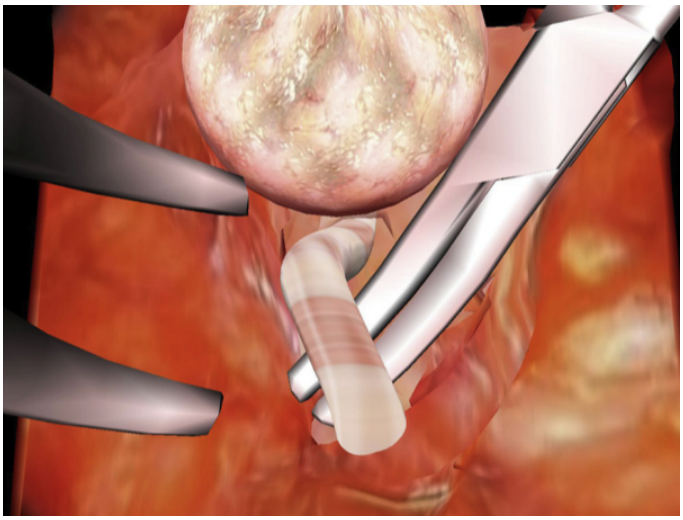

Figura 12. Escenario virtual desarrollado [14].

En el artículo se detalla cómo tras diez iteraciones de la operación simulada, se consigue disminuir en mayor medida el tiempo de operación respecto a un grupo de control que no se entrenó con el sistema. Por otra parte, se demuestra una reducción drástica del número de errores a medida que se aumenta la práctica, errores que se hubiesen cometido sobre pacientes reales si no se hubiese contado el sistema presentado. Se constata así que su uso como herramienta de formación consigue resultados positivos de cara a operaciones reales.

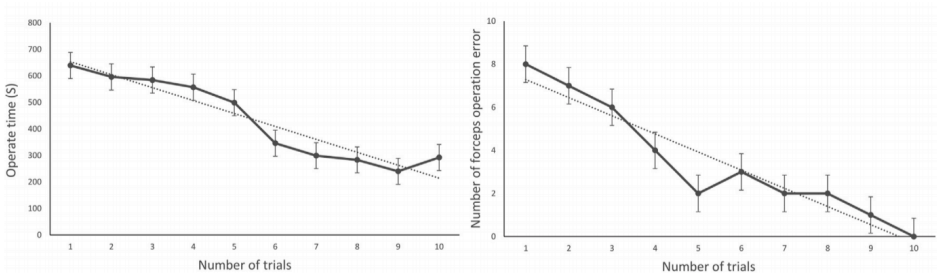

Figura 13.(a) Relación iteraciones-tiempo de operación. (b) Relación iteraciones-errores cometidos [8].

2.2.- Robotic Operating System (ROS).

El continuo desarrollo de los robots ha permitido la implantación de estos equipos en ámbitos de diversa índole. Como es natural, un sistema robótico diseñado para el uso industrial deberá contar con características diferentes respecto a otro que ofrezca un servicio en el campo de la agricultura. La diversidad de aplicaciones y empresas fabricantes, sumada a la ausencia de una normativa en cuanto a su programación, ha propiciado la disparidad actual en el *software* de estos equipos.

Alguna de las consecuencias que esto supone, es la dificultad o imposibilidad de la reutilización de código con el objetivo de lograr una misma meta, si ésta se busca alcanzar mediante el uso de equipos de diferente modelo o marca. Además, la complejidad que han ido adquiriendo los robots progresivamente, supone la creación de un campo de estudio en constante evolución, demasiado grande como para que una única persona pueda alcanzar un conocimiento general.

Se hace por lo tanto necesario contar con un mecanismo que sirva como marco de referencia, que nos permita desarrollar el software necesario para acometer tareas complejas y con el que podamos propiciar la interconexión de elementos con diferentes características. Se crearon y existen diversos proyectos con este fin, siendo *ROS* (*Robotic Operating System*), uno de los que cuentan con una mayor implantación.

El objetivo de *ROS,* es el de crear una plataforma que propicie el desarrollo de software por parte de una comunidad de usuarios anónimos, pudiendo unos aprovecharse del trabajo y conocimiento de otros para desarrollar sus propios proyectos. Este trabajo se lleva a cabo mediante la generación de librerías o ejecutables con finalidades concretas, que pueden ser utilizadas en distintos sistemas robóticos. Otro de los aspectos a tener en cuenta de *ROS*, es la interoperabilidad entre sistemas que propicia. Implementa una serie de mecanismos de intercambio de información, que independientemente de la naturaleza de la programación de cada sistema, pueden gestionar ese servicio.

ROS no es un sistema operativo en los mismos términos que *Linux*, *MacOS* o *Windows*, ya que necesita de las operaciones de uno de ellos (*Linux* o *Windows* de manera experimental) para llevar a cabo sus tareas. Si bien es cierto, que presenta alguna de sus características.

Cuando hablamos de sistema operativo, nos referimos a un software que ejerce como interfaz entre aplicaciones y hardware, encargándose de la administración de recursos como la memoria, el acceso a procesador etc... *ROS*, ofrece una serie de servicios a los procesos que se ejecutan bajo su marco. Debiendo responder a las solicitudes que se registren y gestionando los recursos en caso de competencia por parte de los distintos procesos.

Otra característica de los sistemas operativos es que no están conformados por estructuras monolíticas, su *kernel* se divide en una serie de módulos que trabajan juntos para alcanzar una meta común.

De la misma manera, *ROS* cuenta con una serie de librerías y paquetes que pueden incluirse de cara a conseguir acceso a una mayor o menor cantidad de recursos, dependiendo de las necesidades que presente nuestro proyecto.

Desde su aparición en 2010, se han ido desarrollando nuevas iteraciones de este marco de referencia, siendo cada una de ellas diseñada específicamente para una distribución concreta de *Linux*.

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Kinetic Kame (**Recommended**)	May 23rd, 2016			May, 2021
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)

Figura 14. Ejemplo de distribución, fecha de lanzamiento y fin de soporte técnico.

Por ejemplo, si estuviésemos utilizando *Ubuntu* 14.04, deberíamos servirnos de la distribución de *ROS Indigo*, mientras que si utilizásemos *Ubuntu* 16.04, tendremos que optar por la distro *ROS Kinetic*, contando cada una de ellas con soporte técnico durante un plazo de cinco años desde la fecha de su lanzamiento.

2.2.1.- Características de diseño.

Como se ha dicho anteriormente, la naturaleza de los sistemas robóticos es tan diferente como la del uso que se les quiera proporcionar, en consecuencia, la creación de un *framework* capaz de responder a tal diversidad deberá atender a una serie de priorizaciones y concesiones en su diseño. Debido a estas decisiones, podemos definir a ROS como una plataforma que presenta las siguientes características:

- Comunicación *peer-to-peer.*
- Herramienta multi-lenguaje.
- Diseño modular (tools-based).
- Gratis y de código abierto.

Comunicación *peer-to peer*.

Por norma general, el *software* creado utilizando *ROS*, estará compuesto por una serie de procesos que intercambian información utilizando una topología *peer-to-peer*. Éstos conforman una red en la que no se establecen clientes ni servidores fijos, sino que se crea una estructura en la que todos los nodos cuentan con una posición jerárquica semejante, y en la que estos roles se pueden modificar dependiendo de las necesidades del conjunto en un momento determinado.

Uno de los beneficios que presenta esta topología es la posibilidad de utilizar sistemas descentralizados. Las primeras fases de diseño de *ROS* se orientaron hacia una serie de robots que poseían elementos móviles, en los que se encontraban varias unidades de procesamiento a bordo enlazadas vía Ethernet, conectadas a otra unidad de procesamiento central situada en un emplazamiento fijo, y con la cual mantenían procesos de intercambio de información mediante el uso de una red *WLAN*.

Si se hubiese decidido optar por una topología centralizada en la que existiese un proceso dominante, aparecería la necesidad de mantener un tránsito de datos constante a través de la relativamente lenta red inalámbrica. Por el contrario, hacer uso de la tecnología *peer-to-peer* nos otorga la capacidad de limitar ese tránsito al mínimo necesario, manteniéndose dentro de cada subred el tráfico destinado a los procesos situados en ellas.

Herramienta multi-lenguaje.

Otra de sus características es la posibilidad de escribir código utilizando distintos lenguajes, por ejemplo: *C++, Octave* o *Python*. El estándar de *ROS* se encuentra en la capa de envío de mensajes, en la que los procesos de comunicación *peer-to-peer* se realizan mediante el protocolo *XML-RPC*.

Se decide hacer uso de este procedimiento en lugar de otro basado en uno de los lenguajes mencionados previamente, ya que cuenta con mecanismos de implementación para la mayoría de los lenguajes de programación con un uso más extendido, contando con la posibilidad de utilizar adaptaciones (*wraps*) en aquellos para los que no sea así.

En cuanto a la definición de los mensajes, se utiliza el "lenguaje de descripción de interfaz" (*IDL* por sus siglas en inglés), el cual, mediante la utilización de archivos de texto con muy pocas líneas de código, define los campos de cada mensaje. *ROS* entonces, implementa mecanismos que permiten la "traducción" de unos lenguajes a otros, recibiendo cada nodo objetos que pueden ser tratados como nativos.

Diseño modular (*tools-based*).

En lugar de crear un único y complejo entorno en el que aglutinar los distintos recursos necesarios para el funcionamiento de *ROS*, sus creadores optaron por un enfoque basado en la utilización de múltiples núcleos para la ejecución del *kernel*, almacenando cada uno de ellos una serie de herramientas. Este compromiso en el diseño presenta inconvenientes como la pérdida de velocidad en la ejecución y ventajas como una menor complejidad y mejor estabilidad, lo cual beneficia a la incorporación futura de nuevas funcionalidades.

Gratuito y de código abierto.

Uno de los principales atractivos de *ROS* es la creación de nuevas herramientas por parte de grupos e individuales con fines tanto comerciales como no comerciales.

Para facilitar su tarea, se mantiene el acceso a su código fuente de manera completamente gratuita y se emplea Linux de manera preferente.

Con el objetivo de facilitar el desarrollo por parte de la comunidad, se recomienda el desarrollo de drivers y algoritmos en librerías, situando en ellas toda la complejidad posible. Esto implica que se podrán programar ejecutables sencillos, que tendrán la capacidad de reutilizados más allá de su diseño original.

2.2.2.- Conceptos básicos.

Una vez abordados los principios de diseño de *ROS*, paso a hacer una presentación de sus conceptos básicos.

2.2.2.1.- Sistema de archivos.

Paquetes: son la principal unidad organizativa de almacenamiento en *ROS*. Contienen librerías, ejecutables (nodos), archivos de datos o cualquier otra información que utilizada en conjunto pueda servir para la realización de una actividad concreta.

Dependencias: son librerías o herramientas externas que han de ser provistas por el sistema operativo. *ROS* cuenta con la herramienta *rosdep* para su descarga e instalación.

Pila: conjunto de paquetes que comparten una finalidad análoga.

Metapaquetes: paquetes especializados que sirven únicamente para representar un grupo de paquetes relacionados.

***Package Manifests*:** (*package.xml*) aportan metadatos acerca de un paquete como su nombre, descripción, licencia, dependencias, versión…

Repositorios: ficheros compuestos que tienen como finalidad permitir la descarga de uno o más paquetes y pilas de manera simultánea.

Archivos ".msg": definen las estructuras de datos utilizadas en los mensajes de *ROS*.

Archivos ".srv": definen las estructuras de las solicitudes y respuestas realizadas por los servicios de ROS.

2.2.2.2.- Procesamiento.

Nodos: son cada uno de los procesos informáticos que deben ejecutarse con el objetico de alcanzar un fin determinado. Tareas complejas generalmente necesiten de la ejecución de múltiples nodos, ya que en ROS se promueve el uso de cada nodo para la elaboración de una tarea muy concreta.

Maestro: proceso encargado del registro de servicios, nombres y búsqueda de direcciones para los nodos en ejecución. Su actividad es necesaria en el envío de mensajes, petición de servicios y en ciertas ocasiones para el almacenamiento de datos.

Mensajes: estructuras de datos que los nodos utilizan para comunicarse. En estos intercambios de información pueden utilizarse tipos de datos standard como *integers*, *boolean* o *floating point*.

***Topics*:** cada uno de los canales de comunicación que se generan entre nodos, los cuales se identifican mediante un dato tipo *string*. Si un nodo quiere enviar un mensaje a otro, deberá convertirse en publicador en un *topic*. Por el contrario, si lo que quiere es recibir una información, deberá suscribirse al *topic* encargado de portarla. Múltiples publicadores o suscriptores pueden hacer uso de un *topic*, teniendo también la posibilidad un nodo de actuar como publicador o suscriptor en uno o más *topics*.

Servicios: mientras que el mecanismo publicador- suscriptor de los mensajes puede ser utilizado en multitud de situaciones, no es el más adecuado para otras. En las transacciones de información en los que se quiera o necesite utilizar el paradigma solicitud-respuesta, utilizaremos los servicios. En este caso un nodo ofrecerá un servicio esperando que un nodo cliente lo solicite. Cuando esto ocurra el nodo servidor tramitará dicha solicitud y enviará un mensaje respuesta.

***Bags*:** son archivos utilizados para el almacenamiento de datos enviados a través de mensajes. Son un importante mecanismo que puede servir como depósito de datos provenientes de sensores en la fase de testeo o desarrollo de algoritmos.

2.2.3.- Funcionamiento.

Como he explicado previamente, el nodo maestro de *ROS* proporciona funciones de registro de nombres y servicios que permiten la comunicación entre otros nodos. Se puede poner en funcionamiento mediante el comando *roscore,* o utilizando *roslaunch*, el cual iniciará los nodos concernidos por dicho comando, además del nodo maestro si este no fue activado previamente.

En función de las características que posean los mensajes o servicios a utilizar por parte de los procesos activos, deberán reflejarse en el o los archivos *package.xml* de cada paquete las instrucciones que invoquen las dependencias necesarias de la API basada en protocolo *XML-RPC* en que se basa el mecanismo de transmisión de información en *ROS*.

Un ejemplo sencillo del funcionamiento de los mecanismos de comunicación en *ROS* es el siguiente. Contamos con dos nodos: "/teleop_turtle" y "/turtlesim_node". El primero realiza las tareas necesarias para transformar en consignas de velocidad lineal y angular los comandos recibidos a través de teclado, mientras que el segundo se encarga de mostrar por pantalla la imagen de una tortuga, desplazándose por un entorno bidimensional, de acuerdo con las consignas producidas por el primer nodo.

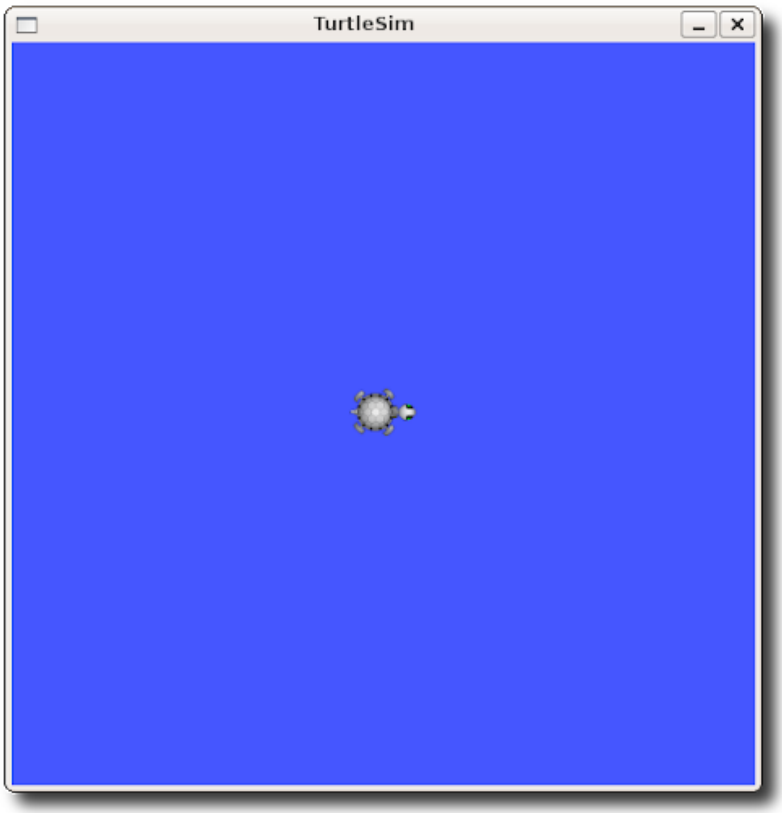

Figura 15. Interfaz de usuario de "turtlesim_node".

Para que "/turtlesim_node" pueda recibir información de "/teleop_turtle", deberá establecerse un canal de comunicación. En este caso se utilizará un *topic* llamado "/turtle1/command_velocity", a través del que se enviará una estructura tipo *geometry_msgs/Twist.msg,* conformada de la siguiente manera:

geometry_msgs/Twist.msg	*geometry_msgs/Vector3 linear*	*float 64 x* *float 64 y* *float 64 z*
	geometry_msgs/Vector3 angular	*float 64 x* *float 64 y* *float 64 z*

Al inicio de la ejecución de los nodos, "/teleop_turtle" deberá notificar al maestro su intención de convertirse en publicador de "/turtle1/command_velocity", quedando el canal iniciado, pero sin existir aún transmisión de información, puesto que todavía no se cuenta con ningún suscriptor que reciba la información. Es entonces cuando al iniciar "/turtlesim_node", éste informará a maestro de la intención por su parte de ser suscriptor de un *topic* existente de nombre "/turtle1/command_velocity".

Es entonces, cuando una vez establecida la relación publicador-suscriptor, el nodo maestro notifica a cada nodo la existencia del otro, pudiéndose iniciar el proceso de transmisión de información, que se mantendrá hasta el cese de la ejecución de los programas o de *roscore*.

Figura 16. Diagrama funcionamiento.

2.2.4.- Aplicaciones.

Debido a las características de diseño que se han discutido anteriormente, *ROS* se ha convertido en la herramienta a la que recurrir para la elaboración de casi cualquier tarea relacionada con la robótica. La gran cantidad de recursos y documentación generada por la comunidad de *ROS*, permite que una persona o grupo de trabajo que busque acometer una nueva tarea, va a contar casi con toda seguridad con paquetes o al menos información que les pueda ayudar a alcanzar su meta.

Además, el sencillo procedimiento de comunicación entre nodos permite la adición de nuevas funcionalidades con cierta facilidad, lo cual permite el desarrollo continuo de las aplicaciones en las que se trabaje.

Algún ejemplo concreto del uso de *ROS* es la programación de *turtlebots*, robots que mediante el uso de visión artificial o cualquier otro medio de detección de obstáculos además de la programación pertinente, son capaces de desplazarse por una superficie. Versiones baratas de estos dispositivos pueden utilizarse de manera lúdica con la intención de iniciarse en el mundo de la robótica, pero también pueden ser y son utilizados en aplicaciones más profesionales. Por ejemplo, en [15], se detalla el uso de un robot *TurtleBot 2* para la generación del mapeado de una serie de estancias, utilizando cámaras y herramientas nativas o coordinadas a través de ROS.

2.3.- Robots colaborativos.

Cuando hablamos de robots colaborativos o *cobots*, estamos haciendo referencia a sistemas robóticos que poseen la capacidad de realizar un trabajo de manera segura en las proximidades de un ser humano. Éste es un término muy amplio en el que podemos englobar tanto a robots de uso doméstico como dedicados a la manufactura, campos en los que el uso de estos sistemas se ha ido extendiendo debido a su versatilidad.

Hasta mediados de los años 90, los robots industriales carecían de mecanismos integrados que detuvieran sus actuaciones en caso de accidente, únicamente se contaba con barreras físicas, mecanismos de prevención o botones de parada de emergencia, siendo todos estos sistemas dependientes del factor humano. Es a partir de esos años cuando se comienza a valorar el desarrollo de sistemas que puedan compartir un espacio con personas sin suponer un riesgo para ellas.

El primero de estos sistemas, el primer *cobot*, se acredita a los profesores J. Edward Colgate y Michael Peshkin, quienes en 1996 desarrollan y presentan junto a Witaya

Wannasuphoprasit un artículo en el que detallan el funcionamiento de dos prototipos a los que llaman “Unicycle” y “Bicycle” *cobots* [16].

Su funcionamiento es el de un interfaz háptico que define dos zonas: una accesible en la que el sistema se desplaza con la menor resistencia posible y otra a la cual el sistema impedirá el acceso, quedando delimitadas ambas por una frontera que se describe como “pared virtual”.

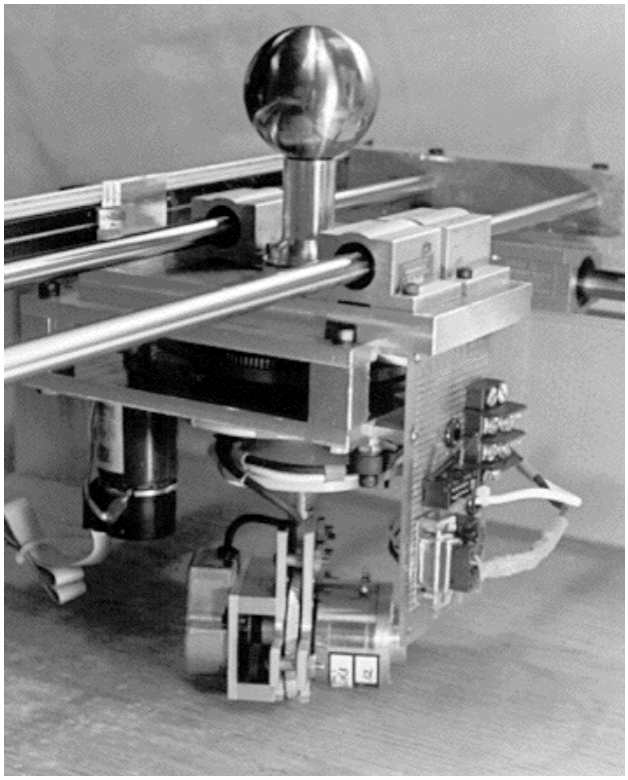

Figura 17. Cobot "unicycle" [16].

El sistema “Unicycle” consta de una empuñadura que operará el usuario del sistema, la cual reposa sobre un elemento pasivo que desliza sobre railes en las direcciones X e Y. Debajo de él tenemos una rueda cuya dirección será determinada por un sensor de fuerzas y un motor. Si dentro de la zona accesible el sensor detecta fuerzas tangenciales, el motor orientará la rueda de tal manera que se encuentre la posición de mínima resistencia. Por el contrario, si se alcanza la pared virtual, el motor la hará girar hasta alcanzar la posición perpendicular a la “pared virtual”, impidiendo el movimiento en su dirección a no ser que se venza la fuerza ejercida por el sistema y se haga deslizar la rueda.

El modelo “Bicycle” se desarrolla como una mejora del sistema anterior, haciendo posible restringir la orientación además del movimiento en las direcciones X e Y del plano.

Partiendo de esta base, comenzaron a desarrollarse las primeras iteraciones industriales de estos sistemas, los cuales recibían el nombre de “Sistemas de Asistencia Inteligente” por su traducción del inglés.

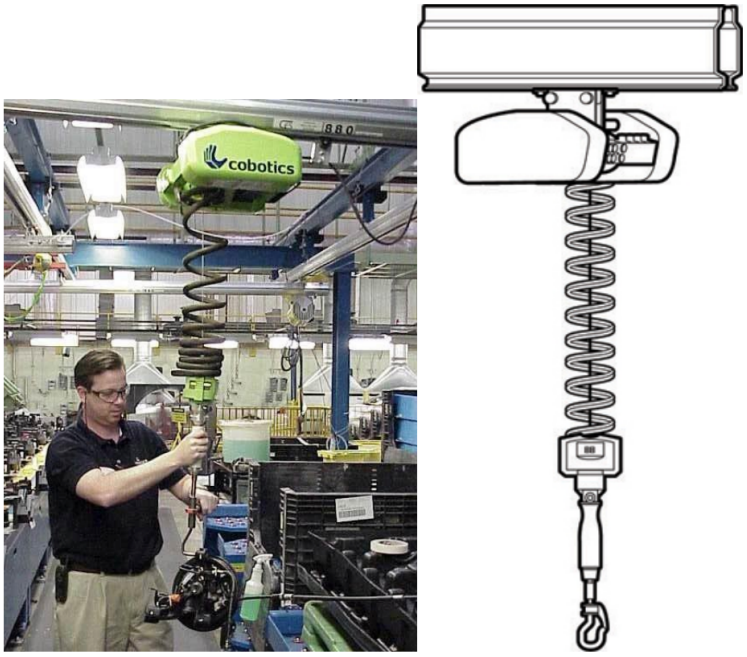

Figura 18. Ejemplo de Sistema de Asistencia Inteligente [16].

Eran sistemas que complementaban las capacidades del operario, es decir, carecían de mecanismos que les permitiesen moverse por sí mismos, siendo el operario a su cargo la que determinaba su posición. Su finalidad no era la de remplazar al elemento humano si no la de reducir su carga de trabajo, tarea que realizaban de las siguientes maneras [16]:

- Amplificación de fuerza: sostenían pesos sin necesidad de esfuerzo por parte del operario.
- Enmascaramiento de inercias: reducían las fuerzas necesarias en el inicio y final de los movimientos además de en los giros. Se buscaba un comportamiento homogéneo independientemente de la dirección en que se realizara el desplazamiento.
- Guiado mediante el uso de superficies y sendas virtuales.
- Monitorización con fines de control. Mediante el uso de sensores pueden obtenerse mediciones de pesaje o seguimiento de las líneas de producción.
- Comunicación con los sistemas de información de la planta a fin de registrar datos y detectar errores.

En 2005 se da el siguiente paso en el desarrollo de esta tecnología con la aparición de la empresa Universal Robots. Esta compañía nace con el objetivo de crear un brazo robótico, capaz de operar de manera segura para los trabajadores, pero sin necesidad de que uno de ellos determine manualmente sus movimientos. Esta meta se alcanza tres años después con el lanzamiento del *cobot* UR5.

Este sistema cuenta con limitaciones tanto en la velocidad de sus movimientos como en la fuerza que es capaz de ejercer, además de estar producido con materiales ligeros y bordes redondeados. Todo ello dirigido a minimizar las consecuencias en caso de que se produzca un escenario inesperado.

Cuenta con otra característica que lo diferencia de los robots precedentes, que es el entorno de programación. Éste está diseñado de manera que cualquier persona, independientemente de su grado de preparación, sea capaz de implementar las instrucciones necesarias para conseguir el comportamiento deseado por parte del *cobot*. Esto reduce los costes de instalación y en consecuencia facilita el acceso a la automatización a empresas menos capaces de realizar inversiones para estos fines.

Figura 19. Robot colaborativo UR5. [17]

A partir de la aparición del sistema creado por Universal Robots, diversas empresas comenzaron a diseñar sus propios sistemas, pero manteniendo los principios anteriormente mencionados del *cobot* UR5. Se desarrolla además un nuevo concepto: la colaboración hombre-robot. Es decir, se busca que los robots colaborativos no solo sean capaces de operar de manera segura compartiendo un mismo espacio, sino que cada uno con sus habilidades, pueda complementar la tarea del otro.

Podemos poner como ejemplo los *cobots* "ABB YuMi", "KUKA–LBR iiwa" o "Baxter" de la empresa "Rethink Robotics", los cuales, mediante el uso de sensores, cámaras o la combinación de ambas, pueden reaccionar a un entorno cambiante y modificar sus respuestas en consecuencia. Algunos incorporan además formas humanas, de tal manera que aquellos que se encuentren en su entorno puedan reconocer en ellos rasgos más familiares. Por ejemplo, el *cobot* "Baxter" de "Rethink Robotics" incorpora un *display* en el que se pueden ver unos ojos que representan una serie de expresiones en función de la actividad que esté llevando a cabo el robot.

Figura 20. Robot colaborativo "Baxter" [18].

2.3.1.- Aplicaciones.

Como he señalado anteriormente, las características que poseen los robots colaborativos les han permitido trascender el ámbito industrial, siendo empleados en un número de escenarios creciente. Un ejemplo de ello son los modelos “Bot Handy” y “Bot Care”, recientemente presentados por la empresa surcoreana Samsung.

Figura 21. Samsung Bot Care (izq) y Handy (dcha) [19].

Según indica el fabricante, Samsung Bot Handy se servirá de la inteligencia artificial y de una serie de cámaras para reconocer objetos de distintos tamaños, formas y materiales, de tal manera que pueda recogerlos con su brazo robótico ejerciendo una fuerza apropiada. Se encuentra aún en fase experimental y no se proporcionan especificaciones, pero se anuncia que podrá realizar tareas del hogar como llenar un lavavajillas o servir bebidas tanto de manera autónoma como colaborativa con un ser humano.

Por otra parte, Samsung Bot Care se ha creado con la intención de actuar como un robot asistente, que pueda monitorizar la actividad de una persona para ofrecerle recomendaciones que ayuden a mejorar su estado de salud.

Existe también la posibilidad de utilizar robots colaborativos dentro de un ámbito hospitalario. En [20] se detalla el diseño de una estrategia de control de un robot colaborativo Universal Robots UR3, dirigida a la realización de operaciones quirúrgicas mediante la técnica de la endoscopia, pensada en concreto para la realización de cirugía endoscópica endonasal.

La trayectoria a seguir por el robot durante el ejercicio de sus funciones será determinada durante el preoperatorio, sin embargo, en caso de que existan discrepancias entre el modelo realizado previamente, y la situación del entorno o de las herramientas médicas en el momento de la operación, esta trayectoria deberá ser modificada. Se implementa para ello un sistema de control que, en caso de percibirse una fuerza durante el avance, modifique la trayectoria planificada de tal manera que se consiga disminuir en la mayor medida posible el contacto con los obstáculos presentes.

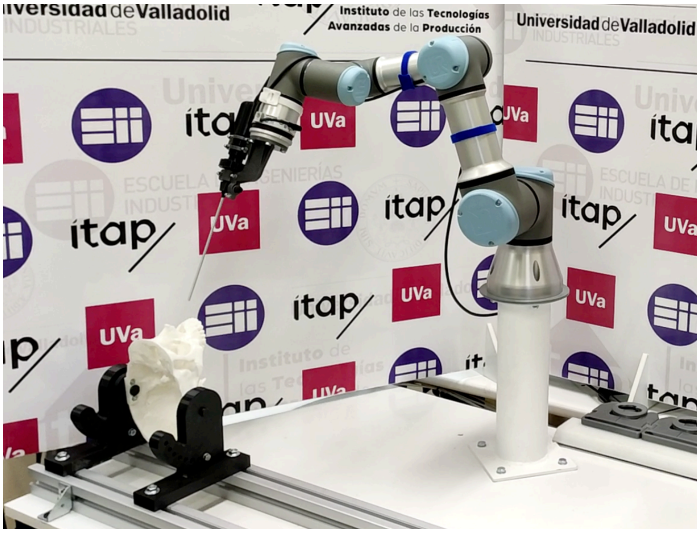

Figura 22. Modelo experimental desarrollado para el ejercicio de cirugía endoscópica endonasal [20].

Para terminar, reseñar la implantación de los robots colaborativos en la industria, donde llegan a suponer el 28% del total de los robots en el sector automotriz, según la Federación Internacional de Robótica (IFR por sus siglas en inglés) [21].

Nissan por ejemplo, tomó la decisión de comenzar a utilizar estos sistemas en una de sus fábricas en Japón junto a empleados humanos. Se aprovecha la posibilidad de que hombre y máquina compartan espacios para organizar actividades de manera simultánea, ahorrándose así tiempo respecto a una planificación secuencial de los mismos eventos.

Otra empresa, esta vez del sector agro-alimenticio llamada Atria Scandinavia fabrica una serie de productos con características variadas, debiéndose ajustar frecuentemente la cadena de producción. Puesto que no existe la necesidad de mantener vallados de seguridad, los empleados pueden realizar estos cambios con mayor celeridad, revirtiendo de manera positiva en la producción.

3.- Descripción de los sistemas

En este capítulo se lleva a cabo una descripción de los sistemas físicos (*hardware*), y de los programas informáticos (*software*) que he utilizado. Se expondrán las capacidades técnicas de los sistemas, para posteriormente explicar su funcionamiento a través del software que rige y administra sus capacidades.

3.1.- Interfaz háptico.

3.1.1.- Hardware.

Figura 23. Sistema háptico 3D Systems Touch. [6]

El dispositivo que se utilizará es el *3D Systems Touch* de la compañía *3D Systems*. Es un interfaz háptico controlado por impedancia, que utiliza el método *proxy* como modelo de contacto. Partiendo de lo expuesto en el apartado 2.1.2., cuando la interacción háptica se localiza en un único punto, se convierte en necesario la utilización de alguna herramienta que otorgue un carácter más intuitivo a la operación del sistema.

En este caso se ha optado por asemejar el último eslabón del brazo robótico a un bolígrafo, el cual recibe el nombre de "*stylus*". Su manipulación por parte del usuario determinará la posición del HIP dentro del escenario en el que se utilice el interfaz háptico. El punto del dispositivo cuyos movimientos se representan en dichos escenarios y que colisiona con los objetos que allí se encuentren, se sitúa de acuerdo con lo mostrado en la imagen 23.

Mediante la ejecución de fuerzas por parte de los motores situados en el brazo robótico del interfaz, podremos percibir la dureza, flexibilidad, fricción y formas de los distintos objetos que hayan sido colocados dentro del entorno en el que se actúa, además una serie efectos hápticos de los que hablaré más adelante.

Figura 24. Stylus con localización del punto de interfaz háptico (HIP). [22]

Entrando al detalle de sus características técnicas, éstas son las más relevantes [22] [23]:

- Resolución: 0,055 *mm.*
- Fuerza máxima: 3,3 *N* (en cada dirección cartesiana).
- Viscosidad máxima: 0,26 *N*.
- Inercia (masa aparente en la punta): 0,045 *Kg*.
- Rigidez: en eje X >1,26 *N/mm*, en eje Y 2,31 *N/mm*, en eje Z 1,02 *N/mm*.
- Force Feedback: en ejes X, Y, Z.
- Posicionamiento: en X,Y,Z mediante encoders digitales y orientación mediante potenciómentros con resolución del 5%.
- Lazo de control: 1000 *Hz* (regulable por software).

Nominal position resolution	> 450 dpi ~ 0.055 mm
Backdrive friction	< 1 oz (0.26 N)
Maximum exertable force at nominal (orthogonal arms) position	0.75 lbf (3.3 N)
Continuous exertable force (24 hrs)	0.2 lbf (0.88 N)
Stiffness	X axis > 7.3 lbs / in (1.26 N / mm)
	Y axis > 13.4 lbs / in (2.31 N / mm)
	Z axis > 5.9 lbs / in (1.02 N / mm)
Inertia (apparent mass at tip)	~0.101 lbm (45 g)
Force feedback (**3** Degrees of Freedom)	x, y, z
Position sensing [Stylus gimbal]	x, y, z (digital encoders)
(**6** Degrees of Freedom)	[Pitch, roll, yaw (± 5% linearity potentiometers)

Figura 25. Características técnicas más relevantes del dispositivo [22].

La información relativa a la situación espacial del *HIP* es generada por los valores aportados por los *encoders* digitales situados en las articulaciones J1, J2 y J3 del sistema, mientras que la relativa a la orientación es dependiente de los valores arrojados tanto por dichos *encoders* como por los potenciómetros situados en J4, J5 y J6 (*yaw, pitch* y *roll* respectivamente).

Figura 26. Articulaciones con encoder digital (posición). [22]

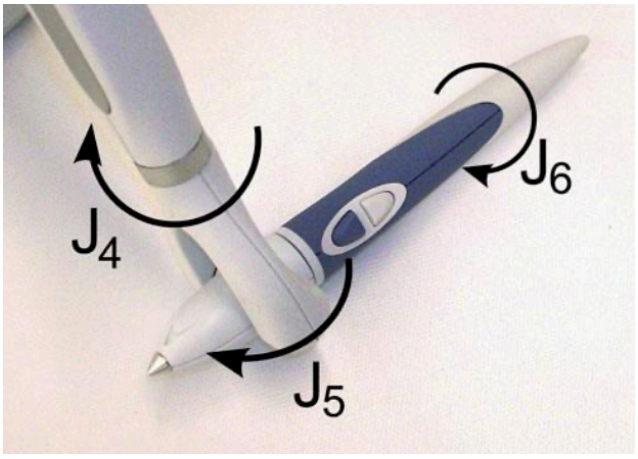

Figura 27. Articulaciones con potenciómetro (orientación). [22]

Desde un punto de vista cinemático, puede analizarse el sistema mediante los parámetros de Denavit-Hartemberg. Éstos han sido calculados y se recogen en la Tabla 1:

Tabla 1. Parámetros Denavit-Hartemberg del dispositivo.

Articulación	α_{i-1} (deg)	a_{i-1} (mm)	d_i (mm)	Θ_i (Ángulo límite(deg))
1	0º	0	169	$\Theta_1 \in [-58.3º, 54.4º]$
2	– 90º	0	0	$\Theta_2 \in [-2.1º, 102º]$
3	0º	134	0	$\Theta_3 \in [-46º, 68.6º]$
4	– 90º	0	134	$\Theta_4 \in [-144º, 149º]$
5	90º	0	0	$\Theta_5 \in [-83.2º, 60.8º]$
6	– 90º	0	0	$\Theta_6 \in [-150º, 150º]$

El dispositivo cuenta con un eje de coordenadas predeterminado, siendo las direcciones de sus ejes las mostradas en las ilustraciones 28 izquierda y derecha. El origen de coordenadas se encuentra a 153 *mm* de altura respecto la base, 95 *mm* en dirección positiva del eje Z respecto el borde inferior del *inkwell* (orificio en el que se guarda el *stylus* en la posición de reposo), y dentro del plano de simetría de la base.

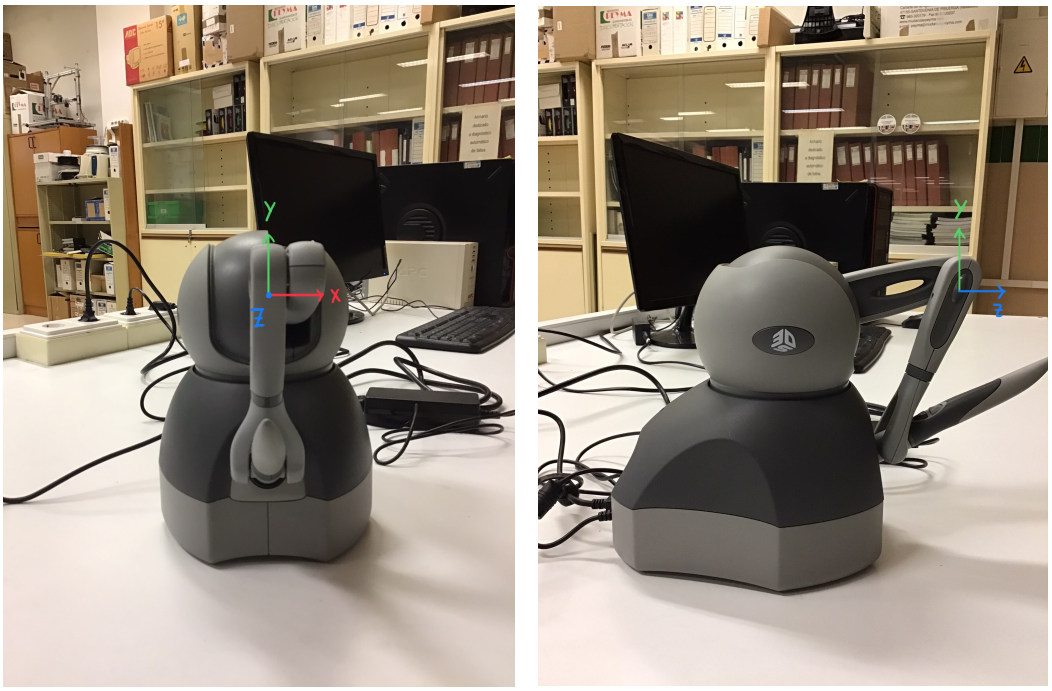

Figura 28. Direcciones X,Y,Z del sistema de referencia.

Figura 29. Localización del borde inferior del Inkwell.

Pasando al ámbito de la renderización de fuerzas, el dispositivo cuenta con tres motores: uno por cada una de las tres primeras articulaciones (J1, J2 y J3), dejando libres en su movimiento a las tres últimas (J4, J5 y J6). A causa de ello, el sistema solo podrá generar fuerzas en la dirección de los ejes cartesianos (3 GDL). En modelos más avanzados de la gama de dispositivos hápticos de 3D Systems, sí que se pueden encontrar motores en todas las articulaciones, lo que permite la programación de momentos de torsión en los últimos eslabones (6 GDL). El punto de aplicación de fuerzas, como ocurre en los interfaces hápticos de esta clase, se sitúa en el mismo lugar que el *HIP*.

El dispositivo cuenta con una serie de botones que nos permiten sumar usos, como puede ser la sujeción de objetos o cualquier otra opción que el usuario plantee introducir mediante software. Dos de ellos se sitúan en el *stylus*, a los que se ha identificado como 0 y 1, estando presente un tercero en el *inkwell*. Este último únicamente sirve para informar si el dispositivo se encuentra en posición de reposo, en cuyo caso la renderización de fuerzas es desactivada.

Figura 30. Botonera del stylus.

Por último, indicar que el *inkwell* cuenta con un diodo *LED* que nos informa acerca del estado del dispositivo. Si conectamos a la red el cable de alimentación y a una entrada *USB* el cable de datos, el *LED* se encenderá ligeramente. Si iniciamos una simulación éste se iluminará con mayor intensidad y si se produce un error durante la simulación parpadeará durante aproximadamente un segundo antes de retornar al estado de menor intensidad.

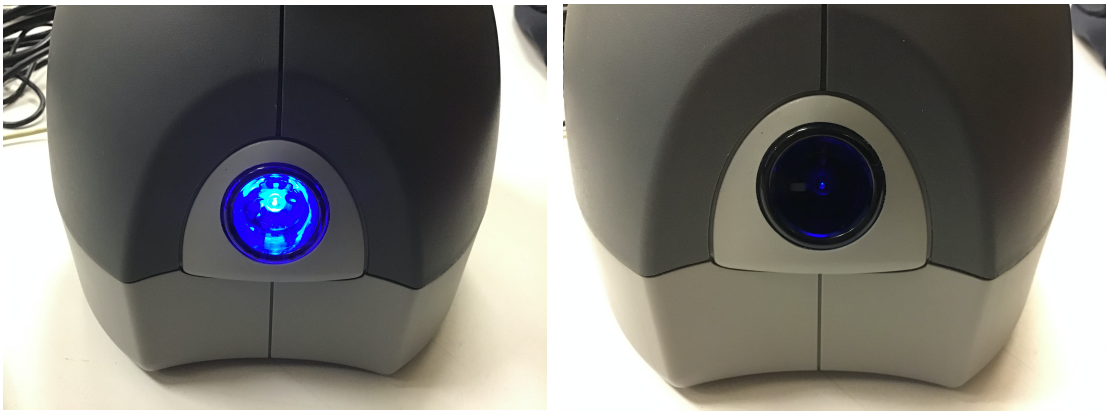

Figura 31. (Izq) Mayor intensidad. (Dcha) Menor intensidad.

3.1.2.- Software.

Para operar el dispositivo háptico con nuestro ordenador, deberemos descargar el *OpenHaptics® Toolkit* versión 3.5.0. En este *Toolkit* se incluyen el *Quick Haptics Micro API*, la *Haptic Device API* (*HDAPI*), *Haptic Library API* (*HLAPI*), los drivers del sistema (*PDD*), códigos de ejemplo y una serie de recursos.

Entre estos recursos se incluyen aplicaciones que permiten conocer el estado del dispositivo y/o calibrarlo, como por ejemplo *Touch Diagnostic, Setup* y *Smart Setup*, o aplicaciones que sirven como demostración de las capacidades del dispositivo como *Touch Demo* u *OpenHaptics Unity Beta Demos*.

Las *APIs* (*Application Programming Interfaces*) son un conjunto de definiciones y protocolos que permiten la comunicación entre dos aplicaciones software mediante un conjunto de reglas determinadas [24]. En este caso se utilizan para mantener una comunicación entre el entorno de desarrollo del código (por ejemplo *Visual Studio*) y las instrucciones de bajo nivel que determinan el comportamiento del dispositivo háptico, sirviéndose además de las funcionalidades de conexión entre software y dispositivo que ofrecen los *PHANTOM Device Drivers.*

Hay que tener en cuenta que en el transcurso de las simulaciones o escenarios en los que se desenvuelve el sistema háptico, debemos conjugar dos aspectos: el táctil y el visual. Cada uno de los cuales posee su propio hilo de programación, teniendo siempre preferencia el háptico sobre el visual en caso de competencia sobre recursos.

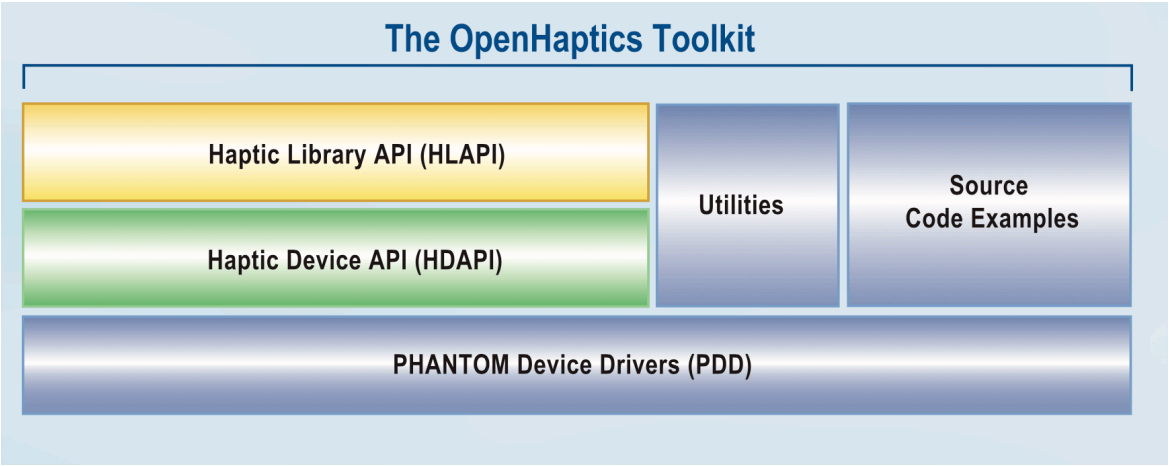

Figura 32. Contenido del OpenHaptics Toolkit [25].

Dentro del ámbito visual, pueden utilizarse aplicaciones como *Unity* a través del *3D Systems Openhaptics Unity Plugin*, (que también nos permite la introducción de físicas en el plano háptico a través de la *HLAPI*), o hacer uso de librerías como *OpenGL* (*Open Graphics Library*), que define una *API* multilenguaje y multiplataforma que nos permite generar y manipular escenarios en dos y tres dimensiones.

Centrándonos en *OpenGL*, éste recibirá parámetros del espacio de trabajo del dispositivo háptico, de tal manera que pueda crear un escenario con dimensiones acordes. En cada iteración del graphics loop, recibirá nueva información, generalmente acerca de la posición y orientación del *stylus*, a los que se le podrán añadir otros datos como información del estado de los botones o fuerza ejercida por los motores en caso de que se quiera mostrar esta información a través de una interfaz de usuario. La representación visual se actualizará con una frecuencia de 30 *Hz*.

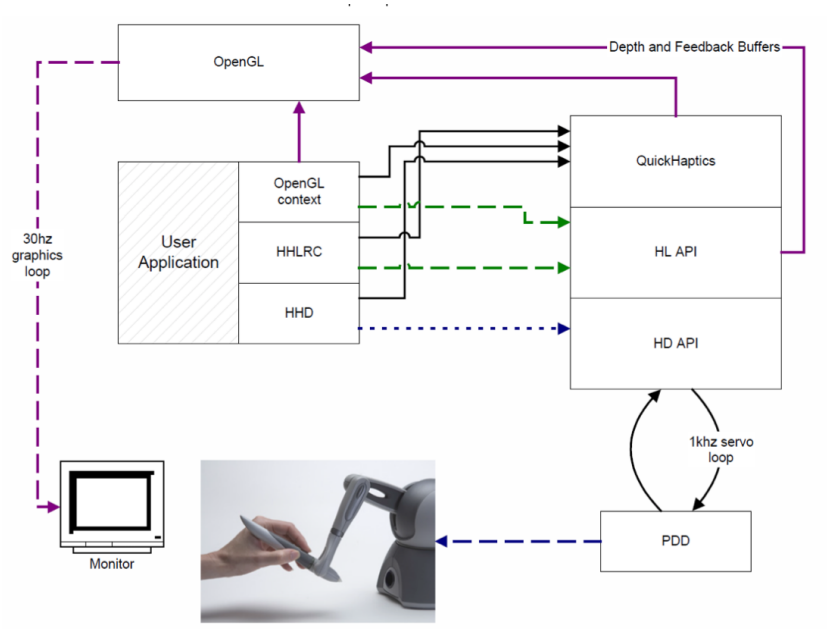

Figura 33. Esquema de funcionamiento de los hilos de programación [10].

Para la generación de fuerzas nos serviremos de las funciones y parámetros que nos ofrecen las distintas *APIs*. Como veremos a continuación, podremos extraer distintas funcionalidades a costa de una mayor o menor complejidad en función de la *API* que escojamos. El hilo encargado de administrar el intercambio de información entre ordenador y dispositivo deberá realizarse con una frecuencia de 1000 *Hz* como se muestra en la ilustración anterior.

El valor de la frecuencia del ciclo de servicio no se ha escogido en vano: mientras que el sentido de la vista toma imágenes unas 60 veces por segundo, el del tacto necesita alrededor de 1000 muestras durante el mismo periodo de tiempo, por lo que el sistema deberá actualizarse de acuerdo con ello. El *servo loop* puede regularse a valores entre los 500 y los 2000 *Hz*, aunque tal vez no sea lo más recomendable, ya que disminuyendo el valor podríamos perder información, y aumentándolo generaríamos un mayor coste computacional, el cual no revertiría necesariamente de manera positiva en la simulación.

El *OpenHaptics Toolkit* cuenta con una *API* diseñada para una operación rápida y sencilla que se llama *QuickHaptics MicroAPI*. Está pensada para casos muy sencillos y se deben utilizar comandos provenientes de las *HLAPI* o *HDAPI* si se quiere aumentar la complejidad de la simulación.

En la ilustración anterior aparecen dos términos que no han sido presentados pero que son dignos de mención: *HHD* y *HHLRC*. *HHD* (*Haptic Device Handle*) sirve como identificador del dispositivo dentro de la simulación y *HHLRC* hace referencia al estado háptico que corresponde al dispositivo identificado por *HHD*, es decir, el estado de los motores que persiste en el tiempo que transcurre entre cada una de las iteraciones realizadas por el ciclo de servicio.

3.1.2.1.- Haptic Device API (HDAPI).

La *HDAPI* nos permite acceso de bajo nivel al sistema háptico, otorgándonos la capacidad tanto de conocer pormenores del estado actual del sistema como de manipularlos. Un ejemplo de ello es que nos permitirá modificar el *servo loop*, de tal manera que cumpla con las necesidades del usuario.

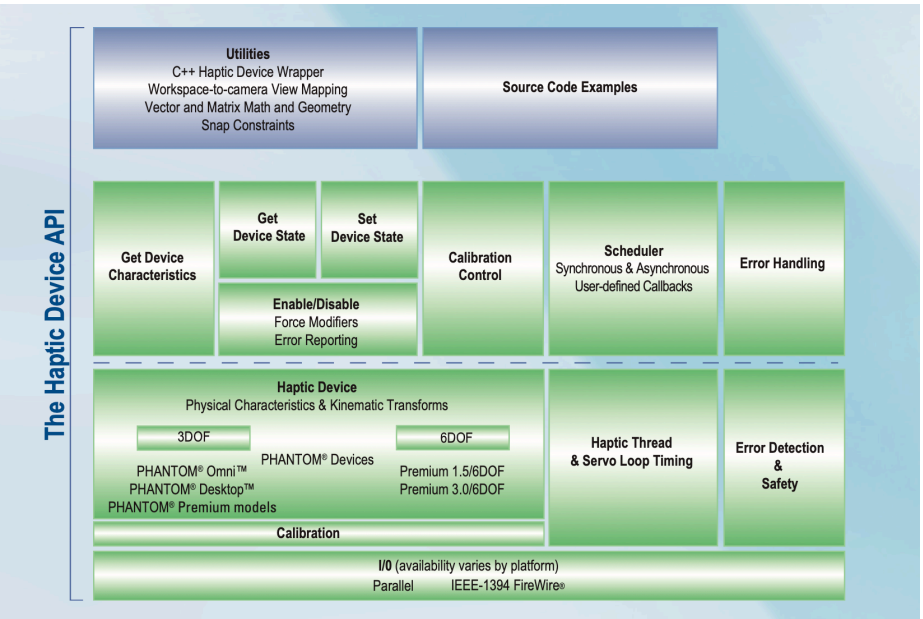

Figura 34. Contenido de la HDAPI [10].

Algunas otras funcionalidades que podremos extraer del sistema a través de la *HDAPI* son las siguientes:

Estado del sistema

Podemos conocer posición, orientación y velocidad del *stylus* respecto al eje de coordenadas cartesianas del dispositivo, los ángulos de rotación descritos por las articulaciones y las lecturas de los *encoders/DAC*, además del estado de los botones. Podremos conocer también la fuerza actual que está ejerciendo el sistema, duración del último ciclo de servicio (*servo loop*), además de una larga lista de otros datos que se detallan en la guía "*API Reference Guide*" [26].

Características del sistema

Indica modelo, versión, número de serie, dimensiones del espacio de trabajo, valores máximos de fuerza y velocidad entre otros, temperatura de los motores o capacidades de calibración. De nuevo, la lista completa se encuentra en la "*API Reference Guide*".

Modificar el estado del sistema
Renderizar fuerzas y torques (torques solo en modelos que lo permitan) en el espacio cartesiano, colocar los motores en posiciones determinadas o modificar el estado de los *LED*s del sistema.

Activación y desactivación
De fuerzas, restricción de movimiento respecto a un punto o superficie (*constraint*) o comprobaciones y limitaciones de excesos de fuerza o velocidad.

Envío de errores
Pila para errores acerca del funcionamiento del sistema, renderización de fuerzas, *scheduler*...

Calibración
Tanto manual como automática.

Scheduler
Ajustable de tal manera que se pueda conseguir ciclos de servicio determinados, llamadas (peticiones de servicios) en instantes determinados o sincronización entre los hilos del proceso háptico y gráfico de la manera determinada por el usuario.

Recursos
Adaptación de la programación del sistema a C++, generación del espacio virtual en función del espacio de trabajo del dispositivo, geometría y cálculos de vectores y matrices.

Trabajar a través de la *HDAPI* nos permite, como ya se ha comentado un control muy detallado del sistema, pudiendo obtener los valores en cada momento de la situación espacial del equipo o dar consignas concretas de fuerza.

Por ejemplo, si quisiéramos obtener la última posición del *HIP* medida (en milímetros) por el dispositivo, tendríamos que ejecutar los siguientes comandos:

```
double posicion[3];
hdGetDoublev(HD_CURRENT_POSITION, posicion);
```

Primero necesitaríamos un array de tipo *double* de al menos tres posiciones para almacenar el dato, después deberemos llamar la función *hdGetDoublev*, indicando entre paréntesis primero el parámetro que queremos conseguir, en este caso la posición y en segundo lugar la variable en la que queremos almacenarlo.

Si en lugar de extraer un dato lo que queremos es fijar una consigna, por ejemplo, queremos que el dispositivo ejerza una fuerza determinada, lo que deberíamos hacer es lo siguiente:

```
double fuerza[3] = {1,0,0};
```

Indicando que queremos que se ejerza una fuerza de 1 *N* en dirección positiva del eje X del sistema de referencia del dispositivo.

```
hdSetDoublev(HD_CURRENT_FORCE, fuerza);
```

Después utilizando la llamada *hdSetDoublev,* se consigue implementar en valores determinados por el parámetro que se coloque en primera posición dentro del paréntesis, los valores proporcionados por el segundo.

Para manipular el dispositivo háptico mediante la *HDAPI* se necesita un conocimiento previo acerca del funcionamiento del dispositivo y una idea muy concisa del resultado que quiere obtener de su programación. Los modelos deberán crearse íntegramente por el programador, y a pesar de que su uso puede combinarse con funcionalidades de la *HLAPI*, que pertenece a un más alto nivel y en consecuencia es más sencillo a la hora de programar, continúa poseyendo una complejidad considerable.

3.1.2.2 Haptic Library API (HLAPI).

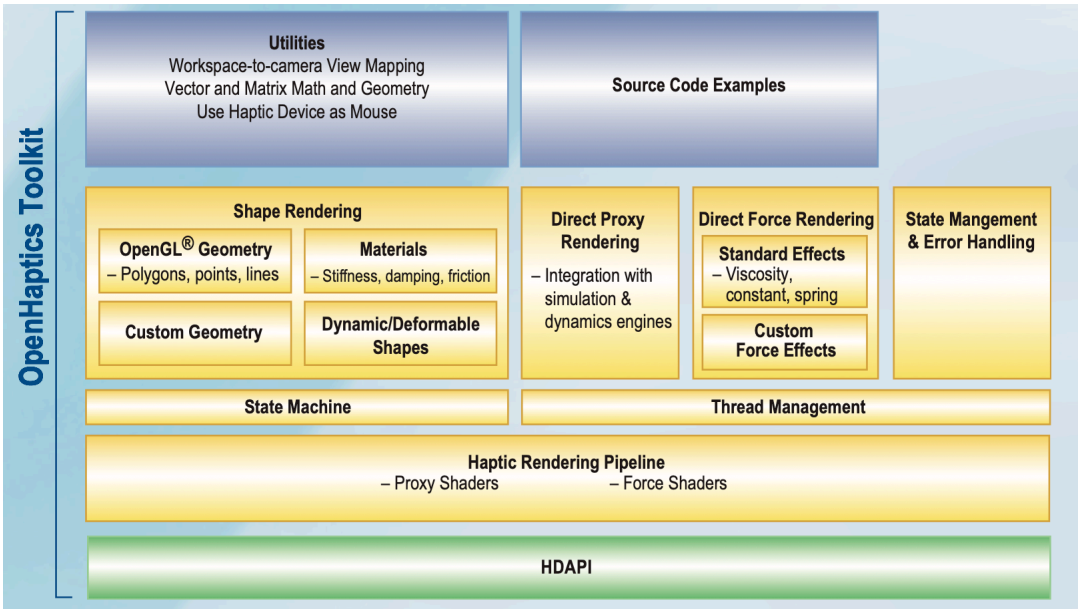

Figura 35. Desglose del contenido de la HLAPI [10].

La *HLAPI* permite la renderización háptica a mayor nivel que la *HDAPI*, habiendo sido diseñada para resultar familiar a los usuarios que posean un conocimiento de la *OpenGL API*, pero que no cuenten con la misma experiencia en el ámbito de la renderización háptica. Se podría decir que busca una mayor sencillez a cambio de la perdida de prestaciones. Cuenta con una serie de efectos hápticos definidos (como muelle o viscosidad entre otros) que permiten al dispositivo emular esos comportamientos en función de los parámetros determinados para cada efecto, pero se pierde la capacidad de introducir fuerzas en direcciones concretas con valores en newtons.

Por ejemplo, si quisiéramos introducir un efecto viscoso, dentro de la *HDAPI* tendríamos que generar un modelo matemático que relacionara la fuerza generada con la velocidad del sistema, obteniendo un modelo preciso. Sin embargo, dentro de la *HLAPI* solo tendríamos que introducir las siguientes tres líneas de código:

```
hlEffectd(HL_EFFECT_PROPERTY_GAIN, 0.4);
hlEffectd(HL_EFFECT_PROPERTY_MAGNITUDE, 0.4);
hlStartEffect(HL_EFFECT_VISCOUS);
```

De acuerdo con lo anterior se consigue una programación más sencilla, pero tanto en cuanto los valores de *Gain* (ganancia del efecto) y *Magnitude* (fuerza máxima que se puede ejercer) solo pueden tomar valores entre 0 y 1 (menor y mayor valor posibles), perdemos control sobre las fuerzas generadas en comparación con la *HDAPI*.

Presenta las siguientes funcionalidades:

Generación de formas.
Uso de primitivas de *OpenGL* (polígonos, puntos y líneas) para la generación de objetos. En *HDAPI* tendríamos que introducir uno a uno los vértices de los objetos para generar contactos.

Efectos hápticos.
Efecto muelle, viscosidad, fricción, fuerzas constantes y vibración.

Modelos de contacto.
Dos modalidades: contacto y restricción (anclaje a un punto).

Propiedades de superficies.
Fricción, valores de dureza y amortiguación superficiales en una o ambas caras de una pieza.

Dinámica y objetos deformables.
Capacidad de integración de productos creados por terceras partes.

3.1.2.3 Conclusiones.

La decisión acerca del uso de la *HDAPI* o *HLAPI* radicará en el uso que cada usuario pretenda hacer del dispositivo. La *HLAPI* está diseñada para ser fácil de utilizar a aquellas personas que tengan formación previa acerca del uso de *OpenGL* y carezcan sin embargo de experiencia en la renderización de fuerzas hápticas. Su uso también es recomendado si se quieren añadir físicas u objetos al escenario virtual, ya que su implementación será más sencilla (permite implementación de librerías con físicas, lo cual no ocurre en *HDAPI*).

Por otra parte, si lo que queremos es centrarnos en la realización de modelos que representen fielmente un modelo físico y que no involucren la integración de objetos complejos, es altamente recomendable hacerlo a partir de la *HDAPI* ya que nos otorgará mucho más control sobre el comportamiento del interfaz.

Estas dos *APIs* no son excluyentes y puede utilizarse la *HDAPI* para, tal y como se dice en la guía de programación, “aumentar” las funcionalidades de la *HLAPI*. Es decir, puede decidirse trabajar con un efecto háptico preparado dentro de la *HLAPI*, por ejemplo vibración, y utilizar en conjunto una instrucción que renderice una fuerza concreta (gravedad por ejemplo).

3.1.3.- Modificación del punto de interfaz háptico.

Las características del sistema certificadas por el fabricante aseguran un correcto funcionamiento del dispositivo y de la representación de sus movimientos en los entornos en que se utilice. Sin embargo, tal y como se ha explicado, el punto de interfaz háptico representa una posición diferente al *TCP* del brazo robótico del sistema, en otras palabras, no es coincidente con el extremo del *stylus*. Debido a que considero más intuitiva su localización en el *TCP*, realizo a continuación una propuesta de traslado del punto de interfaz háptico al *TCP* mediante software.

Por motivos de claridad, se informa de que en adelante la posición del punto de interfaz háptico original recibirá el nombre *HIP*, mientras que el modificado pasará a llamarse *TCP*.

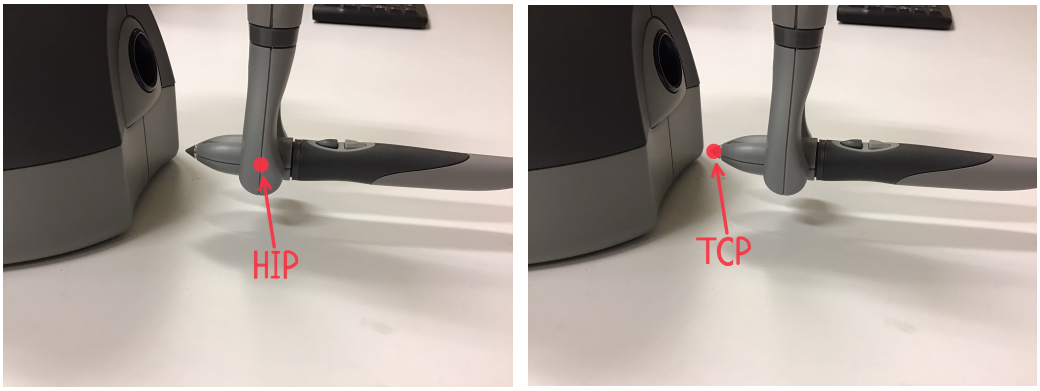

Figura 36. (Izq) Representación actual. (Dcha) Representación buscada.

3.1.3.1.- Modelo matemático.

Tanto la *HDAPI* como *HLAPI* aportan información en tiempo real acerca de la posición del *HIP*, así como de la orientación del *stylus* respecto al eje de coordenadas que se expone en el apartado 3.1.1. En consecuencia, podremos conocer la posición de *TCP* en función de la situación espacial de *HIP* una vez sean implementadas las relaciones matemáticas pertinentes. Debe conocerse para su cálculo la relación de movimientos del sistema que afectan a la posición relativa de *TCP* con respecto a *HIP*, además de la distancia que los separa, cifrada en este caso en 39 *mm*.

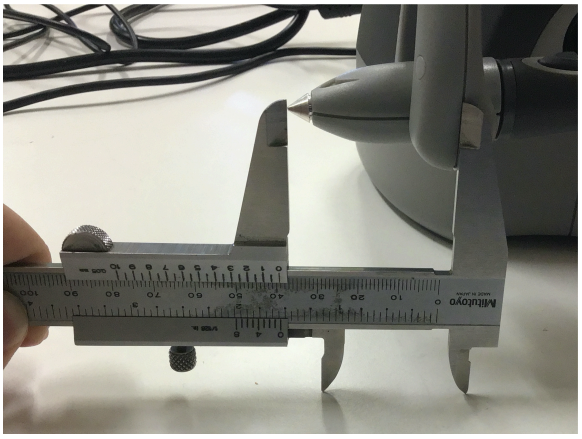

Figura 37. Medición distancia HIP-TCP.

Haciendo en todo momento referencia al eje de coordenadas del dispositivo expuesto en el apartado 3.1.1. y utilizando los milímetros como unidad de medida, se tiene que la posición en la dirección Y se verá modificada únicamente por el ángulo formado entre el plano XZ y el *stylus*, representado en la imagen siguiente por ß. Siendo $\vec{x}, \vec{y}, \vec{z}$ vectores unitarios asociados respectivamente a las direcciones X, Y y Z del anteriormente mencionado eje de coordenadas, tenemos que:

$$TCP\ \vec{y} = [HIP + 39 * sen(ß)]\vec{y}$$

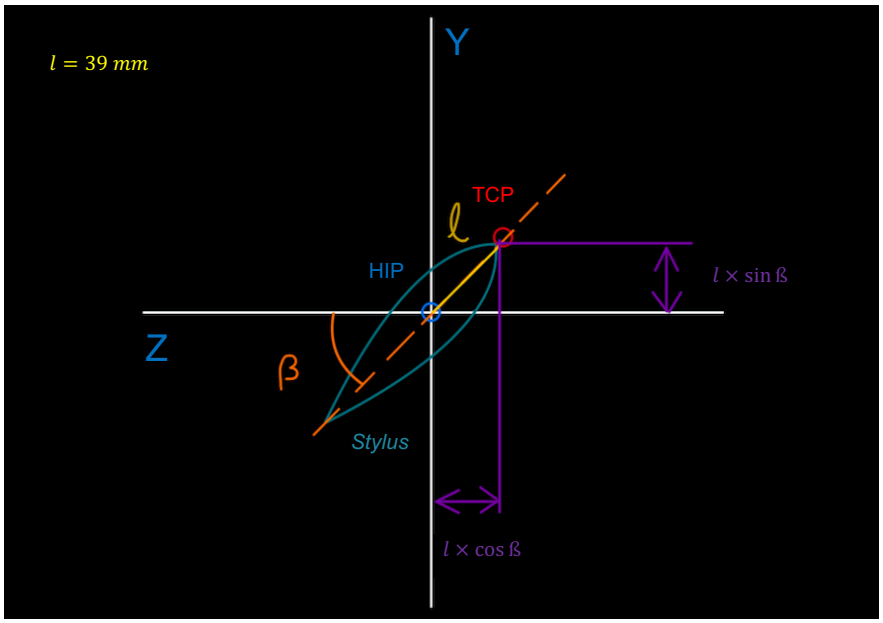

Figura 38. Rotación alrededor del eje X.

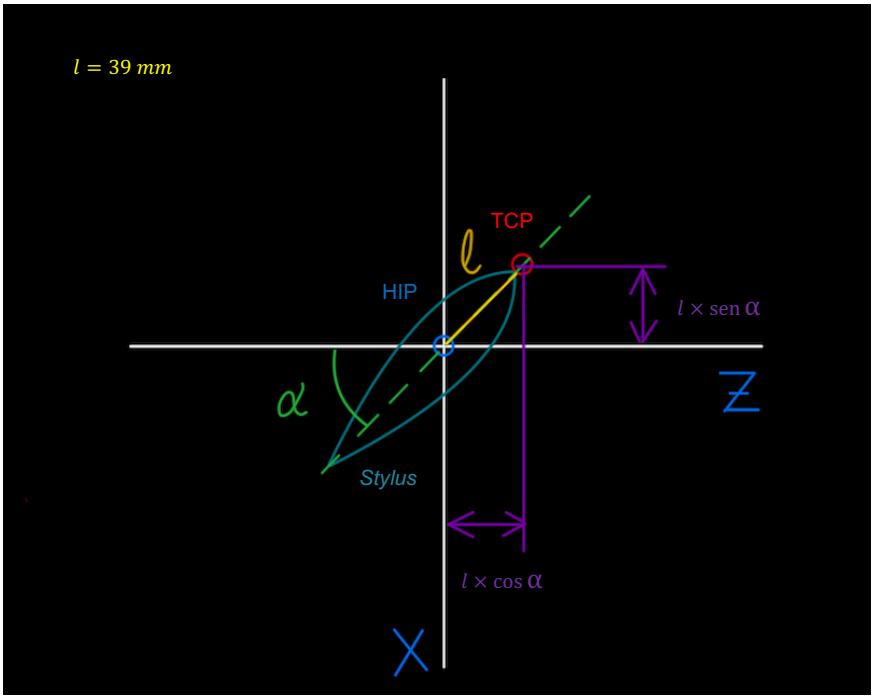

Figura 39. Rotación alrededor del eje Y.

En cuanto a las posiciones sobre los ejes X y Z, tal y como aparece en las ilustraciones anteriores, deberemos tener en cuenta tanto la orientación sobre el eje X como sobre el eje Y. Las ecuaciones quedan entonces definidas de la siguiente manera:

$$TCP\ \vec{x} = [HIP - 39 * cos(ß) * sen(\alpha)]\vec{x}$$

$$TCP\ \vec{z} = [HIP - 39 * cos(ß) * cos(\alpha)]\vec{z}$$

Una vez concretadas las ecuaciones, el siguiente paso debe ser la comprobación del funcionamiento del sistema. Para ello se ha utilizado el programa "*Unity*", un entorno de desarrollo de videojuegos en el que a través del *3D Systems Openhaptics Unity Plugin* se pueden explotar las capacidades del dispositivo háptico. Se ha diseñado un interfaz de usuario con el fin de poder monitorizar distintos parámetros de su funcionamiento. Los valores mostrados son la posición y orientación tanto del *TCP* como *HIP*, sus velocidades, el estado del botón 0 y un texto que informe acerca de la existencia de un contacto o la sujeción de algún objeto.

La comprobación ha consistido en dos partes, una primera en la que se observa el comportamiento del sistema al fijar el *TCP* en un punto y variar la orientación del *stylus*, y una segunda en la que se desplaza el *TCP* por una serie de puntos de localización conocida. Los resultados obtenidos en el primer experimento son los siguientes:

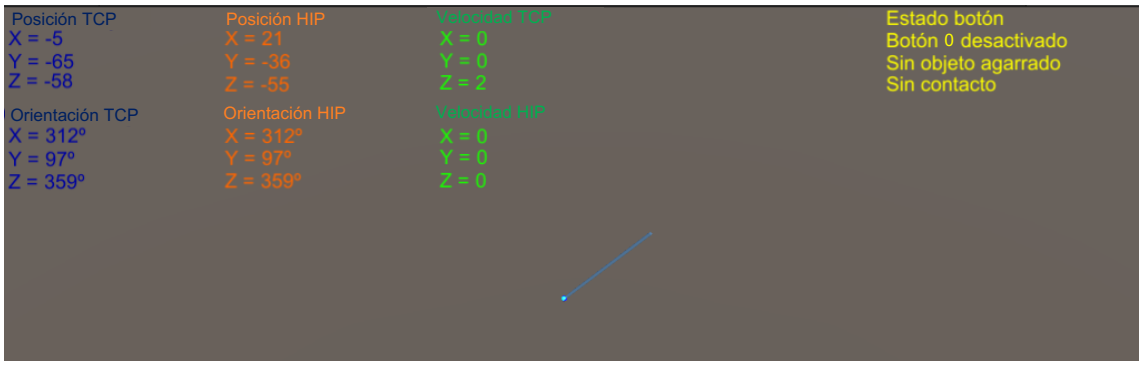

Figura 40. Posición del TCP con orientación en Y de 97º.

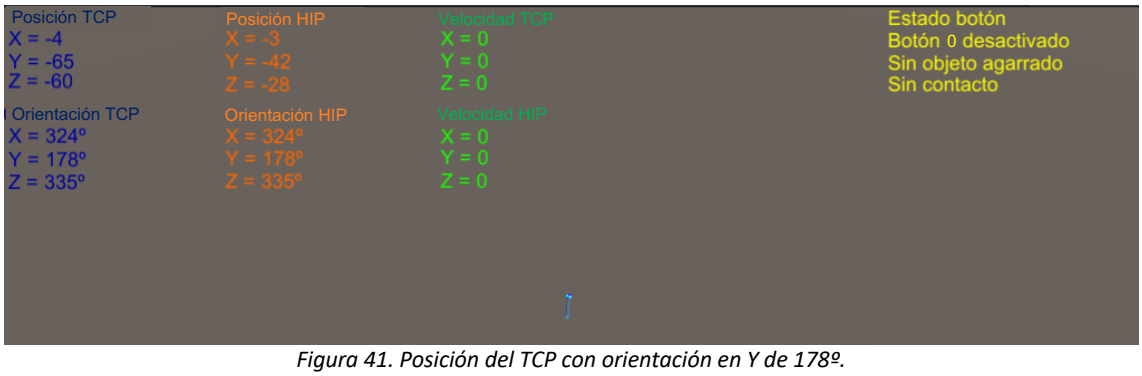

Figura 41. Posición del TCP con orientación en Y de 178º.

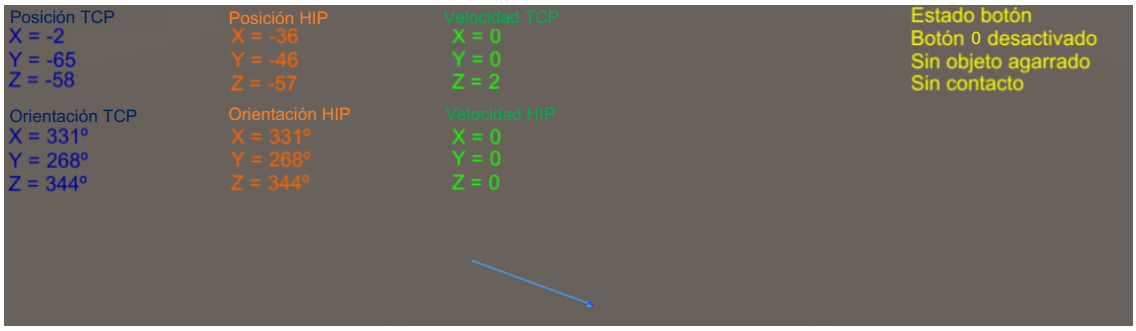

Figura 42. Posición del TCP con orientación en Y de 268º.

Como se puede observar en la parte superior izquierda de las imágenes, el valor de la posición *TCP* no varía en más de 3 *mm* a lo largo del experimento. Tomando esa variación como error se obtendría un valor relativo frente al fondo de escala menor del 1% sobre el eje X (eje en que se produce), ya que en esta dimensión el sistema arroja valores que van desde los 210 a los -220 *mm*.

Para el segundo experimento se ha dibujado un eje de coordenadas sobre una hoja de papel milimetrado, escogiéndose 5 puntos sobre los que se ha posado el TCP sin prestar especial atención a su orientación, ya que como se ha demostrado en la experiencia anterior, no debería afectar de manera relevante los valores recogidos.

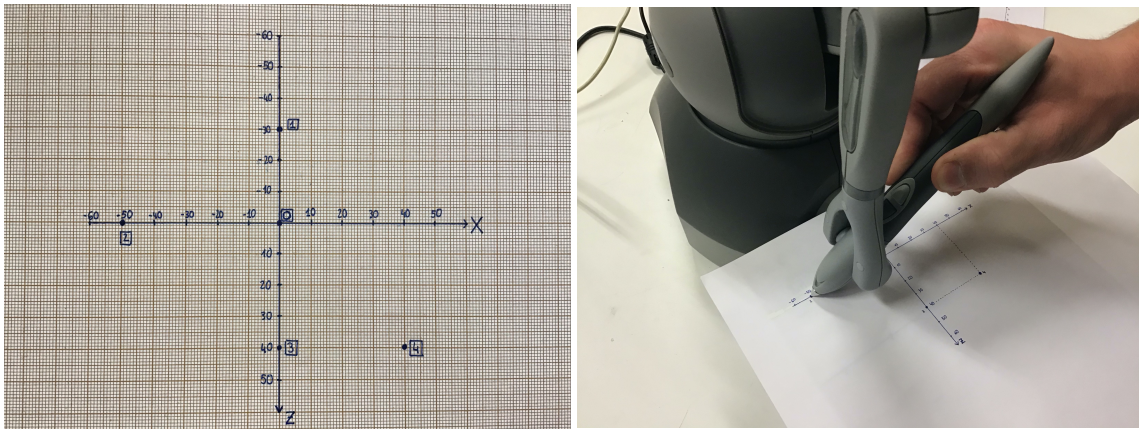

Figura 43. (Izq) Plantilla de medición. (Dcha) Toma de medidas.

Tabla 2. Mediciones obtenidas sobre los ejes X y Z.

Posición relativa a "O" (x , z)	Dato experimental respecto a “O” (x , z)
(0 , 0)	(0 , 0)
(0 , -30)	(2 , -31)
(-50 ,0)	(-47 , -3)
(0 , 40)	(-1 , 37)
(40 , 40)	(42 , 36)

El mayor error obtenido es de 4 *mm*, es decir, del 4% sobre el fondo de escala utilizada en el experimento (120 *mm* en eje X y 100 *mm* en eje Z), pero alrededor del 2% respecto al fondo de escala total del aparato. Teniendo en cuenta que los potenciómetros del sistema homologan una precisión del 5% y los posibles errores de precisión humanos a la hora de manipular el aparato, se puede afirmar que el modelo representa fielmente los movimientos del *TCP* del dispositivo.

3.1.4.- Generación de efectos hápticos.

Con la finalidad de poder utilizar algún tipo de realimentación háptica en el proceso de interconexión y teleoperación que se aborda en el presente trabajo de fin de grado, paso a diseñar una serie de escenarios en los que poder demostrar el conocimiento adquirido sobre la materia.

He utilizado como base para este programa de ejemplo los archivos situados en la carpeta *"CoulombField"* que se descargan junto con el resto del *OpenHaptics® Toolkit*. Éstos generan desde el punto de vista gráfico un sistema con dos esferas de color azul y granate, las cuales representan el punto (0,0,0) del sistema de referencia del interfaz háptico y la posición del punto de interfaz háptico respectivamente, además de un vector que representa la dirección y magnitud de la fuerza ejercida. Desde el punto de vista háptico, se realiza una simulación de las fuerzas de Coulomb en función de la distancia que separe a ambas esferas.

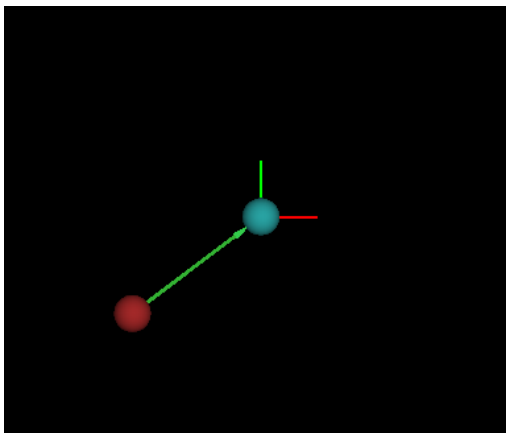

Figura 44. Visualización del programa.

Se han realizado una serie de modelos que ejercen sobre el usuario la fuerza producida por un sistema masa-muelle, masa-muelle-amortiguador y masa con doble muelle-amortiguador en paralelo. Ya que la representación visual diseñada para *CoulombField* es también adecuada para este caso, se ha decidido mantenerla.

En el inicio del programa aparecerá el siguiente mensaje, instando a los usuarios a elegir modo de funcionamiento:

```
Iniciando aplicación

Elige modo de funcionamiento:
       Pulsa 1 e intro para modo muelle.
       Pulsa 2 e intro para modo muelle-amortiguador.
       Pulsa 3 e intro para modo doble muelle-amortiguador.
```

Si no se escoge ninguno de esos tres valores se recibirá un mensaje de error y se pedirá de nuevo la introducción de uno válido hasta que se haga.

Modo masa-muelle.

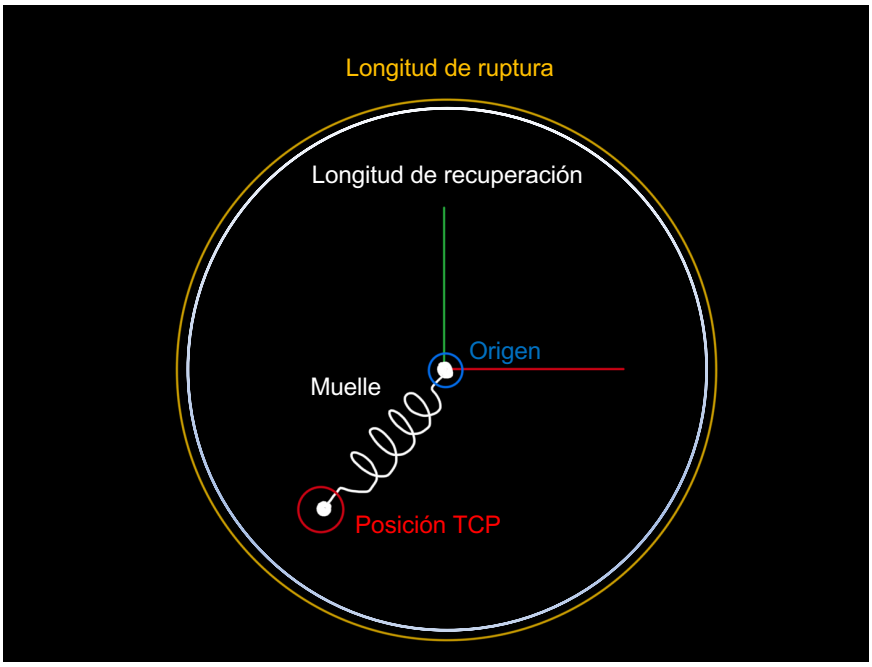

Figura 45. Esquema modelo muelle.

Este modo representa un muelle ideal de longitud cero situado sobre el origen de coordenadas (punto (0,0,0) del eje de coordenadas del dispositivo), adherido a una masa de 0,045 *Kg* (masa correspondiente al *stylus*). Si el usuario no ejerce ninguna fuerza sobre él más que aquella necesaria para contrarrestar su peso, el *TCP* se situará sobre el origen de coordenadas. En caso de que el usuario ejerza una fuerza que provoque un desplazamiento del *TCP* respecto del origen, los motores del dispositivo reaccionarán de acuerdo con la ley de elasticidad de Hooke en dirección y sentido *TCP*-Origen.

En cada una de las tres dimensiones y siendo *F* la fuerza ejercida por el interfaz, *K* la constante de elasticidad y "*x*" la posición de *TCP* respecto a origen tenemos que:

$$\vec{F} = -K \cdot \vec{x}$$

Para el cálculo de la posición de *TCP* se ha utilizado el método presentado en el apartado 3.1.3, introduciendo en el programa las relaciones matemáticas en él establecidas.

Se implementa además un efecto por el cual, en caso de que dicha separación supere una longitud definida al inicio del programa por el usuario, el dispositivo ejercerá una fuerza mucho menor (100 veces menor) a la que correspondería en ese mismo punto según la ley anteriormente mencionada, simulando la rotura del muelle. Una vez se retorne a la zona delimitada, el efecto muelle volverá a funcionar con normalidad.

El usuario deberá aportar entonces dos valores en el inicio de la simulación: uno que indique la longitud máxima del muelle en milímetros (`double limite`) y otro que determine la constante elástica del muelle (`double k`), de manera que pueda obtener la simulación que desee. En ambos casos, si se introduce un valor negativo, se mostrará un mensaje de error explicativo y se pedirá su nueva introducción.

La frontera entre ambas zonas supone un problema, ya que se producen variaciones de importante calado en la fuerza ejercida, que provocan un funcionamiento irregular (vibraciones severas) al tratar de transitar dicha frontera. Para solucionar esta situación se ha optado por hacer 8 *mm* mayor la longitud de ruptura respecto a la de restauración del muelle, lo cual supone una distancia lo suficientemente grande como para evitar las vibraciones producidas anteriormente, pero no tanto como para resultar perceptible en el uso del interfaz háptico.

De esta manera se consigue un funcionamiento suave y que dentro de la zona nominal de funcionamiento, se corresponde con un muelle real hasta que se alcanzan fuerzas de 3,3 *N*, valor máximo que puede ejercer el dispositivo *Touch*.

Modo masa-muelle-amortiguador.

El funcionamiento de este modo es similar al anterior, representando un sistema masa-muelle-amortiguador ideal y de longitud cero colocado en el punto (0,0,0) del dispositivo, que se estira en cualquier dirección en función de los movimientos que realice el *TCP*. En este caso se sigue considerando la masa adherida al extremo del muelle-amortiguador aquella producida por el *stylus*.

Siendo *F* la fuerza ejercida por el interfaz, *K* la constante de elasticidad, *b* la de amortiguación y "*x*" la posición de *TCP* respecto a origen, tenemos que el modelo utilizado para este apartado es el siguiente:

$$\vec{F} = (-K \cdot \vec{x}) + \left(-b \cdot \frac{\partial \vec{x}}{\partial t}\right)$$

No se ha concebido un límite de elongación para este modo por lo que solo se pedirán por pantalla al usuario los valores de las constantes elásticas (*K*) y de amortiguación (*b*). Se deberá en ambos casos introducir un valor mayor que cero y en el caso de la constante de amortiguación, menor que 3,5.

El límite establecido sobre *b* se debe a que si se escoge un valor mayor que 3,5 el dispositivo comienza a vibrar al desplazarse. Si se continúa aumentando ese valor, la vibración se torna en oscilación y el sistema se torna en inservible. En caso de que se introduzca cualquier valor no válido, se mostrará un mensaje por pantalla indicando el error y se pedirá un nuevo valor hasta que se subsane el problema.

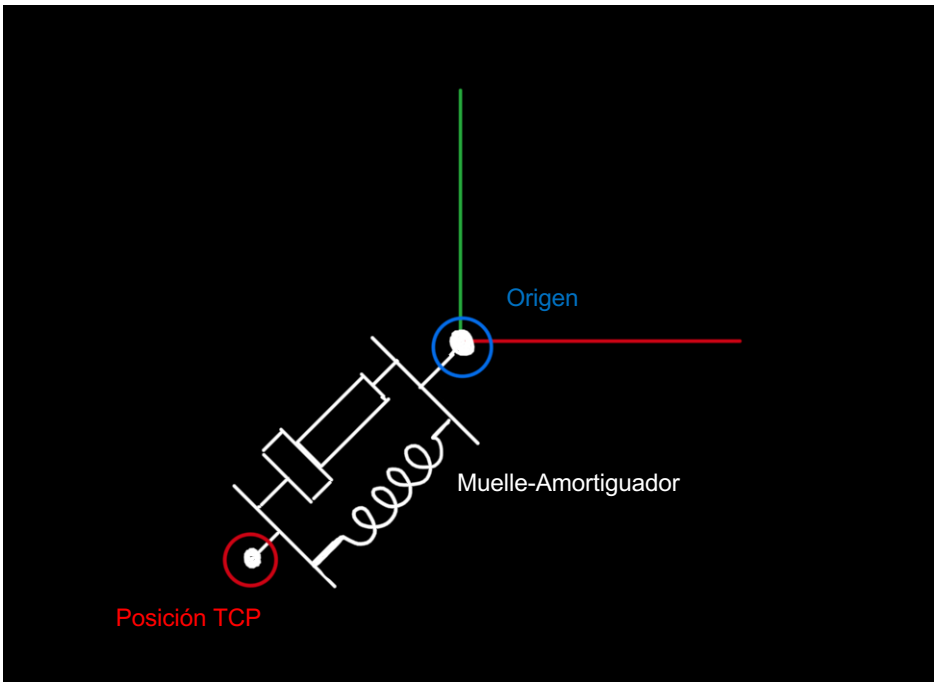

Figura 46. Imagen del modelo muelle-amortiguador.

Como en cualquier sistema muelle-amortiguador, al introducir distintos valores en las constantes, se pueden observar tres clases de comportamiento: subamortiguado, crítico y sobreamortiguado.

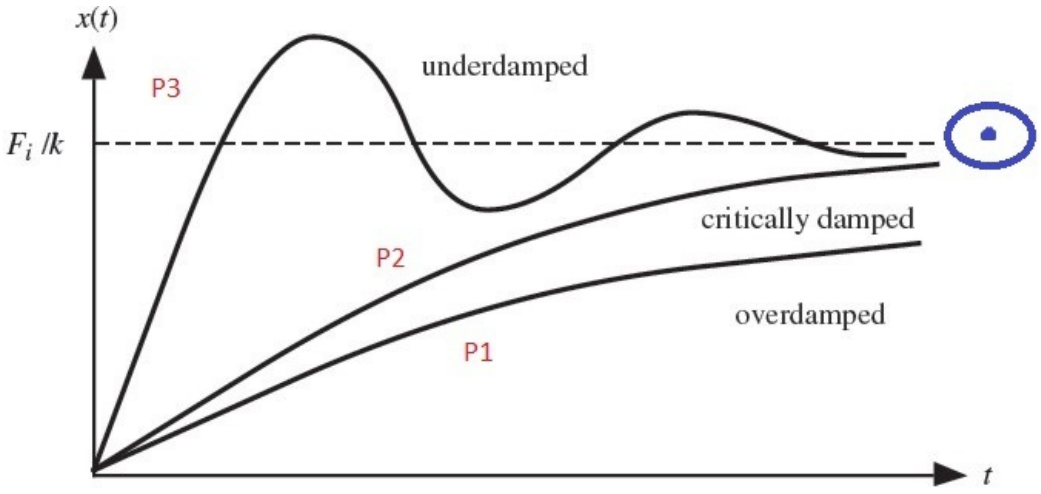

Figura 47. Comportamiento de la posición de sistemas en función de su tipo de amortiguación [27].

Podremos determinar el comportamiento del sistema diseñado atendiendo a la siguiente ecuación:

$$b = 2\sqrt{k \cdot m}$$

Si el valor de *b* cumple con lo establecido tendremos un sistema crítico, si es menor será subamortiguado y sobreamortiguado si es mayor.

Modo doble masa- muelle-amortiguador.

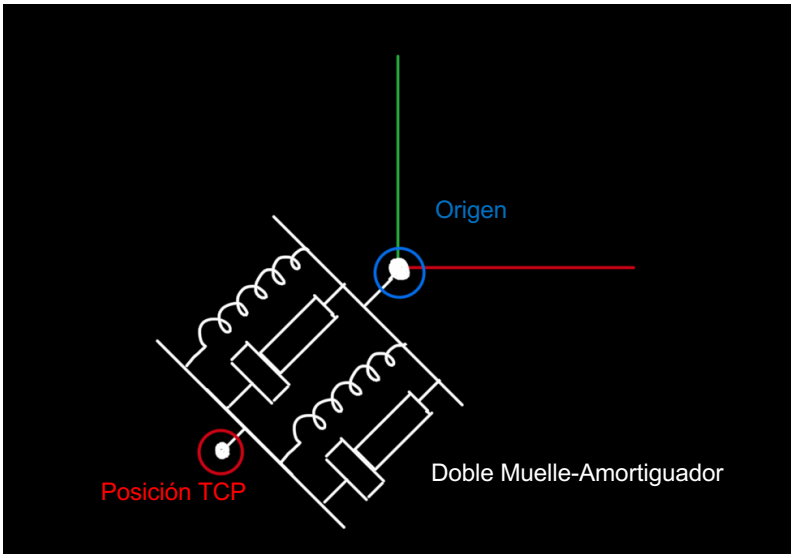

Figura 48. Imagen del modelo doble muelle-amortiguador.

El modo de funcionamiento es prácticamente igual al del caso anterior. En este caso se pedirán al usuario dos valores de constante elástica y de amortiguación, siendo el sumatorio de las dos constantes de amortiguación el valor que no pueda superar los 3,5.

Los valores introducidos por el usuario deberán cumplir los siguientes requisitos:

- K_1 deberá ser mayor que 0.
- B_1 deberá ser mayor que 0 y menor que 3,5.
- K_2 deberá ser mayor que 0.
- B_2 deberá ser mayor que 0 y menor que 3,5.
- El sumatorio de B_1 y B_2 deberá ser menor que 3,5.

En caso de que alguna de estas condiciones no se cumpla, se mostrará el mensaje de error correspondiente y se deberá introducir un nuevo valor.

La ecuación que representa la generación de fuerzas en este apartado será:

$$\vec{F} = (-k_1 \cdot \vec{x}) + \left(-b_1 \cdot \frac{\partial \vec{x}}{\partial t}\right) + (-k_2 \cdot \vec{x}) + \left(-b_2 \cdot \frac{\partial \vec{x}}{\partial t}\right)$$

3.2.- Robot colaborativo.

3.2.1.- Hardware.

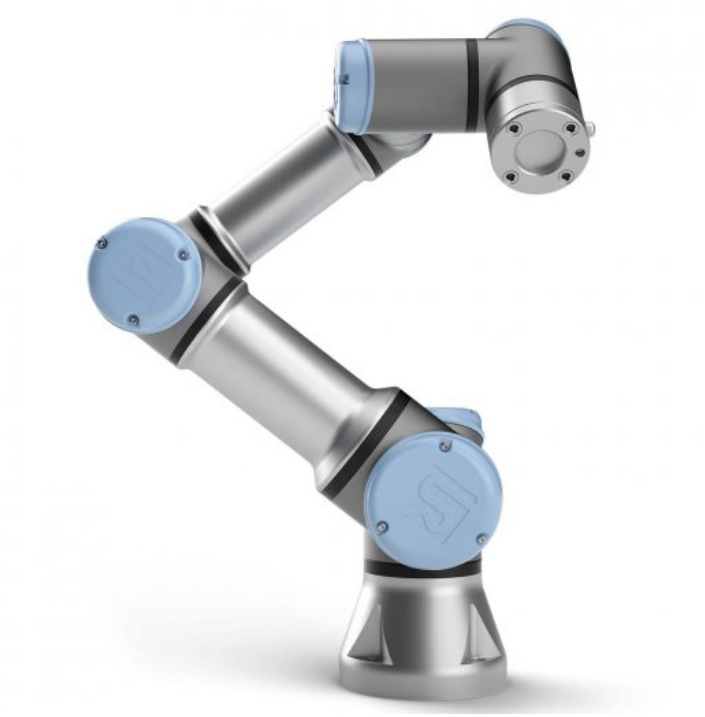

Figura 49. Robot colaborativo UR3.

El brazo robótico utilizado, es el anteriormente mencionado UR3 de la empresa Universal Robots. Cuenta con seis grados de libertad y está fabricado con materiales ligeros, en concreto plástico (polipropileno) y aluminio.

Sus características principales son las siguientes [20]:

- Carga máxima de 3 *Kg*.
- Alcance de 0,5 metros.
- Rango de giro de 360º en todas sus articulaciones, a excepción de la final que no cuenta con limitación.
- Repetibilidad de $\pm$ 0,1 *mm*.
- Velocidad lineal máxima de 1000 *mm/s*.
- Funcionamiento colaborativo con 5 modos de seguridad avanzados.
- Límite de fuerza situado en 150 *N* por defecto, que puede ser rebajado hasta los 50 *N*.
- Diseño modular que permite el cambio de las articulaciones en un tiempo reducido.
- Lazo de control con una frecuencia de 125 *Hz*.

Su corto alcance, reducida carga máxima y elevada movilidad, hacen ideal el uso de este robot en espacios de pequeñas dimensiones en los que se deban manipular objetos ligeros.

En cuanto a sus capacidades colaborativas, el brazo robótico recibe información de la fuerza ejercida por los motores a través de la corriente eléctrica suministrada a cada uno de ellos. La caja de control estima, en función de la posición, configuración de las articulaciones y masa del objeto que se esté o no sujetando, la fuerza que debería estar ejerciendo cada uno de ellos. Si en algún momento se produce una discrepancia entre el valor calculado y el obtenido, el robot identifica la situación y actuará de la manera establecida por su programación.

Según se indica en [28], la sensibilidad de este control de fuerza se sitúa en los 5 *N*, pudiéndose alcanzar valores de 150 *N*. Es posible establecer una relación entre este valor y la fuerza máxima ejercida por el dispositivo *Touch*, a fin de lograr realimentación háptica de los movimientos producidos por el robot. Sin embargo, el umbral definido es demasiado alto para muchas aplicaciones, por lo que puede volverse necesaria la utilización de sensores de fuerza con mejores características.

3.2.2.- Software.

Existen dos maneras de controlar el robot *UR3* de forma nativa: sirviéndonos de *PolyScope* y las herramientas que alberga o programando mediante *script*, puesto que el controlador de bajo nivel del robot *URControl* puede soportar la conexión con ámbas.

3.2.2.1.- PolyScope.

Es un interfaz de usuario diseñado para el manejo de los productos fabricados por Universal Robots, con la intención de generar un espacio que permita una fácil manipulación de los mismos sin necesidad de poseer profundos conocimientos de programación o robótica.
Las distintas herramientas con las que cuenta *PolyScope* se agrupan en *tabs*: pantallas a través de las cuales podremos programar los movimientos del robot, pantalla "*Program*"; desplazarlo de forma directa, pantalla "*Move*"; o monitorizar las señales de entrada y salida, *I/O Tab* entre otras actividades.

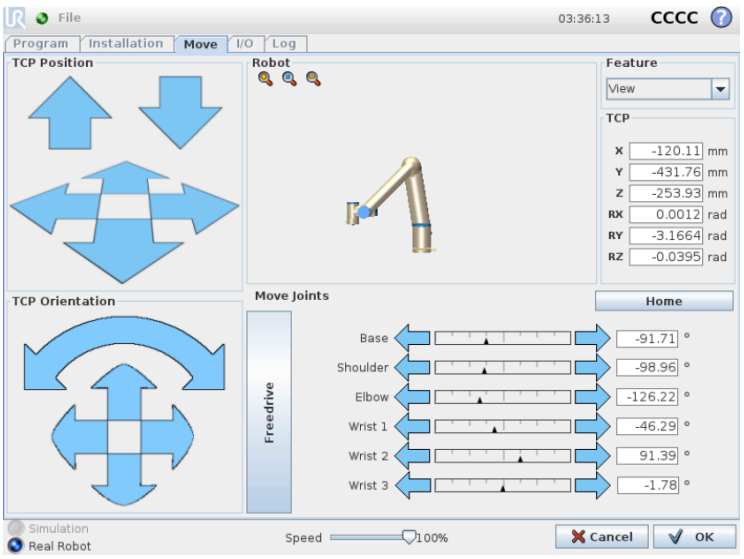

Figura 50. Pantalla "Move Tab" de PolyScope [29].

El lenguaje de programación propio de los sistemas fabricados por *Universal Robots* se llama *URScript*, el cual cuenta con los mecanismos necesarios para controlar y monitorizar los desplazamientos del sistema, así como las señales recibidas y enviadas a través de los puertos de entrada y salida.

Cuenta además con un simulador (*URSim*), que si bien es un tanto pobre en cuanto a la información que ofrece en comparación con otras herramientas de simulación, es la que ofrece una mayor correlación entre la simulación y el robot real para estos sistemas. *PolyScope* puede ejecutarse tanto desde la pantalla presente en el armario de control del robot, como desde un ordenador que cuente con Linux como sistema operativo, ya que la programación de la propia caja de control se basa en una distribución de este tipo.

3.2.2.2.- Control mediante script.

Este control consiste en el desarrollo de código escrito en el lenguaje *URScript*. Dicho programa se ejecutará en un ordenador que actúa como cliente dentro de un proceso de comunicación mediante *sockets* siguiendo el protocolo *TCP/IP*, en el que el controlador de bajo nivel del robot (*URControl*) actúa como servidor.

Para la realización de las actividades relatadas en el presente trabajo de fin de grado relacionadas con el control del robot *UR3*, se opta por este mecanismo a causa de su mayor versatilidad. En concreto se utilizará un sistema de control desarrollado en *ROS Kinetic Kame*.

La estructura basada en nodos de *ROS,* implica la posibilidad de reutilizar todo aquello realizado durante la elaboración de este trabajo, modificarlo o integrarlo en aplicaciones desarrolladas en un futuro. Además, la naturaleza “gratis y de código abierto” que representa supone la existencia de un gran número de herramientas (librerías y paquetes) desarrollados por su comunidad de usuarios y dirigidos al control de elementos robóticos, lo cual facilitará enormemente la tarea a realizar. Alguna de estas herramientas son *MoveIt*, *tf2*, *Gazebo o RViz.*

MoveIt

Es un entorno de trabajo de *ROS* utilizado para la manipulación de sistemas robóticos. Se ha desarrollado un gran número de paquetes compatibles con el *framework* dirigidos a la representación de distintos robots, entre los que se encuentran la gama IRB de la empresa ABB, KR de KUKA o UR de Universal Robots entre otros tantos.

El lenguaje utilizado es *C++* aunque pueden ejecutarse ciertas funciones básicas mediante el uso de *Python*. Para incrementar las funcionalidades que ofrece *MoveIt,* pueden descargarse una serie de *plugins* que complementen sus capacidades. Algunos de uso común son *OMPL* (*Open Motion Planning Library*), utilizado como planificador de movimientos; *KDL* (*Kinematics and Dynamics Library)*, librería empleada para el trabajo con cadenas cinemáticas o *FDL* (*Fast Collision Library*), librería empleada en la detección de colisiones.

La *API* de *MoveIt* puede dividirse en cuatro bloques dependiendo de la complejidad que aparejen las funciones en ellos comprendidos:

- *Move Group Interface*: abarca las funcionalidades principales de *MoveIt*, a saber, movimientos en coordenadas cartesianas y articulares, control de velocidad y aceleración en dichos movimientos además de la recogida y posicionamiento de objetos.
- *MoveIt Core*: permite el control al nivel más bajo al que se permite el acceso.
- *MoveIt ROS*: incluye un mejor control de la planificación de movimientos, permite la relación con otros elementos de *ROS* y suma la facultad de percepción al robot.
- *MoveIt OMPL Interface*: en el caso de que se encuentre implementado el *plugin* OMPL, se adecúa la planificación de trayectorias.

tf2

Es un paquete de *ROS* creado para la administración de sistemas de coordenadas, capaz de gestionar los distintos sistemas asociados a un robot. Con él existe la posibilidad de transformar puntos o vectores asociados a un sistema a otro en cualquier momento de la ejecución del programa.

Gazebo

Es un simulador de robótica en 3D de código abierto, utilizado de forma nativa en ROS.

Rviz

Es el visualizador de objetos en tres dimensiones nativo de *ROS*. A partir de la información generada por un *software* de simulación (por ejemplo *Gazebo*), se puede observar su movimiento en tiempo real, así como una serie de datos que se consideren de interés como por ejemplo la posición y orientación del TCP respecto a un eje de coordinadas fijado.

Para las simulaciones que llevemos a cabo en el desarrollo del mecanismo de interacción, se utilizará *Rviz* como visualizador y *MoveIt* como simulador del comportamiento del robot.

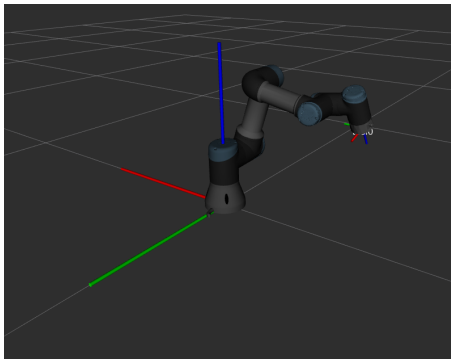

Ilustración 51. Visualización del robot colaborativo UR3 en Rviz.

3.2.2.3.- Drivers UR3.

Si queremos utilizar *MoveIt* y *ROS* para determinar los movimientos del robot colaborativo, se deberán instalar un *driver* que lleve a cabo las tareas de conexión entre el ordenador y el controlador del robot, además de un paquete de configuración *MoveIt*. Éste último deberá almacenar información relativa al modelo cinemático del robot, a sus posiciones predeterminadas, efectores finales, grupos de planificación, configuraciones que puedan provocar colisiones del robot consigo mismo o sensores (si es que cuenta con ellos).

Se han utilizado para ello dos paquetes que pueden utilizarse tanto como *driver* como paquete de configuración *MoveIt*:

- *universal_robot*: contiene la descripción completa del robot, pudiéndose realizar cálculos de cinemática directa e inversa, actúa como un *driver* básico y almacena un paquete de configuración *Gazebo* además de un modelo 3D para la visualización y cálculo de colisiones.
- *ur_modern_driver*: es una versión mejorada del *driver* incluido en *universal_robot.* Incorpora nuevas funcionalidades y resuelve problemas de estabilidad.

4.- Mecanismos de conexión y control.

En este cuarto capítulo, se realiza la descripción completa del sistema diseñado para permitir la interacción entre un dispositivo háptico *3D Systems Touch* y un robot colaborativo *UR3*. A lo largo de la exposición, se presentan los sistemas de control propuestos y su evolución de manera secuencial, comenzando por los diseños preliminares hasta alcanzar la solución final.

4.1.- Introducción

Para la realización de esta tarea, existen una serie de decisiones de diseño que se deben abordar, a saber:

- Sistema operativo en el que ejecutar los *Phantom Device Drivers (PDD)*.
- Mecanismo de transmisión de información entre sistemas.
- Modelo de control.
- Utilización de la realimentación háptica del interfaz 3D Systems Touch.

En lo concerniente a los *drivers* del interfaz háptico *3D Systems Touch*, señalar que a lo largo de la fase de estudio que acompaña a la redacción del apartado 3.1.2, se utilizaron aquellos pertenecientes al *OpenHaptics Toolki*t versión 3.5.0 para *Windows*. Sin embargo, una vez determinado que el control del robot *UR3* se realizará mediante la utilización de *ROS,* considero conveniente analizar la posibilidad llevar a cabo el control y monitorización del dispositivo háptico también mediante el uso de *ROS*, simplificando así el mecanismo de transmisión de información entre sistemas.

Con este fin se utiliza el repositorio "*3D Systems Geomagic Touch ROS Driver*", capaz de establecer conexión con el dispositivo y crear una serie de mensajes de *ROS* en los que se incluye información acerca de su estado (posición de HIP, articular, estado de los botones…). Se usará además una versión del *OpenHaptics Toolkit* para *Linux*, la cual debería permitir mantener los procesos necesarios para la utilización del interfaz háptico en este sistema operativo.

Respecto a los mecanismos de transmisión de información, éstos dependerán por completo del sistema operativo en que se ejecuten los *Phantom Device Drivers.* Si se utiliza la versión Linux, junto con el "*3D Systems Geomagic Touch ROS Driver*", podrán utilizarse los elementos de comunicación nativos de *ROS* (mensajes y servicios). Sin embargo, si se decide continuar con el trabajo realizado previamente en *Windows,* se deberá utilizar una conexión entre los dos sistemas operativos mediante el mecanismo de los *sockets.*

Se deberá determinar también en que forma los movimientos del dispositivo *Touch* se reproducirán en el robot colaborativo, o cómo estos modelos de control pueden alternarse dentro de un mismo programa a fin de encontrar la mejor solución para distintas situaciones.

Existen distintas posibilidades como realizar un control utilizando el *stylus* como un *joystick*, teniendo en cuenta que ambos sistemas comparten el mismo número de grados de libertad, asimilar los movimientos de las articulaciones del interfaz a las

del robot colaborativo, o que el movimiento del *TCP* del robot imite los del *TCP* o *HIP* del interfaz háptico.

Se deberá establecer además en qué manera la interacción de los dos sistemas revertirá en la renderización de una respuesta háptica por parte del dispositivo *Touch*, valorando la viabilidad de la utilización de información relativa a colisiones generada por el robot, o en caso de que no sea posible, idear algún otro mecanismo del que el operario pueda hacer uso como fuente adicional de información.

Por último, indicar que los sistemas de coordenadas utilizados en los sucesivos apartados serán, en el caso del interfaz háptico *Touch* el descrito en el apartado 3.1.1, mientras que para el robot colaborativo UR3 se empleará un sistema llamado "*world*" situado en la base del robot.

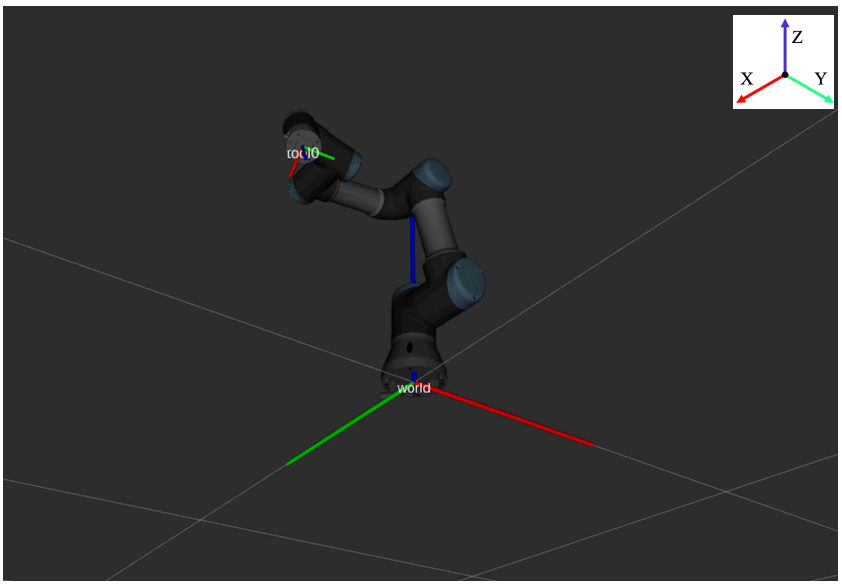

Figura 52. Ejes de referencia del robot colaborativo UR3.

El robot colaborativo contará además con otro sistema de referencia solidario con su *tool center point* que recibe el nombre *tool0.*

4.2.- Sistema de control e interconexión en *ROS*.

En este apartado se discute la viabilidad de la ejecución de los sistemas de control e interconexión de manera íntegra en *ROS*. Primero se analizan las capacidades y comportamiento del *OpenHaptics Toolkit 3.4.0* para *Linux* así como del *3D Systems Geomagic Touch ROS Driver*, para posteriormente desarrollar, (a modo de testeo de los sistemas), una aplicación que permita el gobierno de la tortuga del nodo de ejemplo de *ROS "turtlesim_node",* mediante las acciones ejercidas por el interfaz háptico.

Finalmente, y tomando como punto de partida el código utilizado para el control de "*turtlesim_node*", se planteará el desarrollo del o los nodos necesarios para la interacción con el robot colaborativo.

4.2.1.- *OpenHaptics Toolkit 3.4.0* para *Linux.*

La versión disponible para *Linux* de los *drivers* del interfaz háptico *Touch,* es anterior a la que se había utilizado anteriormente en *Windows*, en consecuencia, al iniciar la aplicación *Touch Setup* con la intención de comprobar el correcto funcionamiento del dispositivo, se observa que se ha establecido conexión con un dispositivo háptico, pero éste no se corresponde con el modelo que está siendo utilizado.

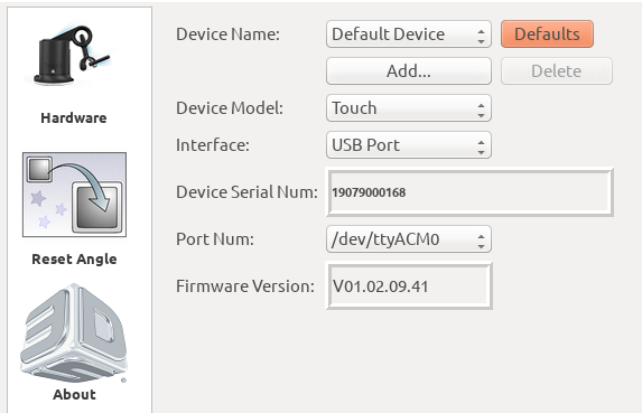

Figura 53. Pantalla Touch Setup al conectar interfaz háptico.

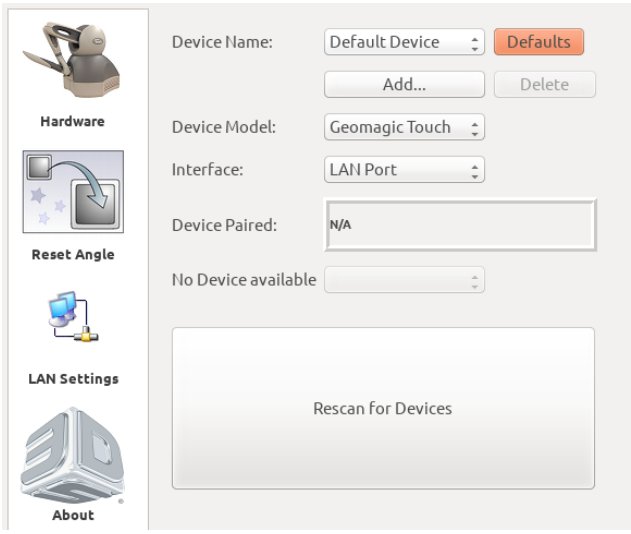

Figura 54. Pantalla Touch Setup para interfaz háptico Geomagic Touch.

Esto se debe a que el dispositivo *3DSystems Touch* realiza su conexión con el ordenador vía puerto USB, mientras que el mismo modelo a fecha de lanzamiento del presente *Toolkit,* contaba con cable *Ethernet.* Entonces, los *drivers* asumen que el dispositivo con el que se están comunicando es otro modelo que en esas fechas sí contaba con conexión mediante USB: el *Touch 3D Stylus.*

Los efectos causados por esta falta de coordinación son también visibles al ejecutar la aplicación *Touch Diagnostics*, en la se puede observar cómo los movimientos realizados sobre la articulación J1 se interpretan en sentido contrario del giro y los de las articulaciones J4 y J5 son representadas en orden inverso.

Figura 55. Representación en Touch Diagnostics de la posición del interfaz háptico.

Sin embargo, y a pesar de las incidencias relatadas, se podría lograr una representación adecuada de los movimientos: no se reproduce ninguna acción que no se haya realizado sobre el dispositivo, y las mencionadas disparidades de comportamiento podrían subsanarse en *ROS* con la programación adecuada.

4.2.2.- *3D Systems Geomagic Touch ROS Driver*.

Es un repositorio de *ROS* creado por Francisco Suárez Ruiz y desarrollado posteriormente por el "Grupo de Robótica y Máquinas Inteligentes" de la Universidad Politécnica de Madrid, diseñado para la utilización en *ROS* del dispositivo *Geomagic Touch* versión USB.

Cuenta con un nodo principal llamado *omni_state* que se ocupa de las tareas de comunicación con el dispositivo, y de poner al servicio de otros nodos una serie de *topics* con los que pueden obtener información relativa al estado del dispositivo (posición y velocidad del HIP, estado de los botones o posición de las articulaciones). Es también suscriptor de un *topic* a través del cual se podrán enviar datos de cara a la implementación de realimentación háptica.

Durante la ejecución del mencionado nodo, se observa que un número de funciones no se encuentran disponibles:

- Los datos relativos a la posición del HIP se encuentran anclados a un punto, y su velocidad, en consecuencia, es cero.
- Los valores aportados por los *encoders* de las articulaciones J1, J2 y J3 revelan un valor 0 permanente.
- La introducción de consignas para la renderización háptica no surge ningún efecto sobre el dispositivo.

Sin embargo, los potenciómetros situados en las articulaciones J4, J5 y J6 sí que aportan información válida y su uso junto con los botones situados en el *stylus* pueden proporcionarnos datos suficientes para acometer la tarea de interoperación.

4.2.3.- Interacción interfaz háptico - *turtlesim*.

Como se indicaba anteriormente, se plantea el desarrollo de un mecanismo que permita el control de las actividades del nodo *turtlesim_node,* a fin de establecer una base que permita abordar con mayores garantías el desarrollo del sistema de interacción con el robot *UR3*.

Para la realización de esta tarea se propone la utilización de tres nodos. El primero de ellos (*omni_state*) se encargará de realizar las tareas descritas en el apartado 4.2.2, mientras que *turtlesim_node* llevará a cabo las relatadas en el 2.2.3. El nodo situado entre ellos (*nodo_medio)* recibirá la información proveniente de *omni_state,* tratándola de tal manera que pueda convertirse en valores coherentes para su uso por parte de *turtlesim_node.*

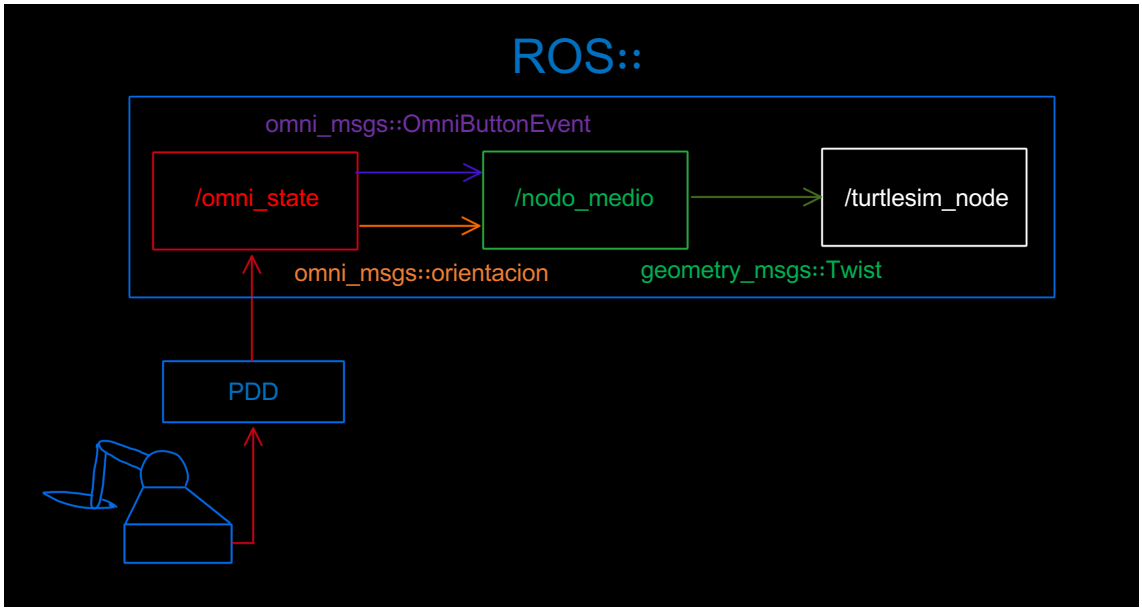

Figura 56. Esquema de interrelación de procesos.

Los procesos de comunicación se llevarán a cabo mediante el uso de tres mensajes de ROS:

omni_msgs::orientacion

geometry_msgs/Vector3 linear	*float 64 x* *float 64 y* *float 64 z*

Se utiliza para enviar los valores correspondientes a la posición angular (en radianes) de las articulaciones J4, J5 y J6. En concreto "*x*" almacenará el valor de J4, "*y*" el de J5 y "*z*" el de J6.

omni_msgs::OmniButtonEvent

int32 grey _button
int32 white_button

Activado por evento. Si el valor de uno de los botones es modificado, se envía un mensaje con el valor actual de ambos botones. Al botón 0 le corresponde la variable *grey_button* y al botón 1 la restante.

omni_msgs::twist

geometry_msgs/Twist.msg	*geometry_msgs/Vector3 linear*	*float 64 x* *float 64 y* *float 64 z*
	geometry_msgs/Vector3 angular	*float 64 x* *float 64 y* *float 64 z*

De este mensaje se utilizarán únicamente los valores "*x*" e "*y*" del vector *linear* y el valor "*z*" del vector *angular.* Los dos primeros determinarán la velocidad de avance de la tortuga, mientras que el último será utilizado para determinar su velocidad angular.

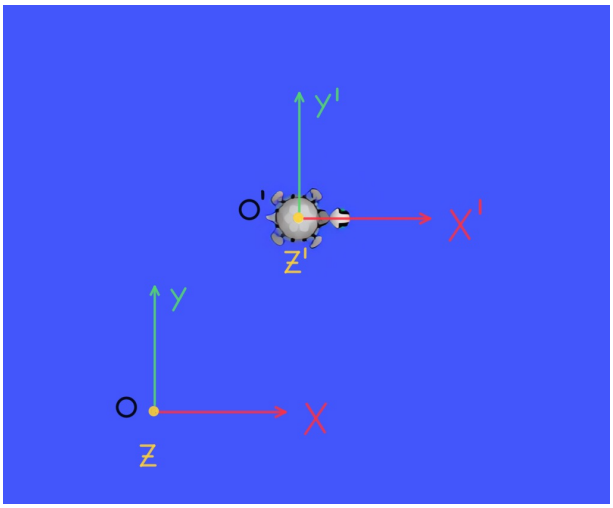

Figura 57. Eje de referencia de turtlesim.

Los ejes de coordenadas que son mostrados en la imagen son los utilizados en lo referente a los movimientos realizados en *turtlesim*. El eje *O* es fijo y las flechas indican el sentido positivo de sus direcciones. Por otra parte, *O'* solidario a la tortuga, situándose en el centro del caparazón el origen de coordenadas y apuntando siempre el sentido positivo del eje *X'* a la cabeza del animal.

En cuanto al mecanismo de control, se realizará un control por velocidad utilizando el movimiento de las tres últimas articulaciones del brazo articulado del interfaz para determinar la dirección del movimiento de la tortuga, sirviendo la pulsación del botón 1 como confirmación del movimiento.

Mientras se produce un giro mayor de 60º en sentido antihorario sobre la articulación J4 y el botón 1 está siendo pulsado, el valor *x* del vector *linear* tomará el valor 0.4, provocando el desplazamiento de la tortuga en el sentido positivo de la dirección de *X'*. Dándose la misma situación, pero ejecutándose un giro en sentido horario sobre J4, *x* pasará a tener valor -0.4 y el desplazamiento se realizará bajo las mismas condiciones, pero en sentido opuesto. En caso de que el botón 1 no esté siendo pulsado o de que no se esté realizando un giro mayor de 60º en cualquiera de los sentidos, *x* mantendrá un valor 0 y la tortuga no se desplazará en la dirección *X'*.

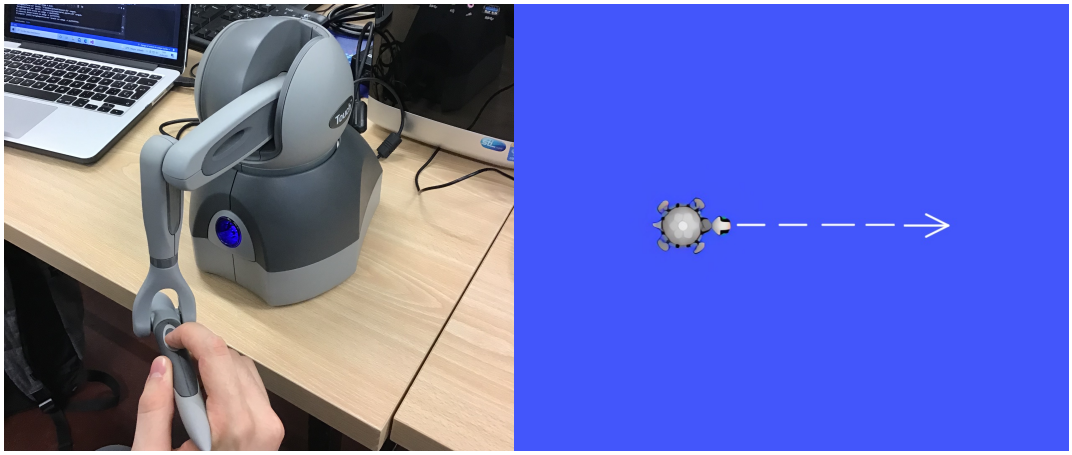

Figura 58. Movimiento determinado por giro antihorario de la articulación J4.

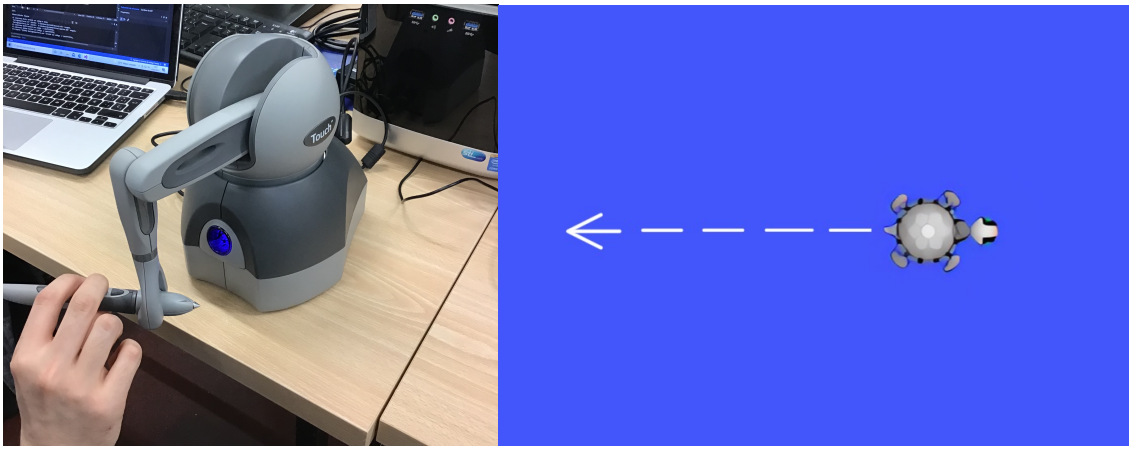

Figura 59. Movimiento determinado por giro horario de la articulación J4.

Para lograr que la tortuga se desplace en la dirección *Y'* se deberá modificar el "*pitch*" del *stylus*, es decir, realizar un movimiento sobre la articulación J5 que sitúe su parte posterior del por encima o por debajo del *HIP.* Mientras se obtenga una lectura del potenciómetro situado en J5 indicando que se han superado los -30º y se esté pulsado el botón 1, se introducirá en la variable *y* del vector *"linear"* el valor 0.4, provocando el desplazamiento de la tortuga en el sentido positivo de la dirección *Y'*. Si por el contrario la lectura de J5 arroja un dato mayor de 30º y se pulsa el botón 1, el valor introducido en y es de -0.4, produciéndose el movimiento de la tortuga en sentido contrario. En caso de que no se cumplan ninguna de las dos condiciones anteriores, el valor almacenado en *y* será cero, por lo que no se producirá ninguna clase de movimiento en esa dirección.

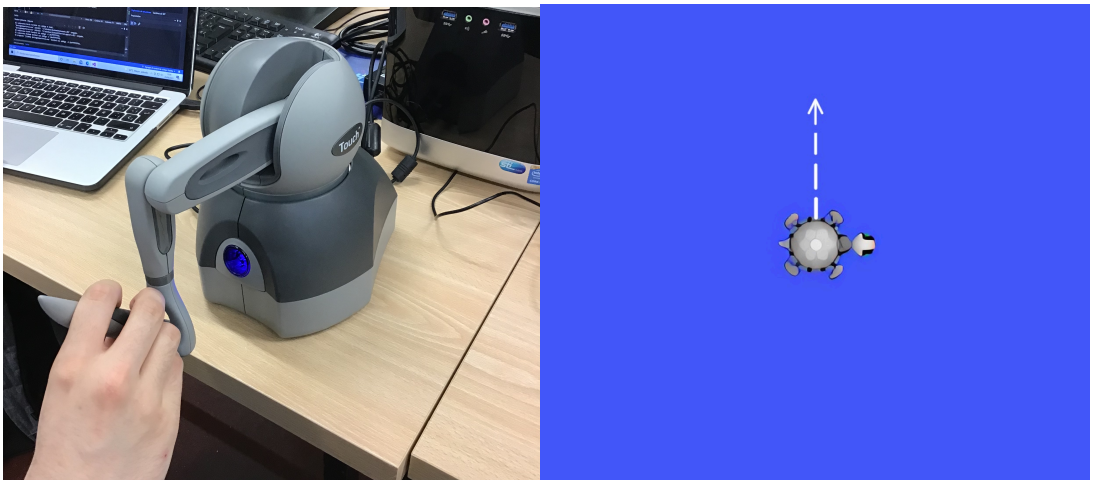

Figura 60. Movimiento del stylus determinado para el avance en el sentido Y' positivo.

Figura 61. Movimiento del stylus determinado para el avance en el sentido Y' negativo.

Por último, si lo que se busca es modificar la orientación de la tortuga, deberá realizarse un movimiento de rotación sobre la articulación J6. Durante los instantes en los que se perciba un giro mayor de 60º en sentido horario o antihorario sobre la presente articulación, encontrándose pulsado el botón 1, se introducirá en la variable *z* del vector el valor -0.2 o 0.2 respectivamente. El primero de ellos determinará un giro en sentido horario alrededor del eje *Z'*, mientras que el segundo lo hará en sentido antihorario. Si no se cumplen ninguna de las dos circunstancias anteriores, la tortuga no rotará.

Figura 62. Movimiento determinado por el giro antihorario de la articulación J6.

Figura 63. Movimiento determinado por el giro horario de la articulación J6.

Los movimientos aquí presentados podrán realizarse de manera separada o combinada, siendo posible, por ejemplo, avanzar en dos direcciones de manera simultánea o realizar un movimiento de avance y giro.

Figura 64. Movimiento combinado en sentido X' negativo y giro antihorario alrededor de Z'.

4.2.4.- Interacción interfaz háptico – *UR3*.

Aquí se expone cómo se ha tratado de adaptar el trabajo previo realizado en el apartado 4.2.3 para el control del robot *UR3,* teniendo en cuenta las limitaciones que causan los problemas de compatibilidad del dispositivo háptico en *Ubuntu* tratados anteriormente.

El planteamiento es muy similar al del apartado anterior: utilizar la orientación de las articulaciones J4, J5 y J6 para determinar los movimientos del robot *UR3,* y servirse de los botones para poder abarcar un mayor conjunto de acciones. El nodo *omni_state* permanece inalterado, realizando las mismas funciones y utilizando los mismos mensajes para comunicarse con *nodo_medio* que en el caso anterior. Éste último por su parte, sufre las modificaciones pertinentes para adaptarse a la nueva situación, transformando esta vez la información recibida desde *omni_state* en valores que sirvan para el control del robot.

Por último, *turtlesim_node* desaparece en favor de una serie de nodos que realizan las tareas de control del robot *UR3* y que se comunican con su controlador de bajo nivel *URControl* mediante *sockets.* El mecanismo de mensajes utilizado por *nodo_medio* para el envío de información a *turtlesim* no se modifica ya que cumple con las características necesarias para su utilización en este caso, es decir, *nodo_medio* continúa publicando un mensaje tipo *geometry_msgs::twist*.

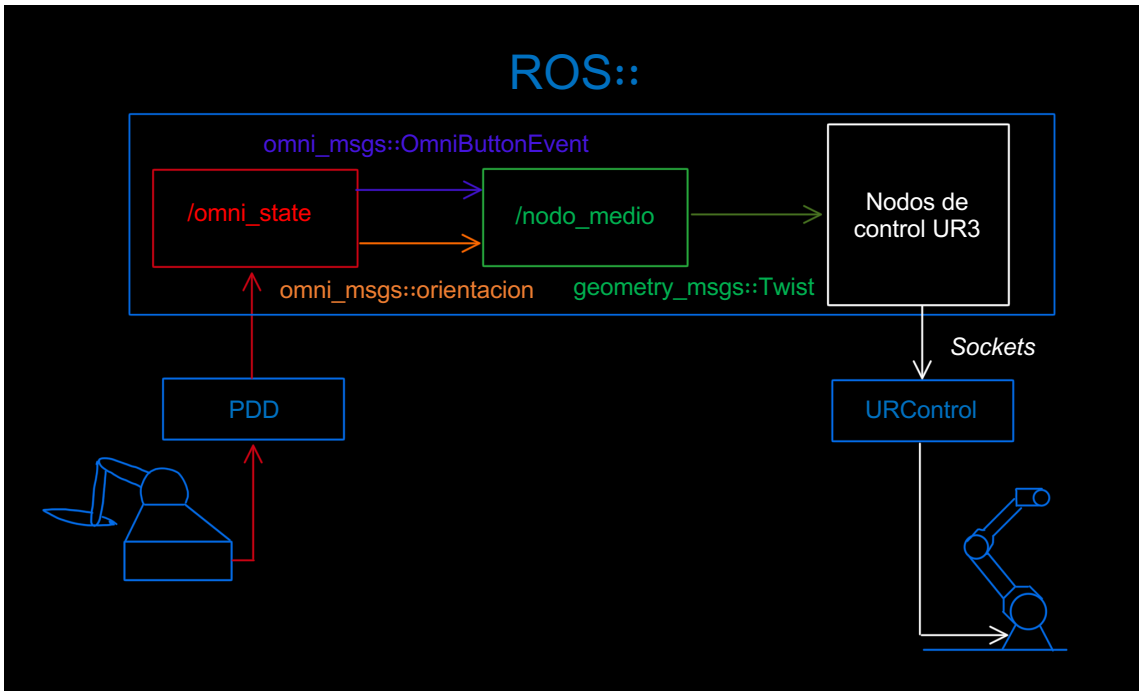

Figura 65. Esquema de funcionamiento.

En este caso se modifica el control por velocidad utilizado en el anterior apartado en favor de un sistema de control por posición. El objetivo es ser capaces de desplazar el *TCP* del robot a un punto concreto de nuestra elección, y contar con la posibilidad de que una vez situado, se modifique la orientación a nuestra conveniencia sin que exista desplazamiento en el espacio.

El botón 1 se utiliza en esta ocasión con dos fines: confirmación y cambio en la precisión de los movimientos. Si se realiza una pulsación corta (menor de 200 *ms*) variará la precisión de los movimientos, mientras que si se realiza una pulsación larga se llevará a cabo el movimiento comandado por el usuario. Es importante contar con distintos modos de precisión, ya que previsiblemente existan durante la vida útil del sistema momentos en los que se prefiera un rápido desplazamiento del mismo, y otros en los que prime la precisión en los movimientos sobre la velocidad de ejecución.

Puesto que no se cuenta con ninguna información proporcionada por los encoders situados en las articulaciones J1, J2 y J3, deberán utilizarse los valores enviados por J4, J5 y J6 tanto para la generación de consignas de posición como de orientación. Se hace uso por ello de dos modos de funcionamiento (posición y orientación), realizándose el cambio entre ellos por flanco de subida del botón 0.

Los mismos desplazamientos del *stylus* que en el apartado anterior determinaban el movimiento de la tortuga, son los que en este caso se utilizan para el movimiento del robot. El desplazamiento en sentido antihorario de la articulación J4 junto con la confirmación del botón 1, supone el desplazamiento del TCP del robot en el sentido positivo de la dirección *X* del eje *world* o la rotación en sentido horario alrededor de la dirección *Z* del mismo eje, dependiendo del modo de uso en que se encuentre. El movimiento del *stylus* en sentido contrario producirá los mismos movimientos, pero en el sentido contrario de los ejes.

En cuanto a la articulación J5, se utiliza la misma acción que determinaba el desplazamiento de la tortuga en sentido *Y'* positivo para que el TCP del robot se desplace en sentido positivo de la dirección Y del eje de referencia *world* o rote en sentido horario respecto de la dirección *X* del mismo eje. De nuevo, si se realiza el mismo movimiento de la articulación en el sentido contrario, se obtienen los mismos movimientos por parte del robot, pero en sentido opuesto.

Por último, la rotación de la articulación J6 en sentido horario se utiliza bajo las mismas condiciones que en el caso anterior: para producir un desplazamiento del *TCP* del robot en sentido positivo de la dirección *Z* del eje *world* o rotación en sentido horario alrededor de la dirección *Y* del mismo eje. Si el objetivo del usuario es ejecutar estas acciones en sentido opuesto, debe rotarse la articulación J6 en sentido antihorario.

En cuanto al mecanismo de control, se introduce una posición y orientación que indican el estado de inicio del robot, para que en función de los movimientos del *stylus* y de la precisión con que se trabaje, se modifiquen esos valores en mayor o menor medida, generándose así las consignas de posición y orientación del robot. Como se exponía anteriormente, es necesario realizar una pulsación larga del botón 1 para que se confirme la orden de movimiento. A la hora de determinar si una pulsación ha sido larga o corta se siguen los siguientes pasos:

El programa *nodo_medio.cpp* cuenta con una sección cíclica cuyo bucle se repite con una frecuencia de 125 Hz. En el momento en que se detecta un flanco ascendente del botón 1, comienza a incrementarse un contador en una unidad con cada iteración. Si dicho contador alcanza el valor 25 (valor logrado durante el transcurso de 200 ms), se interpreta que se está realizando una pulsación larga, se toman las acciones pertinentes sobre las variables de posición u orientación concernidas por los movimientos del *stylus*, se devuelve el valor del contador a cero y se envía a través del mensaje *geometry_msgs::Twist* el nuevo valor a los nodos controladores del robot. Mientras el botón se mantenga pulsado, los valores se actualizan periódicamente cada 200 ms. El menor desplazamiento que se realiza (cada vez que el valor del contador alcance 25) es de 0,1 *mm,* mientras que la mínima rotación es de 0,1 º. Dependiendo del nivel de precisión con que trabajemos, estas variaciones serán de distinta magnitud especificándose 5 niveles.

Si nos encontramos en el primer nivel de precisión, en cada una de las ocasiones en que se detecte una pulsación larga del botón 1 los incrementos de posición y orientación se corresponderán con los mínimos indicados en el párrafo anterior. A partir de ahí, con cada pulsación corta del botón 1 se entrará en cada uno de los

sucesivos niveles de precisión, en que los mencionados incrementos serán 4, 16, 32 o 128 veces mayores respectivamente, retornando al valor inicial con la quinta pulsación.

El mensaje tipo *geometry_msgs::Twist* está conformado por dos vectores: linear y angular. En el primero se almacenan las consignas relativas al posicionamiento, mientras que el segundo se utiliza para recoger aquellas relativas a la orientación.

Habiendo alcanzado este punto, se valora la viabilidad de continuar el desarrollo del presente modelo de control. Si bien se han conseguido superar obstáculos como la lectura errónea de los valores arrojados por las articulaciones J1, J2 y J3, otros como la imposibilidad de generar efectos hápticos debido a incompatibilidades en el *hardware*, ponen en duda la conveniencia de la utilización de los *drivers* utilizados en *Ubuntu* en favor de los previamente testeados en *Windows.* El principal valor añadido de un dispositivo háptico es su capacidad de renderización de fuerzas, y la imposibilidad de su implementación, implica que cualquier sistema que actúe como *joystick,* podría realizar la misma función pero con un coste mucho menor.

Es por ello que se decide aparcar el presente sistema de interconexión y control, en favor de otro desarrollado utilizando los *drivers* ofrecidos para el sistema operativo *Windows.*

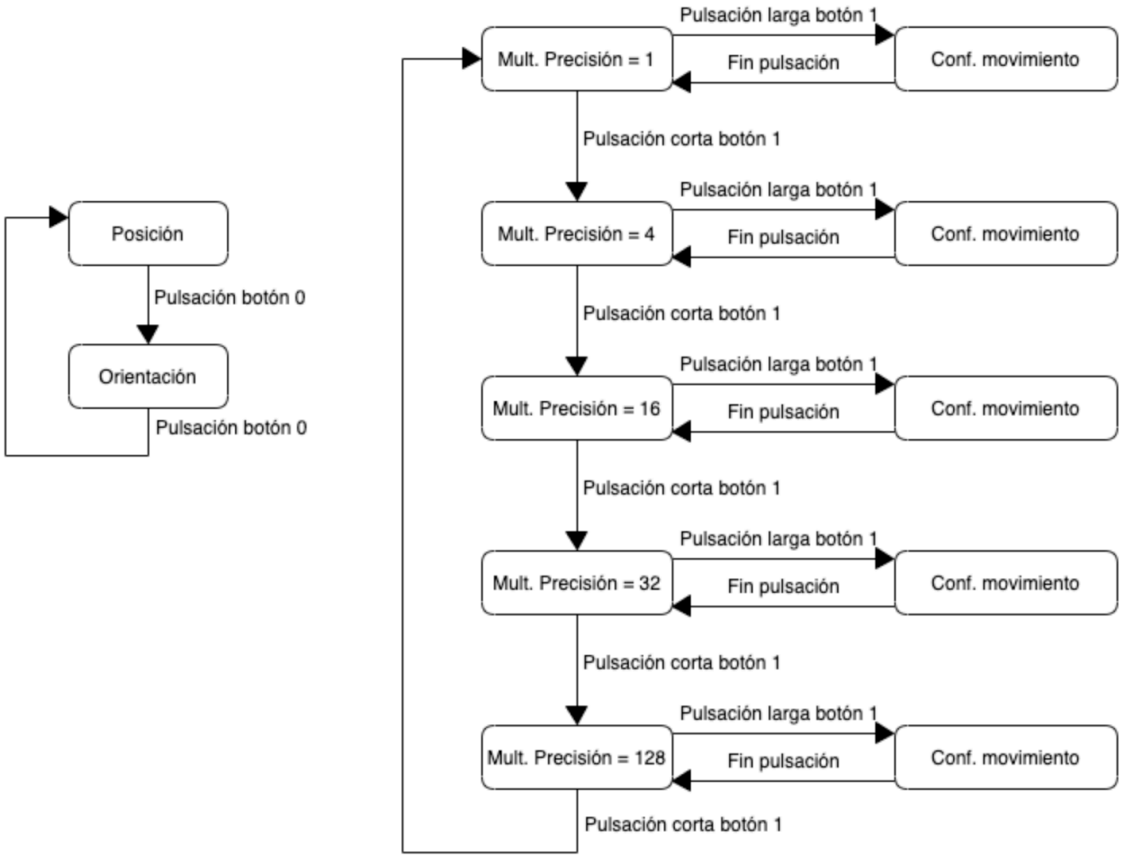

Figura 66. Esquema funciones botones.

4.3.- Sistema de control e interconexión *Windows - ROS*.

Una vez estudiadas las capacidades y limitaciones que presentan los *drivers* del dispositivo háptico en *ROS,* y habiendo hecho lo propio previamente en *Windows,* se determina que la utilización de los *drivers* para *Windows* en lo concerniente a la ejecución de las tareas de control del interfaz háptico, permitirá alcanzar una solución de mayor calidad al problema planteado.

4.3.1.- Desarrollo y características del mecanismo de comunicación y control.

Se propone entonces la reutilización del código desarrollado para la ejecución de los efectos mostrados en el apartado 3.1.4, para que, con las modificaciones pertinentes, sea capaz de generar, almacenar y compartir una estructura la cual contenga información que pueda ser utilizada para el manejo del robot. Al desarrollarse las actividades de control del interfaz háptico en un sistema operativo diferente a aquellas responsables de las acciones del robot colaborativo, deberá establecerse un canal de transmisión de información entre ambos. Se decide optar por comunicación mediante *sockets* regidos por protocolo *TCP/IP*, actuando los programas ejecutados en *Windows* como cliente de un servidor alojado en *Ubuntu.*

Para el manejo del robot se contará con tres modos:

- Modo 0 o de orientación: rotación de la herramienta alrededor de la dirección *Z* del eje *tool0.*
- Modo 1 o de precisión: desplazamiento del *TCP* robótico a baja velocidad.
- Modo 2 o de aproximación: desplazamiento del *TCP* robótico a alta velocidad.

Si el usuario así lo desea podrá cambiar de modo pulsando el botón 0 del *stylus*. El botón 1 por su parte actúa como mecanismo de confirmación del movimiento (de la misma manera que en el apartado anterior) y como “hombre muerto” (mecanismo de seguridad que finaliza los procesos en ejecución si el botón afectado no es pulsado en un plazo de 10 segundos).

En el modo 0 se establece una relación directamente proporcional entre el ángulo definido por el usuario en la articulación J6 y aquel que la herramienta del robot gira alrededor de la coordenada *Z* del eje *tool0.* Por ejemplo, si se lleva a cabo un movimiento de rotación en sentido horario de 30º sobre J6 y se pulsa el botón 1, la herramienta tomará esa misma posición respecto del eje *tool0.* La instrucción utilizada para la ejecución de este movimiento por parte del robot es *movej,* la cual permite el movimiento a una posición de manera lineal en coordenadas articulares.

Los modos 1 y 2 son muy similares, ambos establecen un control por velocidad de la posición del *TCP* del robot, en función de la distancia a la que el *HIP* del interfaz háptico se encuentre del origen de su eje de coordenadas. La única diferencia radica en que el modo 1 cuenta con una velocidad máxima de desplazamiento de 0,1 *m/s* por los 0,6 *m/s* del modo 2.

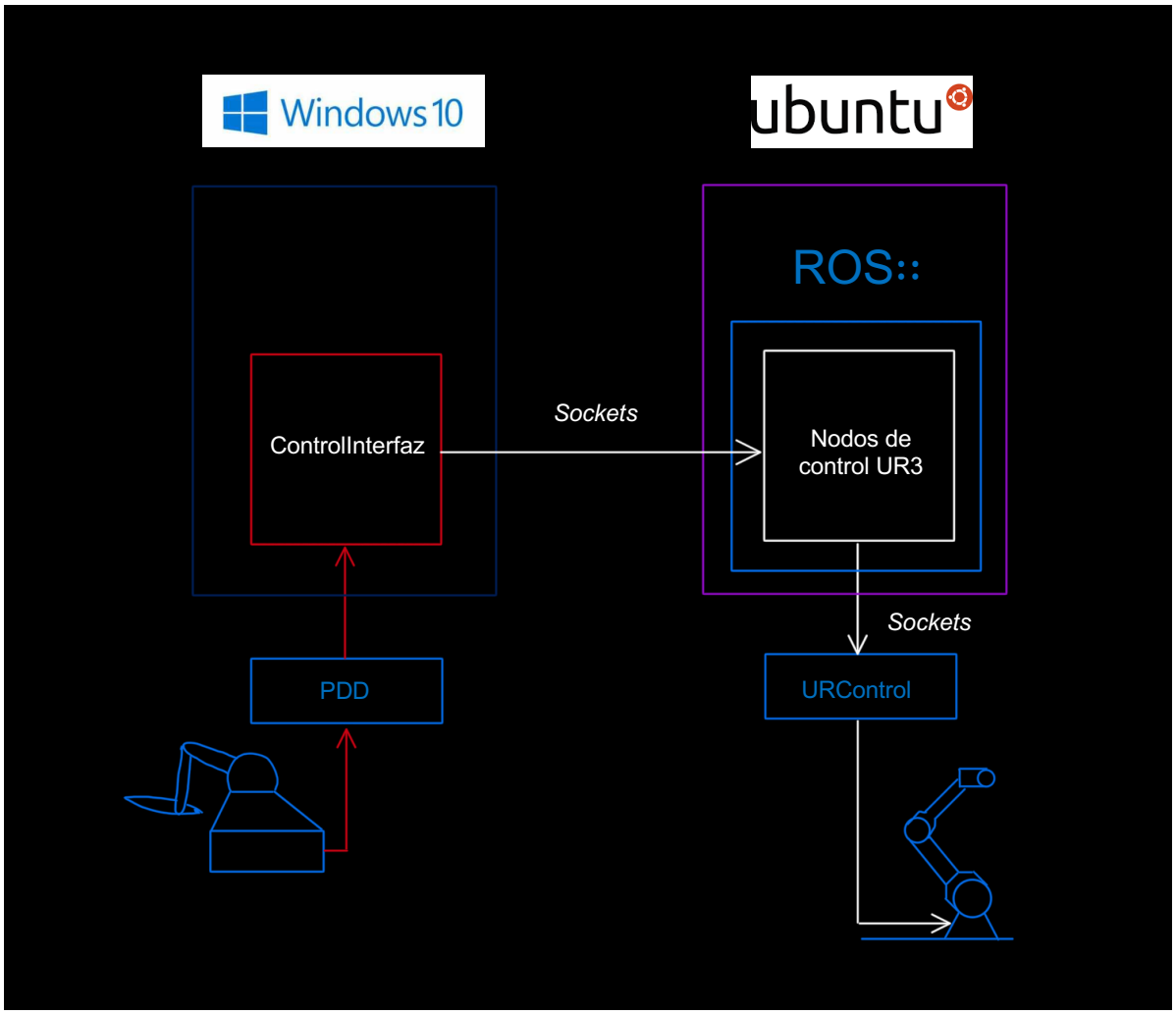

Figura 67. Nuevo esquema de funcionamiento.

Tanto modo precisión como modo aproximación cuentan con un área en el que la velocidad es proporcional al desplazamiento, hasta que se supera un determinado umbral a partir del cual el robot se mueve a velocidad máxima en la o las direcciones de desplazamiento del *HIP*. Por ejemplo, en el eje X del interfaz, este umbral se define a 105 mm en ambas direcciones con respecto al origen. Si se desplaza el HIP una distancia mayor de 105 mm, la velocidad lineal del robot en esa dirección sería la máxima del modo en que se encuentre funcionando el sistema. En caso contrario, si la distancia del *HIP* al origen es mayor de cero, pero no alcanza el umbral, se calcularía la velocidad atendiendo a la siguiente ecuación:

$$V_{TCP}\vec{\imath} = \left(\frac{d_{act}}{d_{umb}} * V_{maxTCP}\right)\vec{\imath}$$

Siendo V_{TCP} la velocidad comandada para el *TCP* robótico en una dirección determinada, d_{act} la distancia del HIP al origen de coordenadas, d_{umb} la distancia umbral y V_{maxTCP} la velocidad máxima determinada por el modo.

Las direcciones del eje de coordenadas del interfaz háptico y *world* no son coincidentes, por lo que se debe realizar una identificación entre ellas que permita un manejo intuitivo del sistema por parte del usuario. En concreto, movimientos en la dirección positiva de *X* en el interfaz háptico deberán suponer un desplazamiento en la dirección negativa de *Y* en *world*, la dirección negativa de *Z* en el interfaz deberá hacerlo sobre la positiva de *X* en *world* y la positiva de *Y* en el interfaz sobre la positiva de *Z* en *world.*

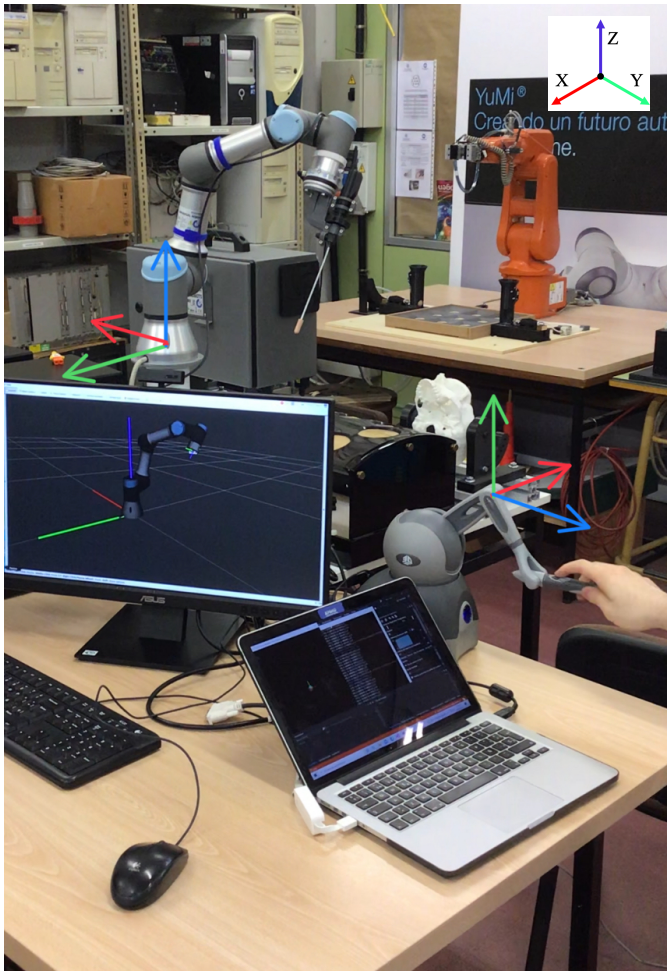

Figura 68.Disposición de los sistemas.

La instrucción implementada para el control del robot en estos dos modos es *speedl,* la cual establece una aceleración lineal en coordenadas cartesianas hasta alcanzar una velocidad establecida, permitiendo mantener un control por velocidad en tiempo real del robot.

El mecanismo de conexión entre los procesos ejecutados en *Windows* y aquellos llevados a cabo en *ROS* consiste en el uso de *sockets.* El ejecutado en *ROS* solicitará con una frecuencia de 125 Hz el envío de información por parte del cliente acerca del estado del interfaz háptico, proporcionándose un mensaje con los siguientes argumentos: tres valores indicando la posición del *HIP*, tres indicando el estado de las articulaciones J4, J5 y J6, tres indicando la variación del estado de las articulaciones, uno indicando el modo de uso y un último que informa del estado del botón 1. Todos ellos se aglutinan en un dato de tipo *string* separado por comas que permiten la diferenciación los distintos campos por parte del servidor.

Se utiliza una frecuencia de 125 Hz ya que es la misma utilizada en los mecanismos de comunicación utilizados entre los procesos ejecutados en *ROS* y *URControl*, y a la que, en consecuencia, se podrá transmitir una mayor cantidad de información sin emplear capacidad computacional de manera innecesaria.

Esta frecuencia de comunicación supone un obstáculo a la hora de generar una respuesta háptica dependiente de los contactos que sufra el robot, ya que como apuntaba en anteriores ocasiones, el ser humano necesita de una frecuencia de 1000 Hz para obtener una experiencia táctil satisfactoria. Es por ello que se utiliza el modelo muelle-amortiguador desarrollado en el apartado 3.1.4 como elemento de interfaz háptico con el usuario.

Se elimina el menú que permitía la selección de otros modos de funcionamiento y se le otorga a la constante de elasticidad un valor de 80, por un valor de 1,7 en la constante de amortiguación. Si el usuario no realiza ningún movimiento, el *HIP* permanecerá en posición de reposo en el origen del sistema de coordenadas. A medida que el usuario aleje el *HIP* del origen, la fuerza ejercida por el dispositivo irá en aumento de acuerdo con la velocidad comandada al robot colaborativo, de tal manera que el usuario sea consciente de que está pidiendo una velocidad mayor al movimiento del robot colaborativo. En los modos 1 y 2 si se alcanza la zona de operación que comanda la velocidad máxima, se producirá una reducción drástica de la fuerza ejercida por el interfaz, en concreto 8 veces menor que la que correspondería por la ley de *Hooke*, con la finalidad de que el usuario sea capaz de percibir la transición entre zonas.

El dispositivo háptico alcanza la fuerza máxima que es capaz de renderizar antes de llegar a la longitud de transición, por lo que, a pesar de percibirse un salto en la fuerza ejercida, ésta no desciende de manera drástica. No se necesita en este caso determinar longitudes distintas en las que el modelo de renderización de fuerzas cambie, ya que el componente amortiguador del modelo impide la acción ejercida por las vibraciones que aparecían en el modelo muelle.

Una vez explicado el funcionamiento, se avanza a la fase de comprobación.

4.3.2.- Verificación del funcionamiento.

Como último paso en el proceso de diseño, se verifica el correcto funcionamiento de los sistemas. Se utiliza para ello, en primer lugar, *Polyscope* junto con el visualizador nativo de *ROS Rviz,* como entorno de simulación y visualización en el que resolver los últimos problemas o imprecisiones que hayan podido generarse en el desarrollo del código necesario para la operación. Una vez el resultado sea satisfactorio, se avanzará a la comprobación del sistema real.

Para la simulación se determina un movimiento secuencial en el que el *TCP* robótico se desplaza en primer lugar en dirección y sentido *Y* creciente respecto a *world,* posteriormente avance en dirección y sentido *Z* decreciente, para finalizar con el brazo robótico extendido en dirección y sentido *X* decreciente.

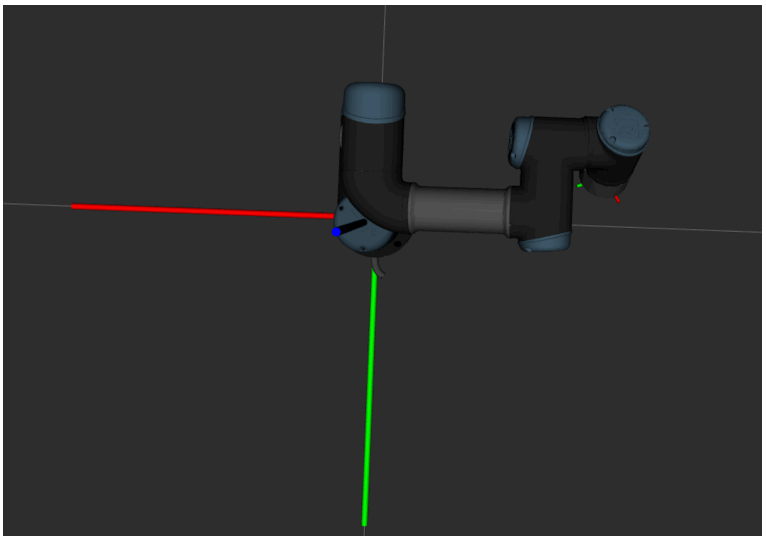

Figura 69.Posición inicial.

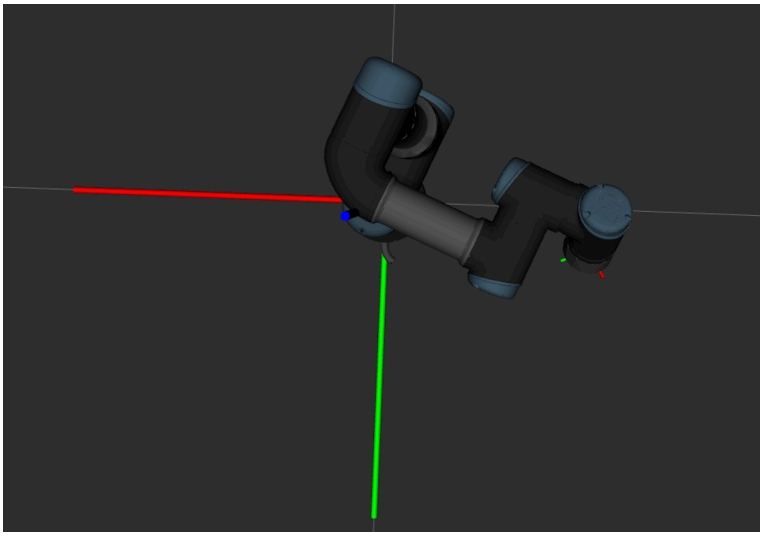

Figura 70. Desplazamiento en Y.

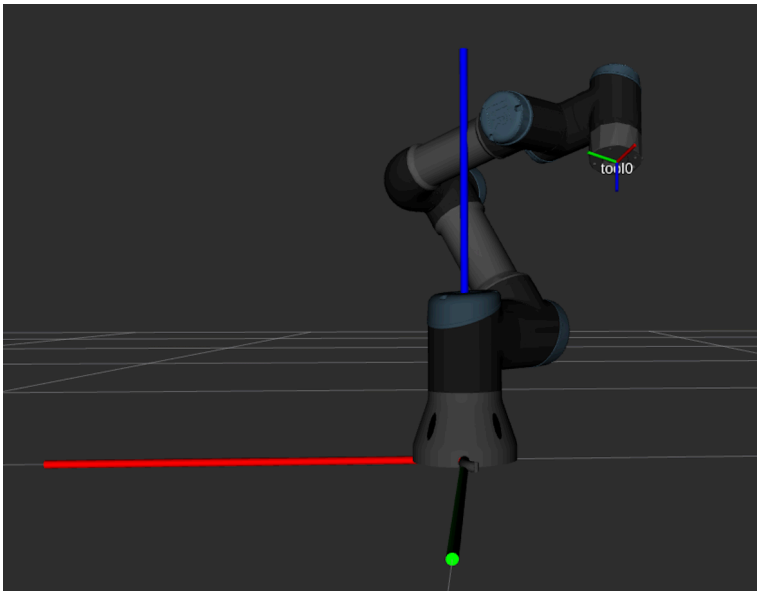

Figura 71. Posición inicial movimiento sobre Z.

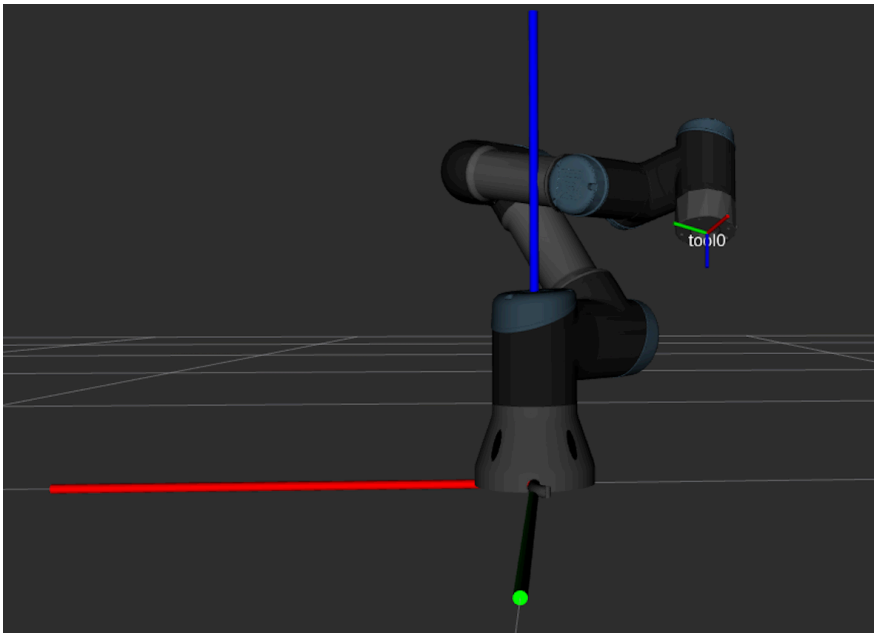

Figura 72.Posición final movimiento sobre Z.

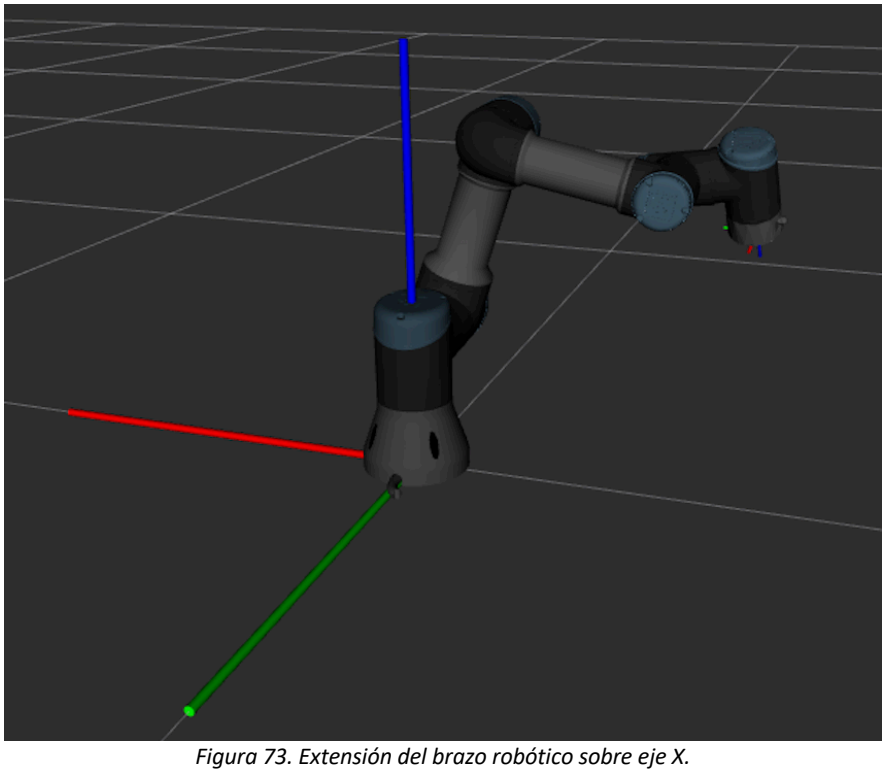

Figura 73. Extensión del brazo robótico sobre eje X.

Puesto que los resultados han sido satisfactorios, se comprueba el funcionamiento del sistema real. Primero se comprobarán los movimientos por separado, de tal manera que quede corroborada la correspondencia entre los resultados obtenidos en la simulación y los movimientos ejecutados por el robot. Una vez haya quedado patente la segura operación del robot, se llevarán a cabo una serie de movimientos que emulen la utilización real del robot.

Se comienza con la comprobación del funcionamiento del modo 0:

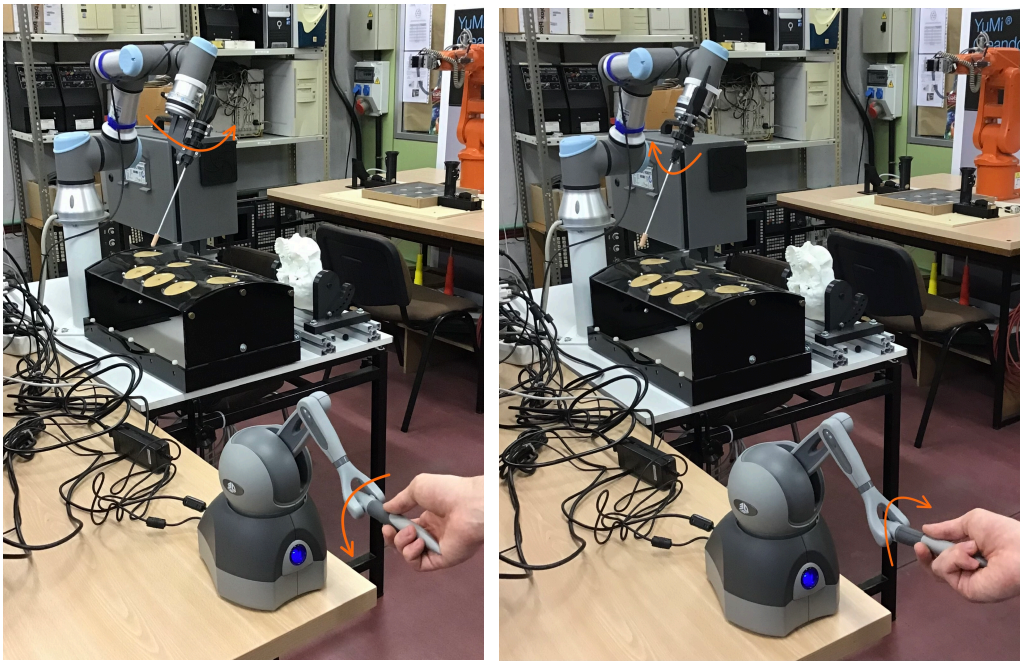

Figura 74. Movimientos modo 0: (Izq) giro antihorario, (Dcha) giro horario.

Tal y como se puede observar, los movimientos en sentido horario o antihorario realizados sobre la articulación J6 del dispositivo háptico, se traducen en movimientos de la misma clase alrededor del eje *Z* del sistema de referencia *tool0.* Se asume el correcto funcionamiento del modo 0 y se avanza a la comprobación de los movimientos lineales del *TCP* robótico:

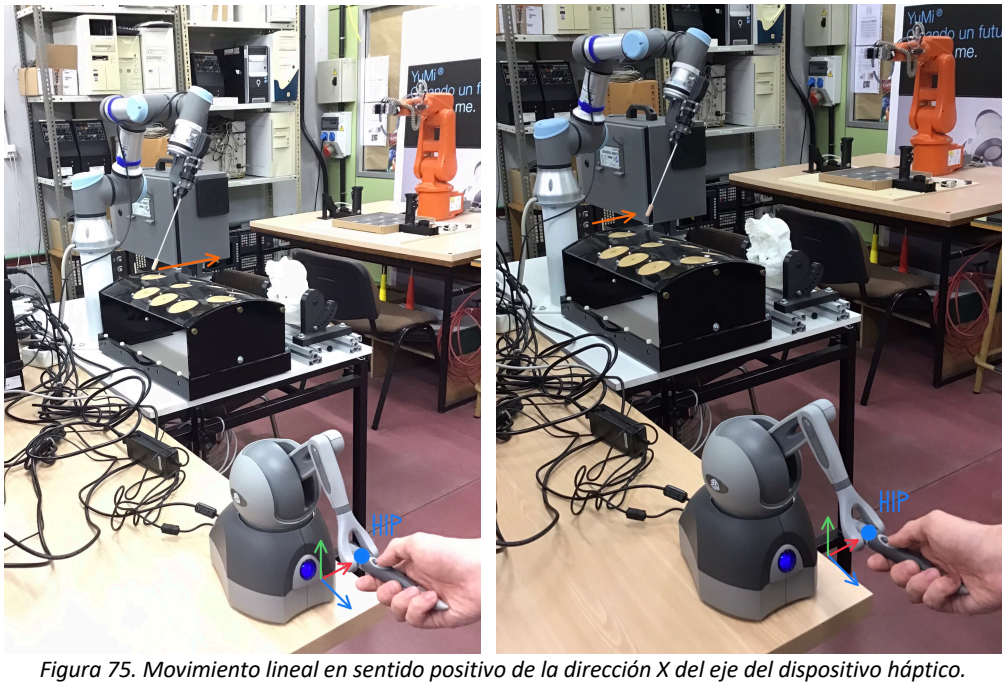

Figura 75. Movimiento lineal en sentido positivo de la dirección X del eje del dispositivo háptico.

Como se muestra en las imágenes, el desplazamiento del HIP en sentido *X* positivo, se traduce en un movimiento acorde del TCP del robot. Se comprueba el mismo movimiento en sentido contrario.

Figura 76.Movimiento lineal en sentido negativo de la dirección X del eje del dispositivo háptico.

Funciona de la manera esperada y se realiza un nuevo movimiento, esta vez en sentido ascendente de la dirección Y del eje del dispositivo háptico.

Figura 77.Movimiento lineal en sentido positivo de la dirección Y del eje del dispositivo háptico.

Una vez llevado a cabo, se comprueba el funcionamiento en el eje *Z* con un movimiento en sentido positivo, seguido por otro en sentido negativo:

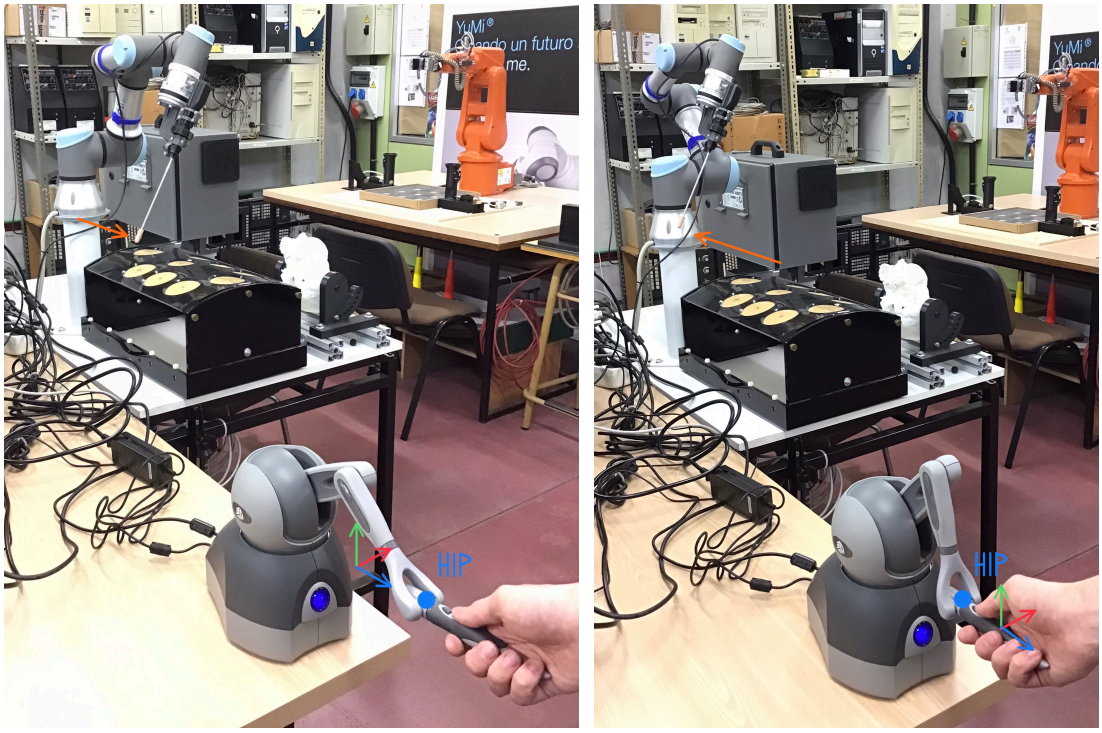

Figura 78. Movimientos a lo largo de la dirección Z del eje del dispositivo háptico (Izq) sentido positivo, (Dcha) sentido negativo.

El robot reacciona de la forma adecuada y se introduce un último movimiento en esta fase de pruebas, consistente en un movimiento compuesto por un desplazamiento en el sentido positivo de la dirección *Z* y en el sentido descendente de la dirección *Y*.

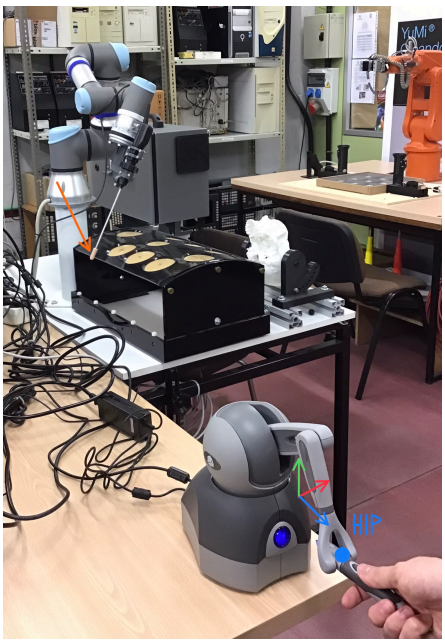

Figura 79. Movimiento descendente sobre Y en sentido Z positivo.

Habiendo visto que los *inputs* en el interfaz háptico obtienen la respuesta adecuada por parte del robot, se plantea la realización de una operación que pudiera asemejarse a una utilización real del mismo. Se parte desde la posición de reposo del robot, y mediante las actuaciones necesarias por parte del usuario, se introduce la punta de la herramienta del robot en uno de los orificios de un *pelvitrainer.* Se utilizará el modo 2 por su mayor velocidad para realizar un movimiento de aproximación.

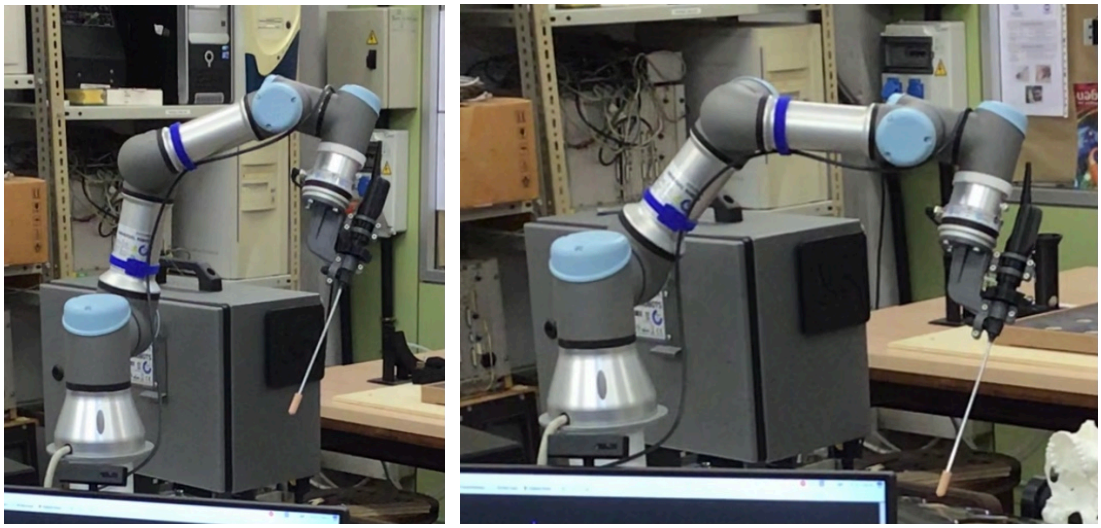

Figura 80. Movimiento de aproximación.

Una vez realizado satisfactoriamente el movimiento de aproximación, se utiliza el modo 1 para introducir la herramienta en el orificio ya que su menor velocidad permite la realización de movimientos más precisos.

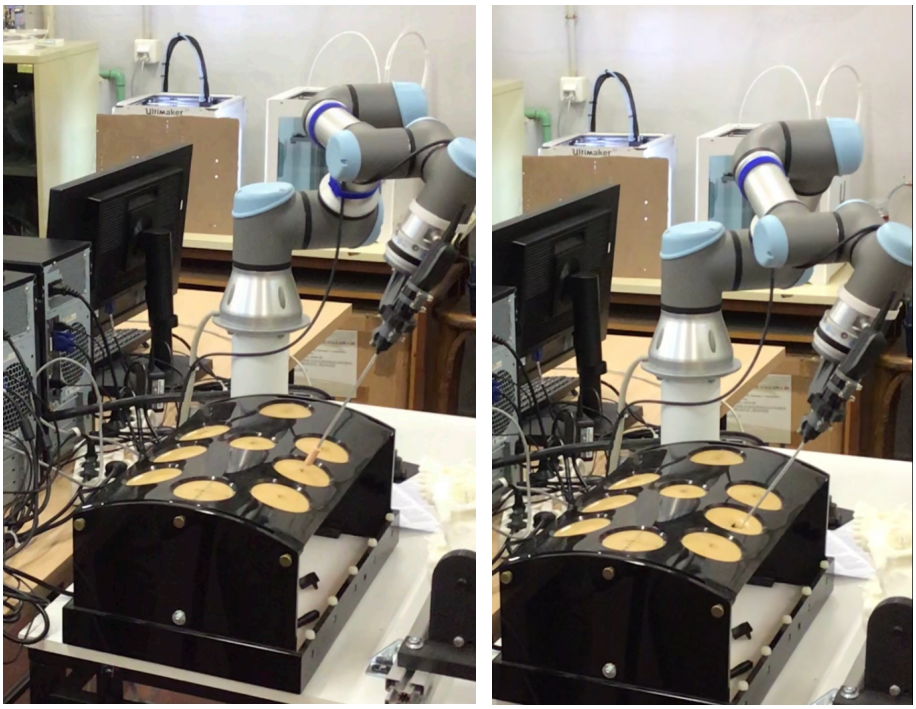

Figura 81. Movimiento de introducción.

Durante todo el proceso los sistemas funcionan correctamente: el robot replica de manera adecuada los movimientos establecidos por el interfaz háptico, no se producen retrasos ni pérdidas de conexión y la realimentación háptica funciona en todo momento de manera adecuada. Se puede afirmar entonces que se ha alcanzado una solución funcional y robusta para la interacción entre interfaz háptico y robot colaborativo.

5.- Conclusiones y líneas futuras.

5.1.- Conclusiones.

Iniciaba el presente trabajo de fin de grado planteando los siguientes objetivos:

- Búsqueda y análisis de información concerniente al funcionamiento, características, limitaciones y usos tanto habituales como experimentales de los interfaces hápticos, robots colaborativos y Robot Operating System, a fin de establecer el estado del arte.

- Puesta a punto de un dispositivo de interfaz háptico 3D Systems Touch y estudio del entorno de desarrollo OpenHaptics Toolkit proporcionado con el dispositivo.

- Desarrollo e implantación de los canales de comunicación entre el interfaz háptico y robot colaborativo.

- Establecimiento de un método de control adecuado a las posibilidades y limitaciones de ambos sistemas.

- Validación de la solución alcanzada.

De acuerdo con el primer objetivo fijado, se realizó una búsqueda de información que permitió determinar aspectos del funcionamiento, historia y evolución además de posibles aplicaciones de cada uno de los elementos utilizados, proporcionando una imagen global del estado del arte de cada uno de ellos.

En lo respectivo al segundo punto, el análisis los distintos recursos incluidos en el *OpenHaptics Toolkit*, permitió establecer unas pautas para la mejor utilización de cada uno de ellos, pudiendo llegar a la conclusión de que lo mejor para el desarrollo del presente mecanismo de control sería la utilización de la *API* de bajo nivel *HDAPI.* Se elaboró, además, un programa que permitía la demostración de alguna de las capacidades hápticas del interfaz, que pudo ser posteriormente reutilizado para el mecanismo de control.

En cuanto a los medios de comunicación, se plantearon dos opciones: la ejecución de todos los programas necesarios en *ROS* a fin de utilizar sus mecanismos de comunicación nativos, o la implementación de las tareas de control del interfaz háptico en *Windows,* mientras que aquellas concernientes al robot colaborativo se realizan en *Ubuntu*, utilizando *sockets* como medio de conexión entre ellas. Finalmente se optó por la segunda de las opciones, ya que la operación del interfaz háptico a través de las herramientas diseñadas para *Ubuntu* traían consigo incompatibilidades de *hardware* que limitaban las capacidades del dispositivo.

El método de control establecido se determinó como un control por velocidad en lo relativo al posicionamiento del *TCP* l robot, utilizándose un control por posición en el caso de la orientación de la herramienta. Esta forma de control utilizada en conjunto con la realimentación háptica descrita en el apartado 4.3.1, permiten un manejo sencillo e intuitivo por parte del usuario.

Por último, se realizó una comprobación del sistema de comunicaciones y control, verificándose la precisión y robustez del mecanismo diseñado.

5.2.- Líneas futuras.

El presente trabajo se plantea como un primer paso que permita el futuro desarrollo de aplicaciones más refinadas que puedan abarcar un amplio espectro de funciones. En consecuencia, se pueden plantear algunas líneas de investigación futuras como las que se describen a continuación:

- El desarrollo de una realimentación háptica basada en los contactos o fuerzas sufridas por el brazo del robot colaborativo supondría un interesante valor añadido al conjunto tanto en aplicaciones dirigidas a la teleoperación como a la simulación.

- Aumento de la velocidad de comunicación con el robot colaborativo. Se tiene que el controlador del robot (*URControl)* mantiene con el ordenador un proceso de comunicación por *sockets* con una frecuencia de 125 Hz. Si quisiésemos utilizar una realimentación háptica basada en los contactos ejercidos por el robot, se deberá encontrar una manera de aumentar hasta los 1000 Hz la frecuencia de recepción de la información referente a las fuerzas que actúan sobre el robot.

- Control del movimiento de manera articular, es decir, ya que ambos sistemas están conformados por brazos articulados de 6 GDL, podría diseñarse una aplicación la cual, mediante la renderización de fuerzas por parte del interfaz háptico, realizase una identificación de los ángulos formados por las articulaciones del robot en las del interfaz háptico. Esto permitiría un control por posición tanto de la orientación como de la posición del *TCP* del robot.

Bibliografía

[1] News Bites Pty Ltd, «Cobots Lead the Future of the Global Industrial Robotics Market, Finds Frost & Sullivan,» *News Bites - Private Companies,* 22 4 2021.

[2] «Technical Q&A: trends and developments in cobotics,» *The Engineer,* Marzo 2020.

[3] J. Vincent, «Haptic feedback is making VR surgery feel like the real thing,» *The Verge,* 2018.

[4] P. Bertrán Prieto, «Médico +,» [En línea]. Available: https://medicoplus.com/neurologia/sentido-tacto. [Último acceso: 23 Mayo 2021].

[5] Minhoon Park, Bo-Gyu Bok, Jong-Hyun Ahn, Min-Seok Kim , «Recent Advances in Tactile Sensing Technology,» *Micromachines ,* vol. 7, p. 321, 2018.

[6] 3D Systems, «3D Systems,» 3 Mayo 2021. [En línea]. Available: www.3dsystems.com.

[7] Salisbury Kenneth, Conti Francois, Barbagli Federico, «Haptic Rendering: Introductory Concepts,» 2004.

[8] Reuters, «Fabrican un "Kindle para ciegos".,» 2019. [En línea]. Available: https://www.reuters.com/article/tecnologia-braille-lector-idESKCN1P90HH. [Último acceso: 23 Mayo 2021].

[9] J. L. Outón Méndez, «Software de Control Desarrollado en ROS para Teleoperar de Manera Eficiente el Robot KUKA Lightweight mediante el Háptico Phantom Omni,» Vitoria, 2013.

[10] 3D Systems, «OpenHaptics Toolkit Version 3.5.0 Programmer's Guide».

[11] Ortega Michael, Redon Stephane, Coquillart Sabine, «A Six Degree-of-Freedom God-Object Method for Haptic Display of Rigid Bodies with Surface Properties,» *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS,* vol. 13, nº 3, 2007.

[12] X. Liu, G. Dodds, J. McCartney, B.K. Hinds, «Virtual DesignWorks—designing 3D CAD models via haptic interaction,» *Computer-Aided Design,* pp. 1129-1140, Octubre 2004.

[13] «Artec 3D,» [En línea]. Available: https://www.artec3d.com/es/3d-software/geomagic-freeform. [Último acceso: 29 Mayo 2021].

[14] Takehiro Miki, Toshinori Iwai, Kazunori Kotani, Jianwu Dang, Hideyuki Sawada, Minoru Miyake, «Development of a virtual reality training system for endoscope-assisted submandibular gland removal,» *Cranio-Maxilo-Facial Surgery,* 2016.

[15] H. I. M. A. Omara; K. S. M. Sahari, «Indoor mapping using kinect and ROS,» de *2015 International Symposium on Agents, Multi-Agent Systems and Robotics* , 2015.

[16] J. Edward Colgate, Michael Peshkin, Stephen H. Klostermeyer, «Intelligent Assist Devices in Industrial Applications: A Review».

[17] P. Horno Pérez, «Robot UR5 guiado por visión artificial.,» 2018.

[18] Schneider Electric implementa Robots Baxter en su cadena de montaje., «infoPLC,» 2014. [En línea]. Available: https://www.infoplc.net/historias-exito/item/102267-schneider-electric-implementa-robots-baxter-cadena-montaje. [Último acceso: Junio 2021].

[19] Samsung Introduces Latest Innovations for a Better Normal at CES 2021, «Samsung Newsroom U.S.,» 2021. [En línea]. Available: https://news.samsung.com/us/samsung-ces-2021-latest-innovations-for-a-better-normal/. [Último acceso: Junio 2021].

[20] Á. M. García, «Estrategia de control del robot UR3 para entornos de cirugía robotizados,» Valladolid, 2020.

[21] Cobot Applications, «Universal Robots,» 2020. [En línea]. Available: https://www.universal-robots.com/blog/cobot-applications/. [Último acceso: Junio 2021].

[22] A. Prof. de Luca, «Geomagic Touch. Elective in Robotics - Haptic and Locomotion Interfaces,» Roma .

[23] T. T. Specifications, «3D Systems,» [En línea]. Available: https://3dsystems.com/haptics-devices/touch/specifications. [Último acceso: 3 5 2021].

[24] Y. Fernandez, «Xataka,» 23 8 2019. [En línea]. Available: https://www.xataka.com/basics/api-que-sirve. [Último acceso: 4 5 2021].

[25] Sensable Haptic Toolkits, «OpenHaptics Toolkit».

[26] 3D Systems, «OpenHaptics Toolkit Version 3.5.0 API Reference Guide».

[27] «Quora,» [En línea]. Available: https://www.quora.com/What-are-over-damped-critically-and-under-damped-systems.

[28] M. Bélanger-Barrette, «Questions on Universal Robots Answered: Here!,» [En línea]. Available: https://blog.robotiq.com/all-your-questions-on-universal-robots-answered-here. [Último acceso: Junio 2021].

[29] Universal Robots, «PolyScope Manual,» 2018.

[30] A. García Cazorla, «ROS: Robot Operating System,» Cartagena, 2013.

[31] Yutao Chen, Jing Zhu, Min Xu, Hao Zhang, Xuming Tang, Erbao Dong, «Application of Haptic Virtual Fixtures on Hot-Line Work Robot-Assisted Manipulation.,» de *Intelligent Robotics and Applications*, Shenyang, 2019.

[32] C. Hatzfeld y T. A. Kern, Engineering Haptic Devices. A Beginner's Guide..

[33] Cobots, what does collaborative robot mean? Everything you need to know about., «Tm-robot,» [En línea]. Available: https://www.tm-robot.com/en/blog/what-does-collaborative-robot-mean/. [Último acceso: Junio 2021].

[34] D. Edwards, «Robotics & Automation News,» Mayo 2016. [En línea]. Available: https://roboticsandautomationnews.com/2016/05/24/a-history-of-collaborative-robots-by-universal-robots/4684/.

[35] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, *ROS: an open-source Robot Operating System.*

[36] What is XML-RPC?, «XML-RPC,» [En línea]. Available: http://xmlrpc.com/.

[37] «ROS.org,» [En línea]. Available: wiki.ros.org/ROS/Concepts. [Último acceso: 15 Junio 2021].

[38] C. Applications, «Universal Robots,» 2020. [En línea]. Available: https://www.universal-robots.com/blog/cobot-applications/. [Último acceso: Junio 2021].

[39] «Universal Robots Launches UR3,» Universal Robots, [En línea]. Available: https://www.universal-robots.com/about-universal-robots/news-centre/universal-robots-launches-ur3/. [Último acceso: Junio 2021].