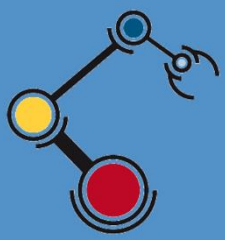
Escuela Politécnica Superior

Entorno virtual para el estudio de señales EMG



Máster Universitario en Automática y Robótica

Trabajo Fin de Máster

Autor:
Lucía Mas Lillo

Tutor/es:
Santiago Timoteo Puente Méndez

Andrés Úbeda Castellanos

Julio 2018

Universitat d'Alacant
Universidad de Alicante

Universidad de Alicante

Máster Universitario en Automática y Robótica

Trabajo de Fin de Máster

Entorno virtual para el estudio de señales EMG

Autor

Lucía Mas Lillo

Tutores

Santiago Timoteo Puente Méndez

Andrés Úbeda Castellanos

Alicante, Julio de 2018

Dedicatoria

A JC por estar a mi lado todo este tiempo.

A mis padres, mis hermanas y mis iaios por animarme y apoyarme.

A Ilias por darme ánimos y enseñarme las valerianas.

Agradecimientos

Quisiera dar las gracias a mi tutor Santiago por darme ideas, encaminarme adecuadamente a la resolución de los problemas encontrados y darme la posibilidad de poder aprender a realizar artículos científicos.

A mi cotutor Andrés por darme ideas de mejora para desarrollar la memoria de este proyecto.

A Brayan y John por enseñarme cómo funciona la mano Shadow.

A mis profesores por enseñarme lo necesario para poder realizar el presente trabajo.

A la Universidad de Alicante por facilitarme los materiales necesarios.

A mis familiares y amigos por dejarme probar la aplicación en sus dispositivos y participar en la encuesta.

Sin todos vosotros no podría haberse desarrollado este proyecto.

Abstract

The aim of this Project is to expand the simulation system developed to control the Allegro on Unity3D hand to use the Shadow hand. In addition the possibility of loading realistic textured hands has been added. The system allows the remote control of the Shadow hand using ROS nodes. A Matlab signal acquisition module is used and the information is sent to the ROS nodes using Matlab. To control the simulation using EMG signals, control commands has been defined in the simulation environment which can be used by the user's interface or using the EMG signals. It is possible to control two robotic hands simultaneously. In the case of using an Allegro hand is possible to use the system in a collaborative way.

As a result, an intuitive system has been obtained. The system has a good update rate on Allegro Systems. As regards the connection with the hands. Shadow is slower than Allegro due to the use of a planner. Shadow simulated system is slower than the real hand.

Keywords

ROS, Matlab, EMG, Unity3D, simulation, robotic hand, Allegro, Shadow

Resumen

En este proyecto se amplía el sistema de simulación en Unity3D realizado para la mano robótica Allegro a la mano Shadow. Además se añade la posibilidad de poder cargar las manos con textura realista. Permite el control de la mano Shadow de manera remota mediante el nodo de ROS correspondiente. Se integra también con un módulo de adquisición de señales que pasa la información a los nodos usando ROS mediante Matlab. Para el control de la simulación mediante señales EMG, se definen unos comandos de control en el entorno de simulación que pueden ser utilizados desde dichas señales o por el usuario desde la interfaz. El sistema posibilita el control de dos manos de manera simultánea y en el caso de la mano Allegro con un control colaborativo.

Como resultado se ha obtenido un sistema intuitivo. Con respecto a la conexión con las manos, el sistema tiene una buena tasa de actualización en los sistemas Allegro. En el caso de la Shadow el retardo se incrementa debido al uso del planificador, siendo mayor este retardo el sistema simulado que en el real.

Palabras clave

ROS, Matlab, EMG, Unity3D, simulación, mano robótica, Allegro, Shadow

Índices

Índice de contenido

Índice de figuras

Índice de tablas

Índice de abreviaturas

AHAS: Allegro Hand Application Studio

EOL: End Of Life

EPS: Escuela Politécnica Superior

ROS: Robotic Operating System

URDF: Undefined Robot Description Format

1. Introducción

1.1. Motivación

Las interfaces hombre-máquina son algo imprescindible hoy en día, el desarrollo de una buena interfaz permite crear sistemas intuitivos y fáciles de utilizar. En el mundo industrial hasta hace poco no se tenía en cuenta la importancia de la interfaz cuando se desarrollaba un sistema de control. El uso de interfaces bien diseñada permite a los usuarios poder trabajar de manera más rápida y simplificada. Las interfaces han evolucionado mucho y dónde mayor ha sido el cambio es en este sector, dónde ahora se desarrollan interfaces más intuitivas.

En este proyecto se ha propuesto una continuación del proyecto de final de grado realizado el año pasado en el grado de Ingeniería Multimedia por mí y con título *"Simulación en Unity y control de una mano robótica"*, con una calificación de 10, que consistía en realizar una interfaz hombre-máquina para el controlar y simular la mano Allegro, con varios usuarios en un ambiente colaborativo en varias plataformas. Para el control era necesario utilizar el controlador existente en ROS comunicando la aplicación con los nodos correspondientes. Para lo que se creó una comunicación usando sockets TCP donde el servidor publica en los *topics* necesarios para realizar la comunicación. El sistema fue testeado en varios dispositivos iOS y Android algunos de ellos descatalogados desde hace años. La aplicación de control se ejecutó en todos sin problemas, permitiendo enviar y recibir las posiciones de una única mano.

Como mejoras se propone añadir una segunda mano al control, pudiendo usar dos manos simultáneamente, manteniendo el entorno colaborativo en sistemas multiplataforma. Además se añade al sistema la carga de una mano de 5 dedos, la mano Shadow para poder realizar el control usando tanto la aplicación desarrollada como señales EMG en el sistema Allegro y en Shadow.

En este proyecto se ha desarrollado un entorno para poder realizar el entrenamiento de prótesis de manera visual. Es posible poder enviar órdenes de las señales EMG sin estar conectado a la mano real o virtual del sistema en GNU/Linux y enviar las órdenes de control a la mano simulada desde el dispositivo telefónico. También se ha pensado en el uso de *shaders* que puedan ser usados en dispositivos móviles sin consumir demasiados recursos que permitan poder usar texturas complejas como es la piel humana.

1.2. Marco del trabajo

Este proyecto se engloba en varias áreas de conocimiento, se enumeran a continuación las más destacables

- Interfaces hombre-máquina para el control de robots
- Señales EMG
- Entrenamiento de prótesis
- Robótica
- Control en sistemas colaborativos
- Aplicaciones multiplataforma
- Simulación
- ROS
- Creación de aplicaciones en dispositivos telefónicos
- Redes
- Programación en sistemas con pocos recursos
- Animación
- Aplicación de texturas en modelos 3D
- Ficheros de carga de robots
- Neuro-robótica
- Bioseñales

1.3. Contribuciones generales

En este trabajo de final de master se ha realizado una interfaz que puede comunicarse con una serie de nodos en ROS para interactuar manos Allegro o Shadow en las versiones reales o simuladas de estos sistemas en GNU/Linux. Este sistema permite el movimiento de las articulaciones de las manos desde uno o varios dispositivos simultáneamente. La interfaz se ha desarrollado en Unity3D para así mostrar el modelo en 3D y poder simular colisiones. Los dispositivos móviles conectados al sistema reciben los datos de las posiciones actuales del robot en tiempo real. Es posible realizar el control del sistema a partir de señales EMG, usando Matlab para realizar la interpretación de estas señales y comunicarse con los nodos en ROS. El módulo de Matlab ha sido proporcionado, por lo que no se realiza el procesamiento en este trabajo. Lo que se hace es comunicarse con ese módulo y dependiendo de si se publica un 1 o un 0 realizar un movimiento de cierre o apertura de la mano. Los movimientos mediante señales EMG están predefinidos en unos comandos de control dentro de la interfaz. Estos comandos de control pueden ser usados directamente desde la interfaz además de ser ejecutados usando señales EMG.

1.4. Estructura de la memoria

Una vez explicado a grandes rasgos lo que se ha realizado en este trabajo de final de master, se realizará una breve introducción al proyecto y el estado del arte dónde se explicarán tanto conceptos teóricos como aplicaciones y se explicará la innovación que añade este proyecto.

Se presentan en los siguientes apartados información suficiente como para poder reproducir el trabajo desarrollado. Se expondrán los diferentes problemas que han surgido durante su programación y las decisiones de diseño que se han tomado para solventarlos.

Tras esto se muestra una serie de experimentos y las conclusiones junto con los trabajos futuros y cómo se podrían implementar estos trabajos futuros.

Al final del documento se añaden los apéndices con más información teórica que se ha considerado menos relevante, la explicación de cómo ejecutar el programa y el manual de la interfaz desarrollada.

2. Cuerpo del trabajo

2.1. Introducción

En los últimos años la importancia de los dispositivos móviles se ha incrementado considerablemente, sus ventas se incrementan año tras año. Los fabricantes mejoran las características tanto *software* como *hardware* con cada nuevo dispositivo, sacando una gran variedad de productos sin que exista un diseño dominante. (Cecere, et al., 2015). Estas mejoras han permitido a los *smartphones* evolucionar siendo capaces actualmente de poder ejecutar aplicaciones en 3D en tiempo real. Esto les permite ser excelentes candidatos para poder realizar aplicaciones de control y simulación de robots. Aunque la potencia de los dispositivos telefónicos y *tablets* sea considerablemente inferior a un ordenador de sobremesa, en determinadas ocasiones puede ser una mejor opción frente a un ordenador convencional. Estos dispositivos son más ligeros, pueden ser usados en cualquier entorno, tienen independencia de la red eléctrica, permiten al teleoperador una mayor libertad de movimientos. Se han utilizado con anterioridad estos dispositivos para realizar el control de sistemas robóticos mediante ROS usando, por ejemplo, dispositivos iOS. (Codd-Downey & Jenkin, 2015)

Como se ha mencionado anteriormente, debido a la gran producción de diferentes tipos de dispositivos y la inexistencia de un diseño dominante, actualmente en el mercado existe una gran variedad de dispositivos móviles muy diferentes entre sí. La variedad de tamaños de pantalla y de resolución hace que diseñar una interfaz que se adapte a todos sea muy complejo si no se usan las herramientas adecuadas. Unity3D simplifica esa tarea gracias al uso de canvas, que reescalan la interfaz, además de permitir el desarrollo de aplicaciones multiplataforma. Unity3D es un motor de videojuegos. Los motores de videojuegos permiten poder desarrollar simulaciones de manera más sencilla ya que incluyen físicas, carga de modelos 3D, cálculos de iluminación, aplicación de texturas sobre modelos, etc. Todo ello de manera optimizada y a tiempo real, algo imprescindible para poder realizar aplicaciones de control de sistemas. Los motores gráficos permiten a los sistemas robóticos poder crear entornos simulados. (Hu & Meng, 2016).

Actualmente existen diferentes maneras de poder desarrollar software para el control de robots, la manera más utilizada y demandada por las empresas es el uso del *framework* ROS ya que ha simplificado su programación y ha juntado en un único *framework* muchas funcionalidades indispensables. Este *framework* se creó para simplificar la tarea del

desarrollo de *software* de robots y actualmente es de código abierto (Quigley, et al., 2009). Las manos usadas Shadow y Allegro al igual que muchos sistemas robóticos tienen una versión del controlador en ROS. Es posible contactar con estos sistemas para enviar órdenes de control a los controladores de estos dispositivos.

Existen diferentes maneras de controlar un robot, aparte de usar un computador; se pueden usar un guante con sensores que reproduzcan el movimiento, un joystick con realimentación de fuerzas o las señales eléctricas del cuerpo, entre otras. Las señales mioelétricas (EMG) se suelen usar para el control de prótesis. Este campo ha ido evolucionando a lo largo de los años y actualmente es posible poder reconocer movimientos globales como el cierre y la apertura de una mano. Para el procesamiento de estas señales se puede usar Matlab. Este programa tiene extensiones para ROS, de esta manera es posible conectarse y enviar ordenes de control a un robot. El sistema desarrollado permite poder recibir esas señales ya procesadas para mostrar el movimiento que se ejecutaría en una mano robótica real y en el caso de estar conectado con una ejecutar el movimiento.

En el presente documento se desarrolla una aplicación multiplataforma que puede ser ejecutada en los sistemas operativos Windows Phone, Android e iOS, por lo tanto se puede usar tanto en *smartphones* como en *tablets*. Esta aplicación permite realizar el control de una o varias manos simultáneamente, pudiendo elegir entre la mano Allegro o Shadow. El control desde la aplicación se podrá realizar de varias maneras: ejecutando secuencias de movimientos ya creadas, seleccionando determinada articulación y modificando su valor en coordenadas articulares o mediante los valores obtenidos en Matlab de señales EMG.

Para realizar este proyecto se ha utilizado Unity3D para poder realizar la aplicación multiplataforma que se encarga de enviar las órdenes de control al robot, ya sea en una versión simulada o en la real. En esta aplicación se visualiza el movimiento de la mano en 3D. Además se posibilita el control utilizando varios clientes simultáneamente en un ambiente colaborativo. También recibirá las órdenes de control de abrir y cerrar la mano mediante señales EMG.

Se ha creado una estructura de nodos en ROS usando GNU/Linux para enviar los datos recibidos por los dispositivos telefónicos y realizar el control. También para recibir las posiciones actuales en las que se encuentra el sistema y enviarlas a la aplicación móvil para actualizar las posiciones del sistema.

El sistema posibilita el control usando señales EMG. Para ello las señales ya procesadas en Matlab envían los datos del resultado obtenido, que puede ser un movimiento de apertura o de cierre mediante la publicación de la información en un *topic* situado en el Master en GNU/Linux.

Para comunicar las órdenes enviadas desde el dispositivo se ha creado un cliente y un servidor que se conectan en una red local usando sockets TCP asíncronos.

En este trabajo no se tratará el procesamiento de la señal para saber si se trata de un movimiento o de otro. El módulo de procesamiento de la señal EMG se ha facilitado. Éste módulo publica en un *topic* de ROS la información de apertura y de cierre de la mano.

2.2. Marco teórico y estado del arte

2.2.1. ROS

ROS (*Robotic Operating System*, en castellano, sistema operativo de robots) es un *framework* desarrollado para simplificar la tarea de programar robots a través de una gran variedad de plataformas robóticas existentes. Está compuesto de una colección de herramientas, librerías y convenciones.

Antes de su creación el desarrollo de software de robots era bastante complejo, debido a que era necesario tener un gran conocimiento en muchos temas para ser capaces de resolver todos los problemas que podrían surgir y se tenía que reinventar la rueda cada vez que un laboratorio creaba un nuevo sistema robótico.

ROS comenzó en Stanford como una fusión de ideas sobre cómo solucionar los problemas de desarrollar un software que permitiese simplificar la programación de robots. En 2007 ROS se convirtió en una entidad formar a cargo del laboratorio Willow Garage. En el 2009 se redacta el primer artículo donde se presenta ROS como un *framework open source.* (Quigley, et al., 2009)

ROS actualmente se desarrolla en un entorno colaborativo, en el que se anima a que cualquier persona que tenga conocimientos en el mundo de la robótica pueda participar en la creación de código añadiendo más funcionalidades. Así pues, ROS es un proyecto de código abierto que ha sido desarrollado gracias a los esfuerzos de una gran comunidad internacional. Cualquier grupo de desarrolladores tiene acceso al código y puede encargarse del mantenimiento teniendo completo control sobre ROS, ya que utiliza una licencia BSD. Además, ROS ha sido desarrollado de manera modular así que los usuarios pueden elegir que paquetes usar de manera sencilla.

Existen diferentes versiones. Dependiendo de la versión de Ubuntu que se tenga será recomendable usar una versión de ROS u otra. Las versiones de ROS coinciden con las versiones del sistema operativo de Ubuntu y los EOL (*End of life*). Así pues, para la nueva versión de ROS lanzada el 23 de mayo de este año, Melodic Morenia, es recomendable usar la versión de Ubuntu Bionic Beaver (v.18.04) con cuya EOL coincide. Lo mismo sucede con Ubuntu Xenial Xerus (16.04) y Kinetic Kane de ROS y Ubuntu Trusty Tahr (v.14.04) con Indigo Igloo. Tenemos también versiones intermedias que coinciden con versiones menos estables de Ubuntu, como es el caso por ejemplo, de la versión Lunar Loggerhead

con Ubuntu Zesty Zapus (v.17.04). El resto de versiones de Ubuntu han llegado ya a su fecha de EOL por lo que no se seguirán actualizando.

ROS está de manera oficial completamente soportado para Ubuntu, sin embargo, no es la única opción de sistema operativo, actualmente está disponible de manera experimental en OS X, Android, Arch Linux, Debian Wheezy y OpenEmbedded/Yocto. Como proyecto en progreso se tiene ROSPod que permitiría poder ejecutar ROS en dispositivos iOS. Es recomendable usar una distribución de Ubuntu al ser las demás opciones experimentales o ser proyectos no acabados en caso de iOS. También es posible ejecutar ROS desde Matlab y tener acceso a los diferentes *topics* del sistema (Corke, 2015).

2.2.1.1. Arquitectura de ROS

Para poder comprender correctamente el funcionamiento de ROS y los siguientes apartados es importante tener en cuenta como está distribuido. Su arquitectura se puede dividir en tres secciones:

- Sistema de archivos
- Grafo computacional
- Comunidad

2.2.1.1.1. Sistema de archivos

En este apartado se van a explicar los archivos que componen un proyecto en ROS. El sistema de archivos de ROS está distribuido en carpetas, dentro de esas carpetas hay diferentes archivos que se encargan de describir sus funcionalidades

***Packages* (Paquetes):** Un paquete tiene la estructura y el contenido mínimo para crear un programa en ROS. Los paquetes están compuestos por archivos y carpetas. Para poder crear, modificar y trabajar con paquetes, ROS posee una serie de comandos que nos simplifican el proceso. (Véase 4.1. Comandos de terminal para ROS)

Manifests: Estos contienen información sobre el paquete; por ejemplo licencias, información, dependencias... Los manifiestos son gestionados con un archivo llamado *manifests.xml*.

***Stacks* (Pilas):** Los paquetes se organizan en pilas. Las pilas tienen la intención de simplificar el proceso de compartición de código. Es imprescindible que las pilas tengan una estructura básica de archivos y paquetes, estas se pueden crear de manera manual o mediante

una orden. Los tres archivos imprescindibles que debe tener una pila son: *CmakeList.txt, Makefile y stack.xml.*

***Stack Manifest*:** Corresponde al archivo *stack.xml*, proporciona información sobre una pila (*stack*), incluyendo la información de la licencia y las dependencias con otras pilas.

***Message (msg) types*:** Los mensajes son la información que un proceso manda a otros procesos. ROS posee muchos tipos de mensajes predefinidos y es posible crear nuevos siempre y cuando se encuentren dentro de la estructura de carpetas adecuada que es además donde se almacenan todos los mensajes de ROS (*nombrePaquete/msg/nombreDelTipoDeMensaje.msg*). Los msg tienen principalmente dos campos, los nombres de las variables y los tipos.

***Service (srv) types**:* Son las descripciones de los servicios, se encuentran guardadas en *nombrePaquete/srv/servicio.srv*. Los servicios definen las estructuras de datos petición/respuesta para la comunicación entre nodos.

2.2.1.1.2. Grafo computacional

Este apartado es muy importante puesto que se va a usar a lo largo del documento mucha terminología expuesta en él. ROS utiliza una arquitectura de grafo para interactuar con los sistemas, para ello crea una red donde todos los procesos están conectados. Cada nodo de esa red puede interactuar con otros nodos de la misma enviando y transmitiendo datos.

Nodes (Nodos): Los nodos son ejecutables que se utilizan para interactuar con otros nodos. Para ello utilizan el *parameter server, messages y topics*. Los nodos permiten separar las diferentes funcionalidades que se implementen permitiendo tener un código más sencillo. Los nombres de los nodos deben ser únicos, son los identificadores que se utilizan para comunicarse con otros nodos, por ese motivo, si se repiten los nombres se podrían producir ambigüedades en el sistema. Los nodos pueden estar escritos en C++, usando la librería roscpp, o en Phyton, con la librería rospy.

***Master*:** Proporciona toda la información para que los nodos sean identificados. Sin él los nodos no se podrían encontrar, ni podrían enviar mensajes o invocar servicios.

***Parameter Server*:** Permite guardar datos de manera centralizada, actualmente forma parte de Master.

***Messages (Mensajes)*:** Los nodos se comunican entre ellos a través de mensajes (msg) que son publicados por *topics*. Como ya se dijo anteriormente es posible crear tipos de mensajes personalizados o usar los ya existentes.

***Topics*:** Son los buses que utilizan los nodos para transmitir mensajes. Para su transmisión se pueden usar los protocolos TCPROS o UDPROS. Los nodos pueden suscribirse a un *topic* para recibir información o publicar información. Los mensajes en ROS están altamente tipados, lo cual quiere decir que para poder suscribirse a un nodo o publicar información es imprescindible tener el mismo tipo de mensaje. Los *topics* pueden ser transmitidos sin la necesidad de tener una conexión directa. Un *topic* puede tener varios subscriptores. En la Figura 1 se puede ver como dos nodos intercambian información al haber uno publicando en un *topic* y otro suscribiéndose.

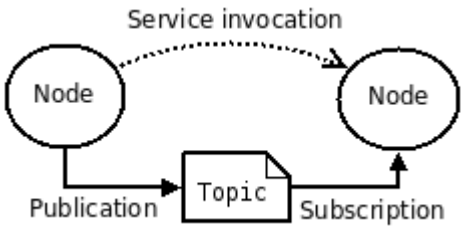


Figura 1: Esquema de paso de información entre nodos[1].

Services (Servicios): Cuando necesitas realizar una comunicación con un nodo y recibir una respuesta no puedes utilizar *topics*, puesto que estos no devuelven ninguna respuesta. Para ello debes utilizar servicios. Los servicios son creados por el usuario, por ello no existe ningún servicio estándar para los nodos. Como podemos observar en la Figura 1, cuando se invoca un servicio la comunicación de los nodos es directa a diferencia de cuando se utiliza un *topic*.

***Bags*:** Es un tipo de archivo que guarda información sobre mensajes, *topics*, servicios... Se puede usar para poder visualizar que es lo que ha sucedido en una ejecución, enviando los mismos datos al mismo tiempo en el que se produjo por primera vez. Se utiliza principalmente para depurar código.

[1] http://wiki.ros.org/ROS/Concepts

2.2.1.1.3 Comunidad

La comunidad juega un papel crucial para ROS. Es ella la que se encarga de desarrollar las nuevas distribuciones del sistema y le brinda un buen soporte. De este apartado podemos resaltar:

Las distribuciones: Como se ha explicado en Distribuciones ROS, son las diferentes versiones de ROS que han salido y que están disponibles para ser instaladas.

Repositorios: Estos permiten a los diferentes usuarios poder crear su propio robot y acceder a diferentes ejemplos de implementación de robots usando ROS.

La wiki de ROS: En esta wiki se puede encontrar mucha información sobre ROS, tanto tutoriales como explicaciones que han realizado diferentes personas de la comunidad.

Contacto por email con las personas implicadas en el desarrollo de ROS, para resolver cualquier duda, y disposición de páginas de respuestas como por ejemplo: ROS *answers*

Blogs: El blog de ROS aporta mucha información actualizada, incluyendo fotografías y vídeos.

2.2.1.2 Fichero de carga de robots URDF

Existen diferentes ficheros para realizar la carga de robots. En este caso nos vamos a centrar en la carga de modelos de robots utilizando URDF (*Undefined Robot Description Format).* Este archivo que contiene la descripción del robot está escrito en formato XML. Se puede ver más información sobre las etiquetas que lo conforman en el apartado 4.2. Entendiendo las etiquetas de un archivo URDF.

A pesar de que esta descripción de este archivo es lo más general posible, no permite poder describir a todos los robots. La principal limitación es que solo puede representar estructuras en árbol, así pues, los robots con estructuras paralelas no pueden ser representados. La especificación asume que un robot consiste en *links* rígidos conectados con *joints*; los elementos flexibles no están permitidos. Tampoco permite especificar la pose del robot sin tener en cuenta el mundo y no es posible cargar objetos que no sean robots o parte de ellos, como por ejemplo las luces. Cubre propiedades cinemáticas y dinámicas, la representación visual y el modelo de colisión del robot.

En resumen, URDF posee una estructura de árbol y son los *links* los que poseen la información de la transformación de los mismos, está compuesto por *joints*, *links* y "extensiones". Es uno de los archivos de descripción de robots más usados por ROS.

Existen dos formas de generar el código URDF, la primera escribiendo el código manualmente y la segunda utilizando Xacro. Xacro es un macrolenguaje que hace más sencillo el mantenimiento de archivos de descripción, aumentando su legibilidad y evitando duplicar información en los archivos. Es más útil usar xacros cuando trabajamos con un XML muy largo. Xacro se usa para simplificar los archivos URDF.

2.2.2. Unity3D

Unity3D es un motor de videojuegos que permite poder realizar de manera sencilla juegos en 2D y en 3D. Permite el uso de físicas, carga de modelos 3D, cálculos de iluminación, aplicación de texturas sobre modelos, entre otras cosas, todo ello de manera optimizada. Permitiendo poder ser ejecutado en sistemas con pocos recursos, siempre que se haya realizado una programación que lo permita.

Unity3D permite poder realizar la compilación para múltiples sistemas, facilitando la creación de aplicaciones multiplataforma. Además, gracias al uso de canvas, es posible poder realizar una interfaz de manera visual que luego puede ser reescalada a cualquier resolución siguiendo la configuración seleccionada. Para implementar las funcionalidades dentro de la aplicación permite la programación en dos lenguajes de programación C# o Javascript. También es posible añadir código nativo de distintas plataformas como *plugin*.

Para facilitar la programación y la puesta en marcha de los juegos en Unity3D posee una tienda de *assets*.

Unity3D presenta una gran cantidad de usuarios, además de una página oficial con tutoriales básicos para aprender a utilizarlo. Al tener una comunidad tan grande es sencillo encontrar soluciones para prácticamente cualquier problema, también se puede añadir nuevas dudas o si se encuentra un error en algún algoritmo que usa Unity3D avisar a los desarrolladores.

Existen diferentes licencias para poder usar Unity3D. Sólo es necesario adquirir una licencia de pago si se va a sacar provecho económico, dependiendo de la cantidad de dinero que se vaya a ganar.

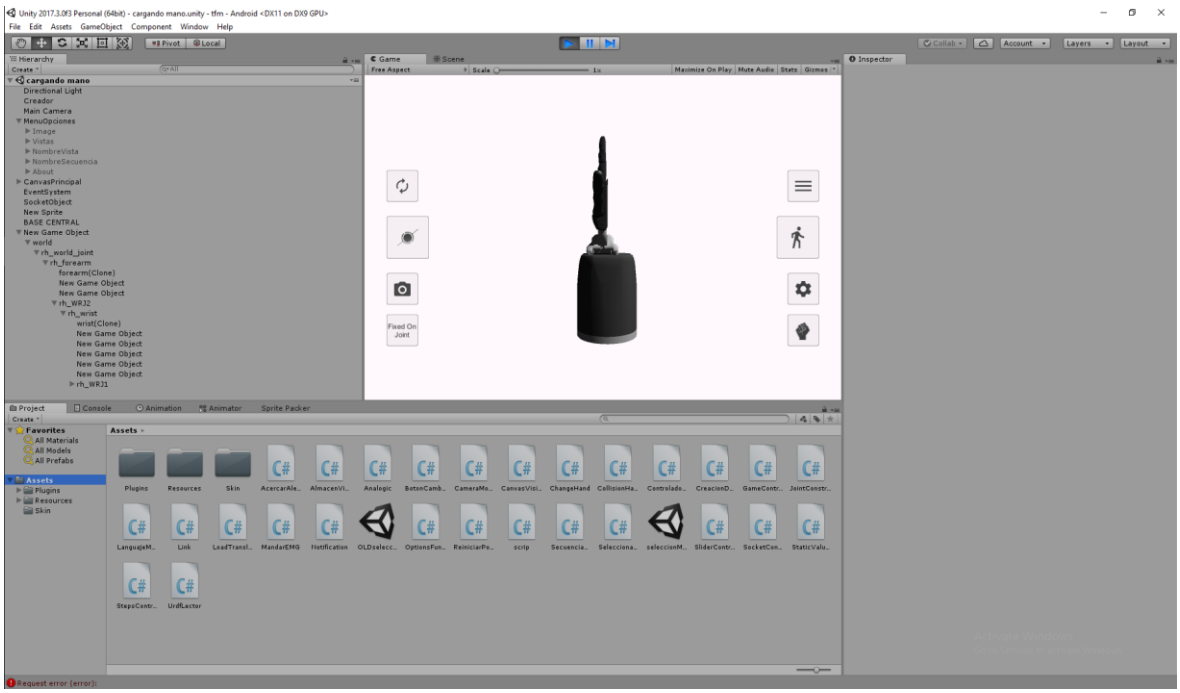


Figura 2: Interfaz con una escena cargada[2]

Un programa desarrollado en Unity3D está compuesto por diferentes escenas, en cada escena hay una serie de objetos (*GameObjects*, cámaras, elementos de interfaz, etc). Se puede ver un ejemplo de una escena cargada en Unity3D en la Figura 2. Unity3D compone los objetos en una estructura de árbol, donde los nodos superiores son los padres. Todos los elementos están dentro del nodo principal, la escena. Los elementos de interfaz como, por ejemplo, botones deben ir obligatoriamente dentro de un canvas para poder ser visualizados. Los *GameObjects* son los elementos principales de Unity3D.

Actualmente se puede desarrollar software en Unity3D en Windows, MacOs y desde hace un año aproximadamente también en GNU/Linux, dónde se recomienda Ubuntu.

[2] Fuente propia

2.2.3. Allegro Hand

Allegro Hand (Figura 3) es una mano robótica antropomórfica de bajo coste que posee cuatro dedos, uno de ellos pulgar. Esta mano ha sido desarrollada por SimLab, utiliza la tecnología para manos robóticas humanoides desarrollada en el instituto de corea de Tecnología Industrial KITECH . Esta mano puede ser usada en dos sistemas operativos, en Windows usando la aplicación AHAS (*Allegro Hand Application Studio*) y en Linux usando ROS y para su simulación Rviz.

Figura 3: Mano Allegro[3]

AHAS permite el control mediante una serie de sliders que hacen referencia a cada una de las articulaciones de la mano, además también vienen implementados en el programa algoritmos de agarre. Se puede ver la Interfaz de AHAS en la Figura 4. Cuando se ejecuta el programa se puede elegir entre la mano izquierda o derecha, además de poder ejecutar una mano virtual o una real.

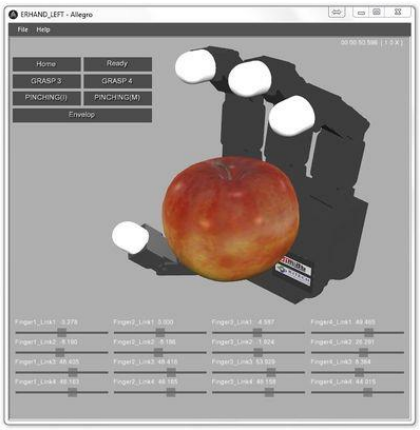


Figura 4: Interfaz AHAS[4]

[3] http://www.simlab.co.kr/Allegro-Hand.htm
[4]http://wiki.wonikrobotics.com/AllegroHandWiki/index.php/Allegro_Hand_Application_Studio_(User_Manual)

Cuando se ejecuta el controlador con simulación en ROS, se carga la simulación en Rviz y todos los nodos necesarios. Su interfaz visual es bastante similar al programa AHAS, en el sentido de usar deslizadores para enviar las órdenes de control al sistema, tal y como podemos comprobar en la Figura 5. Es posible controlar tanto una mano real como una simulada. Para el control se puede utilizar la opción de utilizar la GUI que permite el control a partir de una serie de sliders que representan el valor en radianes de la rotación de cada articulación o mediante la publicación de las posiciones en *topics*. Se pueden ver todas las opciones de lanzamiento de la mano Allegro Hand en ROS en el 4.5. Ejecución de la mano Allegro en Linux.

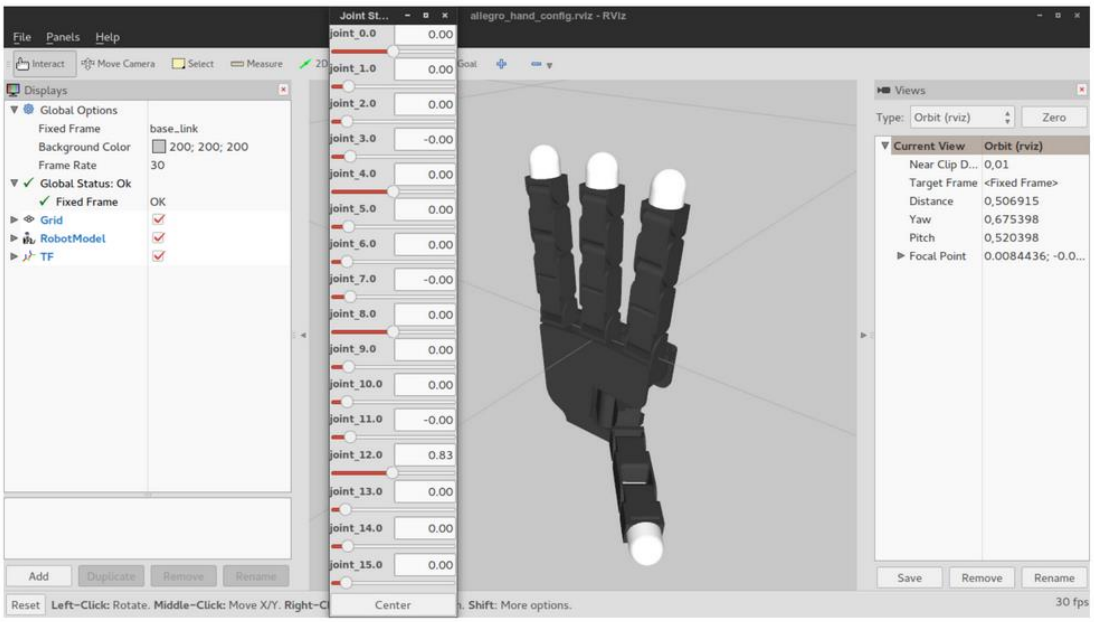


Figura 5: Ejecución de la mano Allegro en ROS[5]

[5] Fuente propia

En la Tabla 1 vienen representadas las características principales de la mano Allegro. En ellas aparecen datos como por ejemplo el peso del sistema, la capacidad de carga o los requisitos energéticos del sistema.

Tabla 1: Especificaciones técnicas[6]

Cantidad de dedos	4, 1 de ellos pulgar	
Número de articulaciones	4 x cada dedo = 16 activas	
Activación	Tipo	DC Motor
	Ratio de marchas	1:369
	Máx. torsión	1.70 (NM)
	Max. Vel. articulación	0.11 (seg/60 grados)
Peso	Dedo	0.17 (Kg)
	Pulgar	0.19 (Kg)
	Total	1.08 (Kg)
Codificación de articulación	Medición	Potenciómetro
	Resolución (nominal)	0.002 grados
Comunicación	Tipo	CAN
	Frecuencia	333 (Hz)
Carga	5 (kg)	
Requisitos energéticos	7,4VDC (7,0V-8,1V), 5A Mínimo	

[6] http://www.simlab.co.kr/Allegro-Hand.htm

En la Figura 6, se muestra la distribución de los motores y los componentes dentro de cada pieza de la mano Allegro, para dar una mayor comprensión del sistema.

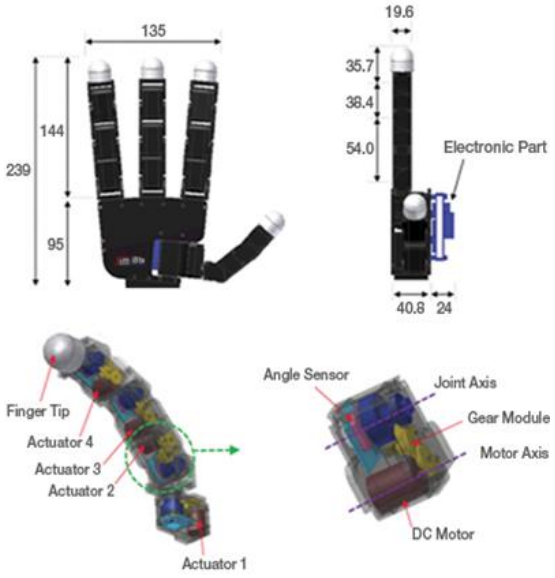


Figura 6: Descripción Allegro Hand[7]

La Figura 7 representa la dirección de rotación de las articulaciones de la mano derecha y de la izquierda.

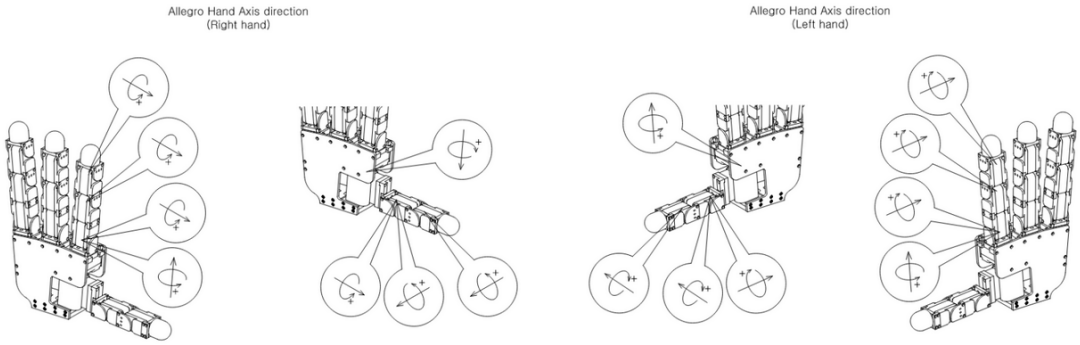


Figura 7: Descripción de las rotaciones[8]

[7] http://www.simlab.co.kr/Allegro-Hand.htm
[8] http://www.simlab.co.kr/Allegro-Hand-download/Allegro-Hand-Joint-infomation.pdf

2.2.4. Shadow Dexterous Hand

La mano robótica Shadow Dexterous Hand es una mano antropomórfica creada por el laboratorio Shadow. Esta mano tiene características similares a las manos reales humanas. Consta de 5 dedos uno de ellos pulgar y 24 grados de libertad. (Figura 8)

Figura 8: Mano Shadow[9]

Existen diferentes maneras de controlar la mano Shadow, el laboratorio Shadow utiliza ROS para realizar el control y es la herramienta oficial, por lo que se utiliza principalmente en GNU/Linux. Existe dos versiones de la mano Shadow, la mano que se encuentra a la izquierda en la Figura 8 utiliza un sistema eléctrico llamado "Smart Motor" la de la derecha un sistema neumático "Air Muscle". Todas las versiones utilizan el bus EtherCAT para realizar la comunicación.

El sistema "Smart Motor" se caracteriza por disponer del control de posición y de fuerza, electrónica de accionamiento de motor, caja de cambios, sensor de fuerza y comunicaciones en un módulo compacto, 20 de los cuales están empaquetados en la base de la mano.

El sistema "Air Muscle" permite controlar la presión y la posición del robot, posee electrónica para el accionamiento de válvula, colector, detección de presión y comunicaciones para 80 válvulas en la base de la mano.

[9] https://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E_20150827.pdf

En la Figura 9 se muestran las medidas de los dos tipos de mano Shadow. A la izquierda se tiene la "Smart Motor" a la derecha la "Air Muscle". En el laboratorio se dispone de la mano en versión Smart Motor.

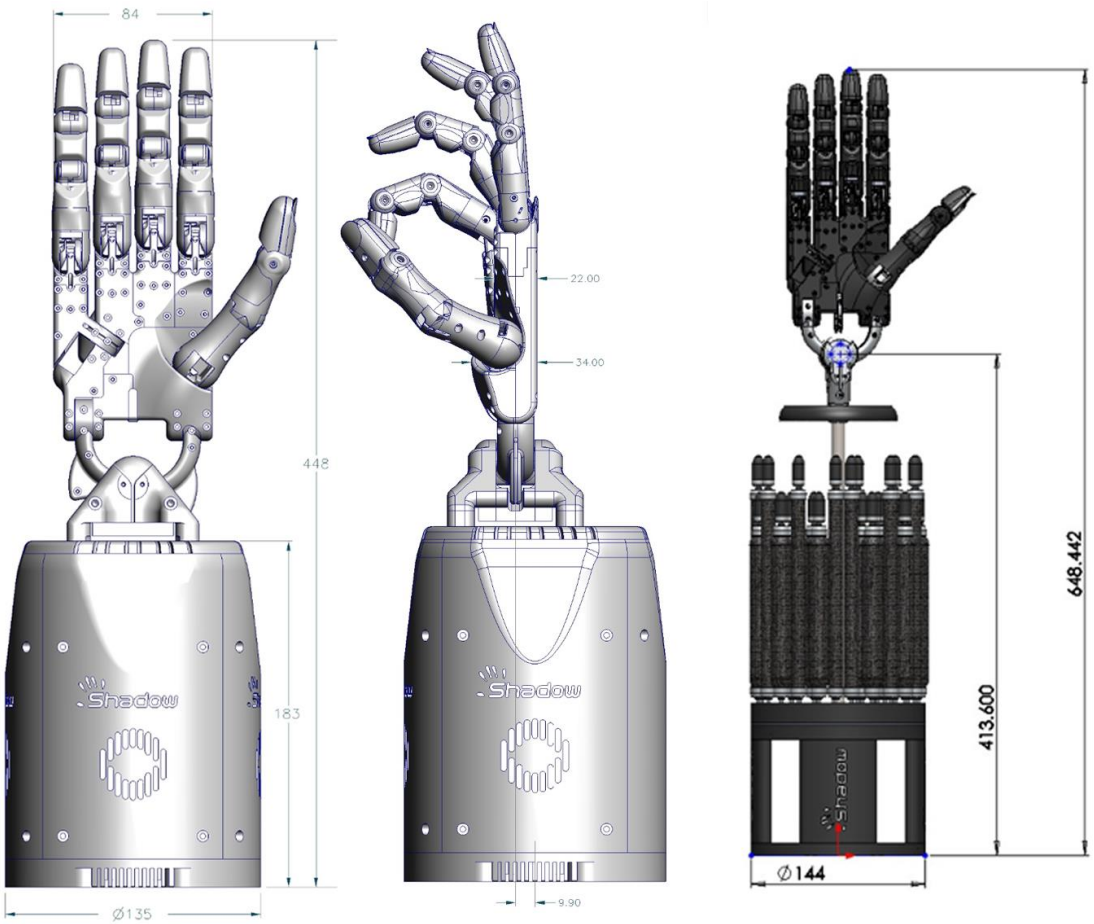


Figura 9: Medidas de las manos Shadow[10]

[10] http://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E1_20130101.pdf

En la Figura 10 muestra su diagrama cinemático.

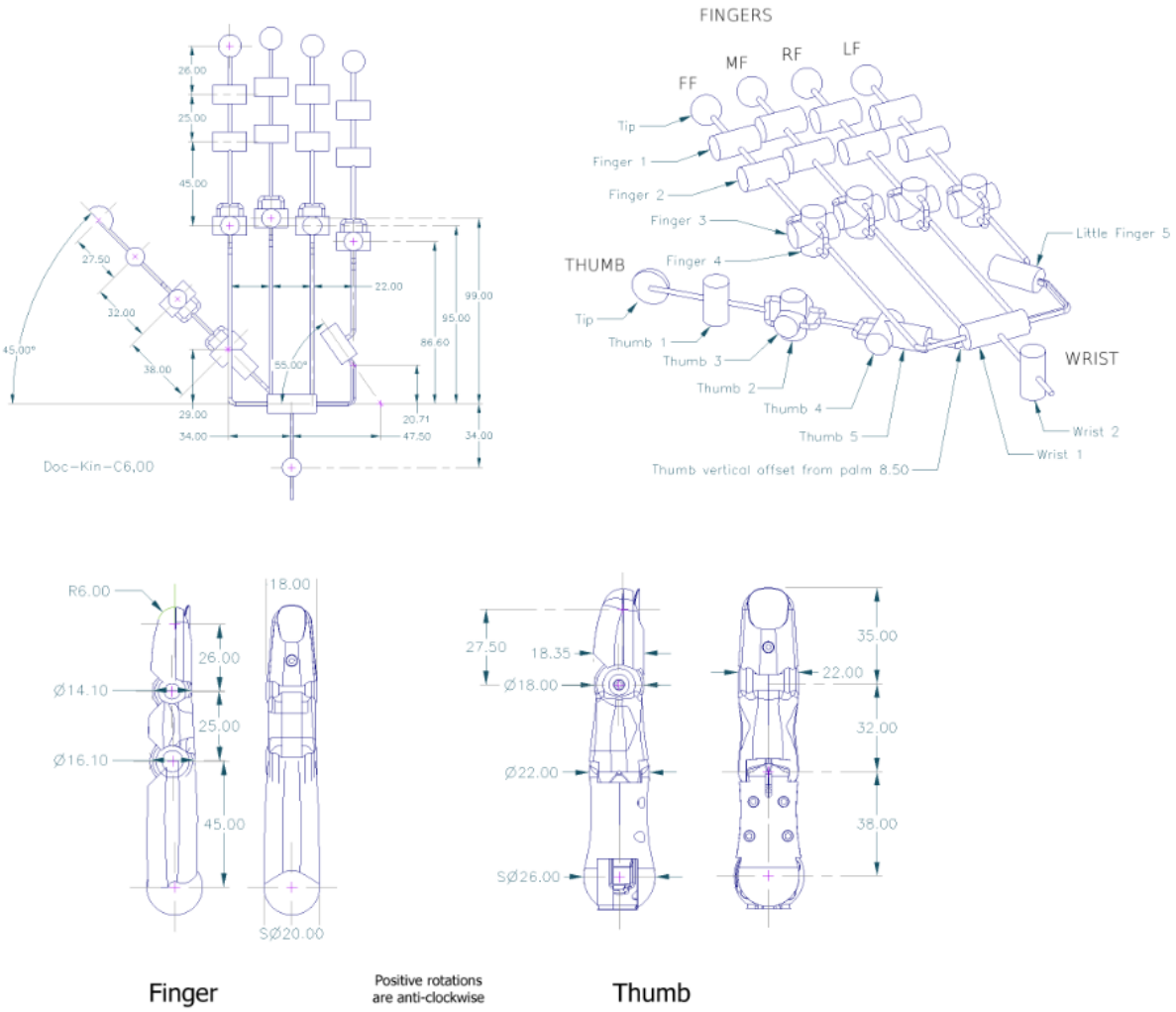


Figura 10: Esquema cinemático de la mano Shadow[11]

[1111] https://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E_20150827.pdf

La cinemática de la mano Shadow ha sido optimizada para ser lo más parecida posible a una mano humana en la Tabla 2 se presentan el giro máximo y mínimo en grados y en radianes que puede realizar cada articulación de la mano. Para saber a qué articulación hace referencia cada una de las mostradas en la tabla nos podemos fijar en la Figura 10, donde TH hace referencia a THUMB, WR es WRIST y el resto de abreviaturas aparecen en la imagen

Tabla 2: Torques mínimos y máximos por articulación[12]

Articulaciones	Grados		Radianes	
	Min	Max	Min	Max
FF1, MF1, RF1, LF1	0	90	0	1.571
FF2, MF2, RF2, LF2	0	90	0	1.571
FF3, MF3, RF3, LF3	0	90	0	1.571
FF4, MF4, RF4, LF4	-20	20	-0.349	0.349
LF5	0	45	0	0.785
TH1	0	90	0	1.571
TH2	-30	30	-0.524	0.524
TH3	-12	12	-0.209	0.209
TH4	0	70	0	1.222
TH5	-60	60	-1.047	1.047
WR1	-40	28	-0.698	0.489
WR2	-28	8	-0.489	0.140

[12] https://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E_20150827.pdf

El dedo pulgar tiene cinco grados de libertad y cinco articulaciones. Cada dedo tiene tres grados de libertad y cuatro articulaciones. Las articulaciones distales varían un +/- 1º.

A continuación se presenta la Tabla 3, en esta tabla se realiza un pequeño resumen de las características de las manos Shadow.

Tabla 3: Especificaciones técnicas[13]

Cantidad de dedos	5, 1 de ellos pulgar y otro meñique	
Número de articulaciones	24 articulaciones	
Peso	De la mano	4.3 kg
	Carga	5 kg
Velocidad	Motor Hand	1.0 Hz
	Air Muscle Hand	0.2 Hz
Comunicaciones	Tipo	EtherCAT
	Velocidad	100Mbps
Requisitos energéticos	Motor Hand	48V @ 2.5ª
	Air Muscle Hand	48V @ 1.0A
Consumo Air Muscle Hand	Aire comprimido (filtrado y libre de aceite) @ 3.5bar	
	Máximo consumo	24litros/min

[13] Información extraída de: https://www.shadowrobot.com/wp-content/uploads/shadow_dexterous_hand_technical_specification_E_20150827.pdf

2.2.5. Electromiograma (EMG)

Las fibras musculares generan actividad eléctrica a nivel celular cuando son activadas. Esto quiere decir que cuando se produce un proceso de contracción a nivel muscular, los potenciales de acción son transmitidos a la superficie de la piel permitiendo capturar la variación de energía transmitida usando electrodos. La principal característica de una señal mioelectrica es que la amplitud de la señal es estocástica y aleatoria por naturaleza y puede tener una distribución gaussiana. Los rangos de la señal están entre 0 y 10 milivoltios. La mayor componente energética está limitada entre 0 y 500 Hz de frecuencia, la energía dominante está concentrada entre 50 y 150 Hz (Consorcio OPENSURG Proyecto iberoamericano para la docencia e investigación en robótica médica utilizando recursos de código abierto Acción CYTED 509AC0372, 2013)

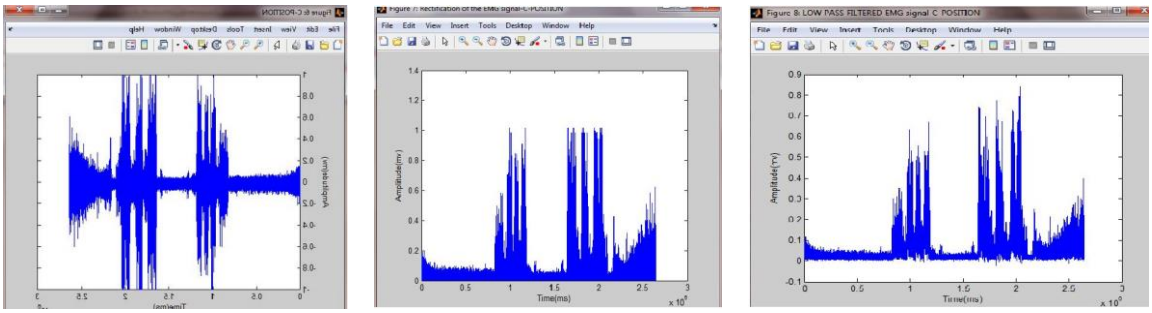
Las señales obtenidas son analógicas y son muy propensas al ruido. El ruido puede ser electromagnético y será mayor si se han recogido las muestras mediante electrodos situados sobre la piel que si se han recogido con métodos más invasivos. También se puede recibir en la lectura interferencias con otros procesos biológicos como puede ser, por ejemplo, el latido del corazón. Se puede eliminar el ruido mediante el uso de filtros analógicos, digitales, pasa-baja, pasa banda o rechaza banda. Este tipo de filtros se construyen con amplificadores.

Las señales EMG son señales analógicas. Las señales analógicas son aquellas que pueden tomar todos los valores posibles en un intervalo de tiempo, mientras que las digitales sólo pueden tomar dos valores. Para recopilar las señales analógicas se utiliza una frecuencia de muestreo que indica la cantidad de veces que se toman muestras de la señal por segundo.

Los filtros pasa baja se usan para eliminar las frecuencias superiores a 500 Hz que es el máximo de frecuencia que devuelven las señales EMG. El filtro pasa alta permite eliminar aquellas frecuencias que estén por debajo de los rangos mínimos establecidos 0 en este caso. El filtro rechaza banda permite eliminar los armónicos que pueden haberse producido.

Un ejemplo de procesamiento EMG se muestra en el artículo (Gaikwad & Sardeshmukh, 2014) que tiene como intención describir el proceso de la detección de diferentes gestos de la mano predefinidos utilizando una red neuronal artificial. En la Figura 11 se muestra la captura de señal EMG cuando se está realizando la letra C en lenguaje de signos. En la Figura 11a aparece la señal cuando es captada y en las Figura 11b y c se aplican los filtros pasa alta y baja en serie para formar un filtro pasa banda. De esta manera se consigue atenuar el ruido,

sin embargo es difícil eliminar el ruido cuya frecuencia se encuentra dentro del rango de la señal EMG.



a)Señal captada *b)Señal rectificada* *c)Señal filtrada*

Figura 11: Filtrado de señal EMG. a) Señal capturada sin realizar ninguna modificación sobre ella, b) Señal con rectificación con filtro pasa alta c) Señal con filtro pasa baja[14]

Otra técnica usada en el caso de extraer características frecuenciales es la digitalización de las señales de analógico a digital y adquirirlas en un ordenador para utilizar algoritmos y filtros digitales.

Para resumir, para la captación de las señales se utiliza un sensor que transforma la medición física en una salida eléctrica, este sensor no debe afectar a las propiedades y características de la señal que se está midiendo. Después de haber detectado la bioseñal ésta es amplificada para incrementar su amplitud y filtrada para eliminar el ruido o para compensar las distorsiones causadas por el sensor. Estos dos pasos pueden ser usados también para ajustar la señal a las especificaciones del *hardware* de determinado sistema. Las señales continuas deben ser limitadas a cierta banda antes de ser digitalizadas. La digitalización es el penúltimo paso, en éste la señal es procesada y convertida en una señal discreta que puede ser almacenada y procesada por un ordenador. El último paso consiste en el procesamiento por parte del ordenador para identificar el tipo de movimiento que se está realizando mediante diferentes algoritmos.

Una vez identificado el movimiento se puede utilizar para realizar el control de un sistema robótico. Es en este último paso en el que se centra el proyecto realizado, en el uso de las señales EMG para controlar manos robóticas a partir de movimientos predefinidos.

[14] https://www.semanticscholar.org/paper/Sign-Language-Recognition-Based-on-Electromyography-Gaikwad-Sardeshmukh/0a6ae3131db4e6f24c4af105d1a92e6fefec2811

2.2.6. Desarrollo de una interfaz Hombre-Robot

Para el desarrollo de cualquier interfaz en general hay que tener en cuenta la usabilidad y la accesibilidad de la aplicación. La usabilidad asegura que el sistema es fácilmente comprensible por el usuario. La accesibilidad posibilita a cualquier persona, independientemente de problemas visuales o físicos, el uso de la aplicación. Esto permite poder llegar a un mayor rango de población.

Actualmente existen varias normas para desarrollar interfaces. Estas normas han contribuido a la creación de interfaces más accesibles y usables. Sin embargo, seguir esas normas generales no asegura el correcto entendimiento por parte de los usuarios, ya que la distribución escogida, los iconos seleccionados, entre otras cosas, pueden resultar poco intuitivos para los usuarios finales. Es por ello, que se necesita probar la interfaz sobre el grupo de usuarios, para poder recibir *feedback*.

Son muchos los artículos ambientados en la importancia de un buen diseño aplicado a cualquier campo (Adams, 2002). El uso de una buena interfaz simplifica el trabajo del usuario, teniendo un acceso más rápido y una menor curva de aprendizaje a la hora de utilizar el *software*.

Para el desarrollo de la interfaz realizada en el proyecto se ha intentado realizar un diseño centrado en el usuario para la interacción humano-robot.

Para poder mostrar esto, se añaden en la experimentación pruebas realizadas sobre un grupo de usuarios.

2.2.7. Aplicaciones relacionadas

En este proyecto se ha propuesto la realización un sistema que permite realizar el control y la simulación mediante *smatphones* de varias manos robóticas comunicándose con los controladores en ROS. Para ello se ha comunicado el sistema mediante sockets TCP/IP utilizando una arquitectura cliente-servidor. Para la implementación de la simulación se ha utilizado el motor de videojuegos Unity3D. Además se permite el control de manos robóticas mediante señales mioelectricas. En este apartado se muestra una serie de aplicaciones para el uso de motores gráficos y señales mioelectricas. También se ilustra algunas de las opciones que hay actualmente para realizar la simulación y control de manos robóticas. Finalmente se hará una comparativa entre los sistemas expuestos y lo que se ha desarrollado en este proyecto.

2.2.7.1. Uso de motores gráficos para el desarrollo de aplicaciones robóticas

El uso de motores gráficos de videojuegos para el desarrollo de simuladores permite poder desarrollar proyectos con un aspecto más realista en un menor tiempo y de una manera más optimizada. En el ámbito de la robótica se han desarrollado simuladores con entornos muy variados. Un ejemplo de esto se ve en (Craighead, et al., 2008), donde se implementa diferentes entornos simulados para realizar rescates usando diferentes tipos de robots. Los entornos en los que se desarrolla esta aplicación son entornos acuáticos, terrestres y aéreos. En este simulador también se integran los sensores correspondientes.

En cuanto a entornos virtuales para múltiples drones, cabe destacar el uso de Unity3D. Un ejemplo de aplicación se muestra en (Meng , et al., 2015) donde se utiliza este motor para simular una ciudad y utilizar diferentes UAVs.

En otros artículos se utilizan Unity3D para realizar el seguimiento de rutas combinando Matlab (Andaluz, et al., 2016)

Es importante decir que Unity3D no es el único motor de videojuegos gratuito usado para la realización de simulaciones, otro ejemplo de ello es el uso del motor Unreal Engine. En el artículo (Hu & Wei, 2016) se realiza una comparativa entre estos dos motores para realizar un simulador para vehículos aéreos que pueda comunicarse con ROS, actualmente sigue siendo más sencillo el uso de Unity3D debido a su amplia comunidad y tener una mejor curva de aprendizaje.

Los motores de videojuegos permiten la conexión con servidores de maneras muy variadas como puede ser el uso de sockets (Meng , et al., 2015) o el uso de rosbridge (Codd-Downey, et al., 2014) el cual proporciona una API JSON para conectar un sistema ROS con otro tipo de sistemas. Existen actualmente otros métodos que pueden ser explotados mediante Unity3D para el control de robots mediante ROS en dispositivos Android e IOS. Para ello se podría añadir como plugin para usar código nativo en Unity3D ROS Android o ROS Pod. Sin embargo, estos sistemas no son estables actualmente.

2.2.7.2. Simulación y control de manos robóticas

La simulación es de vital importancia en los sistemas robóticos. Simular nos permite poder utilizar estos sistemas sin tener que exponerlos a errores, permitiendo probar diferentes algoritmos y situaciones sin poner en riesgo la integridad del robot.

Para realizar las representaciones tridimensionales que nos permiten poder llevar manos simuladas a un ordenador es necesario analizar las características de las manos humanas (Hirt, et al., 2012), (Virgala, et al., 2014). Las características extraídas de ese análisis pueden ser usadas para diseñar nuevas manos robóticas (Ramaswamy & Deborah, 2015) proporcionándoles una similitud con las manos reales.

Actualmente existen diferentes programas que nos permiten simular el control de manos robóticas. Por ejemplo, el framework synGrasp que es una tolbox en Matlab que se usa para manos subacotadas. Para programar dispositivos para manos robóticas se tiene, por ejemplo, el framework Hands.dvi (Salvietti, et al., s.f.). Gazebo (Aguero , et al., 2015), el cual puede ser usado en ROS, permite poder simular cualquier tipo de dispositivo y entorno. Uno de sus principales usos es la simulación de agarres de manos robóticas al usarlo junto con GraspIt (Aguero , et al., 2015). Otra herramienta de simulación es Rviz que permite el uso de MoveIt para realizar movimientos o planearlos.

En la mano Allegro se utiliza para la simulación Rviz, para el control se ejecuta un nodo que contiene una serie de deslizadores que hacen referencia a las articulaciones de la mano, al moverlo, se hace una rotación en dichas articulaciones utilizando como referencia el modelo cargado en el URDF. En el caso de la mano Shadow se utiliza para la simulación Gazebo y Rviz con MoveIt. MoveIt permite poder situar los dedos en determinada posición y planear y enviar la orden a la mano para que se sitúe en determinada posición. El movimiento planificado para llegar a determinada posición se utiliza para situar al robot real o al representado en Gazebo en determinada posición.

Existen diferentes métodos actualmente para el control de sistemas robóticos como es por ejemplo el uso de realidad aumentada (Green, et al., 2008) la cual permite poder mezclar la información real con la virtual. También es posible el control de manos utilizando guantes con sensores que permitan la colocación de la mano imitando una mano actual, o se puede usar visión artificial (Iliukhin , et al., 2017) reproduciendo el movimiento de una mano real humana. El uso del control mediante EMG permite poder realizar movimientos en una mano

robótica mediante la electricidad que genera el cuerpo, es por ello que este tipo de sistemas es muy interesante puesto que es aplicable al uso de control de prótesis robotizadas.

2.2.7.3. Aplicaciones de control mioelectrico

La electromiografía tiene muchas aplicaciones clínicas y en biomedicina. Este tipo de señales se utiliza como herramienta para poder diagnosticar enfermedades neuromusculares y trastornos del control motor (Consorcio OPENSURG Proyecto iberoamericano para la docencia e investigación en robótica médica utilizando recursos de código abierto Acción CYTED 509AC0372, 2013).

Se han utilizado también para intentar reconocer determinadas posturas mediante la lectura de las señales, como puede ser el reconocimiento de determinadas letras del abecedario. (Gaikwad & Sardeshmukh, 2014)

En el mundo de la robótica se utilizan para controlar prótesis. (Graupe & Cline, 1975). El uso de prótesis permite a las personas que han perdido una extremidad poder volver a realizar agarres recuperando la independencia tras la amputación. También es posible aplicarla a la teleoperación de robots en sistemas industriales.

2.2.7.4. Aportaciones con respecto a los estudios expuestos anteriormente

La aportación de este proyecto con respecto a los estudios expuestos anteriormente es la realización de un sistema multiplataforma que permite el control de una o varias manos robóticas simultáneamente utilizando uno o más dispositivos para enviar órdenes en un entorno colaborativo.

El sistema permite el control mediante métodos convencionales, envío de órdenes directamente a la mano o utilizando señales EMG. La posibilidad de poder enviar señales de control utilizando este sistema permite poder utilizarlo para el entrenamiento de prótesis. Además, el hecho de permitir cargar una textura de mano real da al sistema un valor añadido, puesto que permite poder visualizar el posible resultado final al utilizar una prótesis.

2.3. Objetivos

Los objetivos de este proyecto son los siguientes:

- Crear el control ambidiestro de las manos Shadow y Allegro.
- El control se podrá realizar de diversas maneras: Ejecutando secuencias, enviando las posiciones articulares deseadas al sistema con los nodos reales o bien usando señales EMG.
    - Mediante señales EMG se podrán recibir dos movimientos
        - Apertura
        - Cierre
- Los movimientos de apertura o de cierre enviados también podrán ser ejecutados desde la interfaz desarrollada.
- Cargar los modelos 3D en la aplicación de Unity3D interpretando los datos que vienen en los archivos URDF utilizados en ROS y que sea capaz de poder cargar cualquier otro robot que se pase usando los modelos.
- Aplicar texturas de mano real a los modelos de las manos cargadas.
- Comunicar el sistema de Unity3D con los nodos en ROS para el control y enviar las posiciones actuales del sistema a los clientes suscribiéndose al nodo que corresponda.
- Crear una estructura cliente-servidor.
    - El servidor será el encargado de comunicarse con los nodos de ROS mediante suscripción y publicación. Enviará las posiciones actuales de la mano a los clientes y los datos que reciba de los clientes se usarán para poder realizar el control del sistema,
    - El cliente será el encargado de enviar las posiciones de la mano y mantendrá la posición de las manos actualizadas en todo momento.
- Adaptación de los mensajes que se transmiten entre el cliente y el servidor.

Como extras se han implementado las siguientes partes:

- Crear una interfaz capaz de ser reproducida tanto en dispositivos nuevos como en antiguos para no obligar al cliente a tener que comprarse la última tecnología.

- Desarrollo de una aplicación multiplataforma.
- Mejorar el sistema de detección de colisiones de la mano.
- Depuración de errores detectados en el proyecto anterior.
- Mejorar la red del sistema anterior.
- Permitir un control colaborativo del sistema en el que varios usuarios pueden enviar órdenes a la mano.
- Mejorar la cámara del sistema anterior para evitar problemas a la hora de poder manejarla correctamente.
- Posibilidad de poder centrar la cámara en un *joint* específico.
- Mejora de las secuencias para que las que sean creadas con una mano puedan ser reproducidas por las dos manos simultáneamente.
- Control de dos manos simultáneamente.
- Sistema de notificaciones para avisar al usuario de que se ha cambiado algún parámetro y enviar mejor *feedback* mejorando la experiencia de usuario.
- Modificaciones sobre la interfaz para hacer caso a las opiniones de los usuarios que usaron la aplicación con el trabajo de final de grado.
- Modificación sobre los datos guardados para tener en cuenta la mano Shadow.
- Cambio de las distancias mínimas para la carga de la mano Shadow.
- Búsqueda del centro, la cámara apunta al centro de las posiciones de los *joints* en lugar de al primero *joint* encontrado.
- Cambio en la carga del árbol de Unity3D.
- Cambio de la estructura de los ficheros de guardado usados en el trabajo de final de grado.
- Solución de errores encontrados en el sistema.
- Optimización de código implementado con anterioridad.

En la aplicación desarrollada el año pasado se tenía un sistema que permitía el control mediante envío de secuencias o de órdenes directas a la mano Allegro tanto la izquierda como la derecha utilizando los nodos de ROS. Utilizando esta aplicación de base se han ampliado las funcionalidades, se han solucionado errores, se han mejorado las funcionalidades existentes y se han optimizado. Actualmente se tiene un sistema capaz de poder controlar de manera simultánea dos manos o poder utilizar una en concreto. Además del uso de la mano Shadow dentro del sistema. Permite el control de las manos robóticas a partir de señales EMG interpretadas.

2.4. Metodología

Para este proyecto se ha utilizado una metodología ágil, debido a que el tamaño del mismo no debe ser muy grande, se le debe dedicar el tiempo de 12 créditos. Este tipo de metodología permite poder cambiar los requisitos rápidamente durante el proceso.

Al ser desarrollado de manera individual no se va a aplicar una herramienta de proceso en su totalidad. Se ha decidido por comodidad utilizar Kanban pero con modificaciones.

Todas las tareas han pasado por los puntos de: investigación, diseño, implementación, pruebas y finalización. La pizarra Kanban se ha hecho tanto en una pizarra real como en un bloc de notas, en el que estaba apuntada la lista de tareas y el estado en el que se encontraba cada una. A modo de recordatorio se ha utilizado la aplicación Todoist para asignar fechas de finalización de tarea.

A continuación se describe el orden en el que se han realizado las tareas.

En el sistema lo primero que se ha implementado es la carga de la mano Shadow. Para ello primero se ha adecuado el sistema para tener las opciones de carga y por último se ha modificado el lector URDF del sistema para que admitiese los datos de la Shadow, estos al no seguir el mismo orden de carga que la mano Allegro ha tenido que ser modificado, también se ha cambiado la forma en la que se leen los datos para que sea más óptima. Lo siguiente que se ha hecho cargar dos manos simultáneamente en el cliente.

Al cargar las dos manos y tener tanto la mano Allegro como la Shadow ha sido necesario adecuar el resto del proyecto para que funcionase correctamente. Así pues se han modificado las secuencias, las colisiones, la cámara, su movimiento, el menú de vistas. Tras esto se ha añadido la posibilidad de poder centrar la cámara sobre determinado *joint* dentro de la lista de *joints*.

Una vez hecho esto y solventado todos los errores asociados a añadir una nueva mano, se ha pasado al servidor para asegurar el control de la mano Allegro. Para ello se ha cambiado la estructura de los paquetes que se envían y se reciben para poder decir si se trata de una mano derecha, una izquierda o las dos. Se han suscrito a los nodos necesarios y se envía la posición. Sabiendo que la mano Allegro funciona correctamente se ha pasado a la realizar lo mismo con la mano Shadow.

Para Shadow ha sido necesaria cambiar cada cuanto se envían los datos de las secuencias además de cuando se está modificando la posición del *joint*. También ha sido necesaria la

creación de un nodo en python para comunicarse con la mano usando el planificador que se da. Y solucionar algunos problemas asociados a los límites. Tras esto se ha empezado a implementar la recepción de datos mediante envío de señales EMG. Se ha creado en la aplicación los movimientos para la mano Shadow y Allegro de cerrado y apertura, asegurando que esos movimientos son posibles.

Una vez terminado estos pasos se ha introducido la carga de un *shader* de mano realista optimizado para dispositivos telefónicos. Al introducir el *shader* ha sido necesario hacer una modificación sobre la iluminación de la escena y las sombras.

Por último se han realizado una serie de arreglos de errores en el cliente y la implementación que posibilita la ejecución de movimientos en las dos manos de Shadow.

2.4.1. Herramientas usadas

En este apartado se describen las herramientas *hardware* y *software* que han sido utilizadas. También se mencionan los lenguajes de programación que se han utilizado. En las herramientas *hardware* no se tiene en cuenta los dispositivos en los que se ha testeado la aplicación, sólo con aquellos con los que se desarrolló.

A continuación se muestra una lista con el hardware usado.

- Ordenador portátil MSI
- Ordenador de sobremesa
- Smatphone Huawei P9 Lite
- Mano Allegro
- Mano Shadow
- Mac (De la EPS) para poder compilar para iOS

Software utilizado

- Unity3D
- ROS
- Windows 10
- Linux Ubuntu Xenial Xerus
- Git

- SourceTree
- Bitbucket
- Todoist
- Bloc de notas
- Word
- APK de Android
- Excel
- Matlab ROS
- Qt creator
- XCode

Lenguajes de programación utilizados

- C++
- C#
- Python

2.5. Descripción del sistema

En este apartado se describirá el sistema implementado, vamos a dividir esta sección en tres partes: la implementación en Unity3D, la implementación en el sistema que tiene comunicación con ROS y finalmente la interconexión entre estas dos partes.

Es posible obtener más información sobre cómo se ejecuta el sistema en el Manual de usuario situado en 4.4. Manual de la aplicación

2.5.1. Implementación en Unity3D

Unity3D separa el sistema en escenas, tal y como se vio en el apartado 2.2.2. Unity3D. En este caso de disponen de dos escenas. En la primera se hace la selección general con la que se va a trabajar y la segunda se utiliza para operar el robot.

a) Menú selección entre conectarse o no

b)Inserción de dirección IP

c)Selección entre manos

d)Selección entre mano izquierda, derecha o ambas

Figura 12: Primera escena de la aplicación a) Menú que permite conectarse al servidor o trabajar sin conexión b) Inserción de la dirección IP del servidor c) Selección entre la mano Allegro o la Shadow d) Selección entre manos izquierda, derecha o ambas)[15]

La primera escena te permite poder elegir entre las opciones que se muestran en la Figura 12. El orden es el siguiente: primero se pregunta si se quiere trabajar con o sin conexión

[15] Fuente propia

(Figura 12a), si se trabaja con conexión se pedirá la IP a la que se quiere conectar (Figura 12b). Una vez seleccionado si se trabaja con o sin servidor, en el caso de trabajar con servidor y haber introducido la IP se seleccionará la mano con la que se quiere trabajar, la Shadow o la Allegro (Figura 12c), a continuación elegimos el tipo de mano: Izquierda, derecha o ambas (Figura 12d).

Figura 13: Segunda escena de la aplicación muestra de funcionalidades implementadas cargando dos manos Shadow. a) Escena principal, lo que aparece nada más cargar. b)Menú de articulaciones c)movimiento mediante selección de joint d) Menú con el menú de vistas guardadas e) Menú con los pasos insertados para la creación de secuencias f)Menú con las secuencias guardadasa g)Selector para fijar la cámara en un joint h)Insercción de nombre i) Menú de opciones[16]

Una vez seleccionada la mano que se va a utilizar esta se cargará utilizando un lector de URDF. También se cargarán en el sistema el fichero con las preferencias del usuario, el fichero con la lista de secuencias almacenada y el fichero con las vistas almacenadas. Cuando se cargan los ficheros que contienen las vistas y las secuencias en el sistema en cada uno de los botones creados por cada secuencia o vista almacenada se guardan la información transformada y con referencia a los *gameObjects* correspondientes. Durante esta carga aparecerá una pantalla con la palabra cargando. Tras esto aparecerá la segunda escena, la

[16] Fuente propia

cual te permite poder realizar el control de las manos y tiene acceso a todas las opciones implementadas. En la Figura 13 se muestran todas las pantallas del sistema y se describe una a una cada una de ellas.

En la Figura 13a se muestra la escena principal, en cuanto se acaba de cargar los datos del lector URDF. En esa escena se muestran una serie de botones que son accesos a las diferentes funcionalidades implementadas. A continuación vamos a explicar brevemente a qué funcionalidades se llegan pulsando en estos botones. Vamos a separarlos en dos columnas y se explicarán de arriba abajo. Finalmente se expondrán brevemente algunas funcionalidades extra.

- Primer botón de la primera columna permite resetear las posiciones, es decir, las envía a un valor 0 o al valor más pequeño posible para dejar los dedos extendidos.
- El segundo botón de la primera columna nos envía al menú de articulaciones (Figura 13b), en este menú aparece una lista de todas las articulaciones con unos deslizadores que nos permiten cambiar los valores actuales de la mano. Dentro de este menú aparecen varios botones que hacen referencia a funcionalidades que están representadas en la Figura 13a con los mismos iconos.
- El tercer botón de la primera columna abre el menú de vistas (Figura 13d), el cual nos permite seleccionar una vista, eliminarlas o crear una nueva. Para crear una vista se tendrá que insertar el nombre cumpliendo con los requisitos que aparecen en la Figura 13h, los nombres no pueden tener espacios en blanco, no pueden contener el carácter '=' y no puede existir una vista con ese nombre introducido. Las vistas permiten guardar dentro del dispositivo telefónico la posición y el zoom actual de la cámara, así como el *joint* o articulación que se utiliza como pivote para realizar las rotaciones. De esta manera al seleccionar una de las vistas de este menú se cargan las posiciones de cámara almacenadas.
- El cuarto botón de la primera columna nos abre una lista con todos los *joints* actuales del sistema para fijar la cámara sobre ellos. (Figura 13g)
- El primer botón de la segunda columna muestra el menú de opciones. (Figura 13i)
- El segundo botón de la segunda columna se utiliza para acceder al menú de pasos. (Figura 13e). Desde este menú es posible acceder al menú de secuencias o crear un nuevo paso, representados con los mismos iconos del menú inicial, también es

posible eliminar todos los pasos realizados pulsando el botón superior central o el último únicamente pulsado el botón superior derecho. En la parte central de este menú hay una serie de números que hacen referencia a los pasos actuales del sistema. Para crear una secuencia de pasos se seleccionará el icono inferior izquierdo y aparecerá una ventana para insertar el nombre de la secuencia que se desea guardar, para guardarse debe tener al menos un paso almacenado, además de cumplir con las mismas especificaciones que cuando se guarda una vista. La ventana de guardar nueva secuencia será prácticamente igual a la que se muestra en la Figura 13h.

- El tercer botón de la segunda columna nos permite crear un nuevo paso para el sistema. Este botón es más grande puesto que se espera que sea una de las funcionalidades más usadas por parte de los usuarios.
- El cuarto botón de la segunda columna se utiliza para acceder al menú de secuencias creadas. (Figura 13f) Desde este menú es posible reproducir o eliminar las secuencias.
- El último botón de la segunda columna nos permite poder realizar movimientos de apertura o de cierre.
- Si se pulsa sobre una de las articulaciones de la mano aparecerá un deslizador para poder mover la articulación, tal y como se muestra en la Figura 13c.
- En esta ventana también es posible poder mover la cámara si se mantiene pulsado sobre la pantalla y se desplaza el dedo hacia uno de los lados. Se puede hacer zoom en el sistema acercando o alejando dos dedos situados sobre la pantalla.

Las secuencias, las vistas y las preferencias seleccionadas en el menú de opciones se guardan en el dispositivo para que no se pierdan los datos entre ejecución y ejecución.

Ahora que se ha dado un primer vistazo a la aplicación y a la interconexión entre las diferentes pantallas se hablará en detalle de las diferentes funcionalidades implementadas.

2.5.1.1. Carga del lector URDF

Después de realizar la selección de las manos se cargan los modelos a partir de los datos almacenados dentro de un fichero URDF.

Para la carga de los modelos en Unity3D tenemos que tener en cuenta que los valores de los ejes no son los mismos en Unity3D que en el fichero. Cuando se realice la carga de manera normal ésta nos aparecerá tumbada, ya que el eje –z=x.

Para pasar los datos a cuaterniones se han realizado determinadas consideraciones, los valores de los ejes en x, y z serán *rigth, down y back* respectivamente. Dependiendo de si se tiene un valor para el eje x o no se modificará el orden de multiplicación de los cuaterniones cuando se vaya a devolver los datos transformados a cuaterniones. En el caso de tener valor en el eje x según los valores del *axis* dentro del *joint* tenemos que multiplicar los datos poniendo en primer lugar el cuaternión con los datos de z, luego los de x y finalmente los de y. Cuando no se tengan datos para x el orden será x, y y z. (Figura 14)

El paso de información en x,y,z a cuaterniones se muestra a continuación (Figura 14):

```
public static Quaternion EulerToQuaterion(Vector3 vector)
{
        Quaternion qx = Quaternion.AngleAxis(vector.x, Vector3.right);
        Quaternion qy = Quaternion.AngleAxis(vector.y, Vector3.down);
        Quaternion qz = Quaternion.AngleAxis(vector.z, Vector3.back);

        return (qx * qy * qz);
}
```

Figura 14: Código para pasar de coordenadas en tres dimensiones a cuaterniones[17]

Finalmente cuando se acabe de cargar la mano completamente se realizará un giro sobre la base en x -90º, las rotaciones siempre se realizarán usando cuaterniones, tanto en la carga como a la hora de ejecutar los movimientos.

Los cuaterniones tienen muchas ventajas a la hora de ser usados, aunque sean menos intuitivos para los seres humanos, tienen una menor carga computacional a la hora de realizar operaciones con ellos, evita ambigüedades a la hora de realizar rotaciones con ellos y previenen el error conocido como el bloqueo del cardán que se produce cuando al rotar los

[17] Fuente propia

ejes el sistema pierde la referencia de al menos uno de ellos, haciendo que cuando se rote en determinado eje se modifiquen de manera paralela los datos de los dos ejes al mismo tiempo. Los cuaterniones al tener una cuarta componente previenen este error.

Es importante tener en cuenta las rotaciones del modelo que hay por debajo del sistema. Es decir cuando se rote se tiene que tener en cuenta la propiedad *axis* presente en el *joint*, ya que nos dará información sobre el eje en el que se aplica la rotación, para rotar la articulación se multiplica el valor por el que se está rotando por el axis para encontrar la posición dentro del simulador.

Todas las rotaciones y movimientos se realizan con respecto al padre. Cuando se va cargando la mano mediante el lector en el caso de la mano Shadow llama a hijos que aún no han sido creados en el momento de la asignación. Para evitar errores se ha decido asignar este tipo de relación una vez que se han leído todas las etiquetas. Hay algunas rotaciones que se hacen antes de la asignación de las relaciones de padres e hijos como son las rotaciones dentro del modelo que aparece en el *link.* Las posiciones de las rotaciones del URDF son posiciones relativas al padre, como las rotaciones se hacen respecto a este en Unity3D no ha hecho falta realizar ninguna transformación extra.

Para realizar el lector se ha utilizado un componente de .Net para la lectura de ficheros en formato XML. Esto permite poder captar de manera rápida los atributos de los nodos que representa. Este es el principal cambio que tiene el lector URDF con respecto al sistema implementado el año pasado en el que se hacia la lectura del archivo abriendo el fichero de texto y parseando manualmente los datos. El lector URDF se ha realizado desde 0.

Cuando se cargan las dos manos se ha decidido cargar primero siempre la mano derecha y después la izquierda.

Shadow y Allegro tienen varias diferencias a la hora de cargar sus URDF, Shadow declara al inicio los valores de las texturas que se van a usar mientras que Allegro lo hace dentro de los *links*. También con respecto a las colisiones Shadow tienen dos cargas de modelo diferentes en la siguiente imagen se muestras las *bounding boxes* que utiliza Shadow.

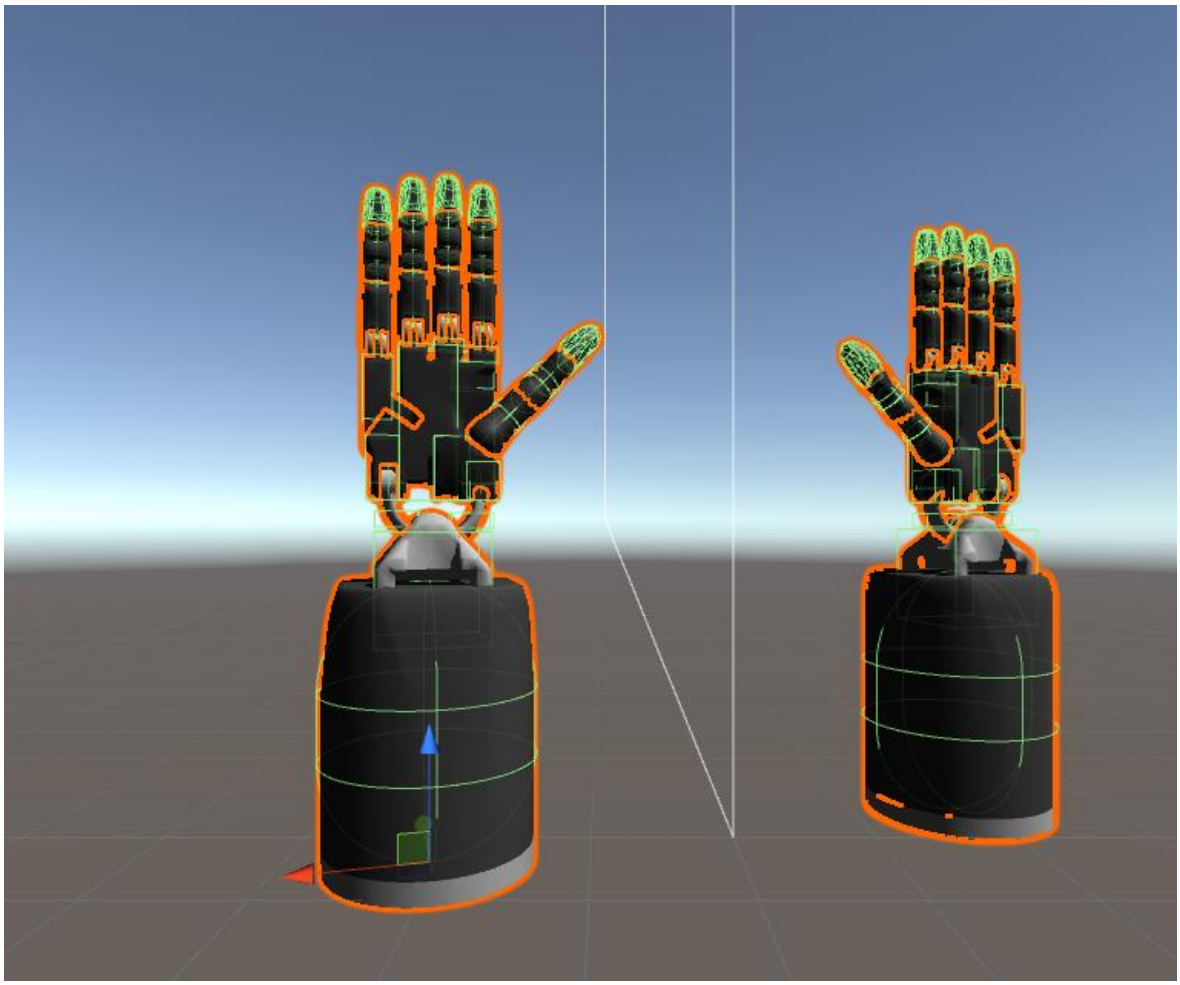


Figura 15: Bounding boxes de Shadow[18]

Shadow utiliza para cargar la *boundig box* tanto simplificando el sistema usando figuras geométricas sencillas como usando un *mesh*, lo que puede caer en una colisión consigo mismo, en la Figura 15 se aprecian las *boundig boxes* usadas en color verde, como se puede comprobar al realizar la carga hay colisiones entre ellas mismas, es por esto que se ha cambiado la estructura de las cargas de los colisionables permitiendo tener más de una figura de colisión dentro de un mismo objeto. También se ha hecho una estructura en la que cada uno de los *GameObjects* sabe quién es su padre y quien es su abuelo para saber con quienes puede o no puede colisionar. Hay que tener en cuenta que URDF utiliza como forma colisionable el cilindro, lo más parecido en Unity3D a un cilindro es una capsula, así pues, se ha decidido sustituir las figuras cilíndricas por las capsulas. Unity3D no utiliza cilindros el cálculo computacional de las colisiones en esta figura geométrica es muy grande. El

[18] Fuente propia

cálculo cuando se utiliza una capsula es mucho menor. Al no diferir mucho un cilindro de una cápsula es habitual su sustitución.

El resto de consideraciones a tener en cuenta se aplican de manera normal a la lectura de ficheros URDF, para saber cómo funcionan estos ficheros se puede encontrar más información en el 4.2. Entendiendo las etiquetas de un archivo URDF.

En la Figura 16 se muestra todas las cargas que son posibles de hacer en la aplicación.

a) Allegro Hand izquierda *b)Allegro Hand derecha* *c)Allegro Hand las dos*

d)Shadow Hand izquierda *e)Shadow Hand derecha* *f)Shadow Hand las dos*

Figura 16:Carga de las manos usando el lector URDF a) Allegro Hand mano izquierda b) Allegro Hand mano derecha c) Allegro Hand las dos manos d)Shadow Hand izquierda e) Shadow hand derecha f) Shadow hand las dos manos[19]

El sistema de carga de ficheros se ha realizado lo más general posible usando toda la información del fichero URDF, de manera que es fácilmente extrapolable a la carga de otros sistemas robóticos. Sin embargo, sería necesario pequeñas modificaciones para admitir cualquier URDF.

Por último, a la hora de realizar la carga los modelos éstos deben estar en formatos compatibles, el formato STDL que utiliza la mano Allegro en ROS no es compatible con Unity3D por lo que se ha pasado a formato obj. Es importante almacenar tanto los URDF como los modelos que se van a cargar en una carpeta llamada *assets* dentro de Unity3D. Esta carpeta no viene por defecto y debe ser creada manualmente.

[19] Fuente propia

Los archivos URDF y los modelos 3D usados son los mismos que se utilizan en los sistemas simulados oficiales. En el caso de Shadow el URDF era .xacro por lo que se ha pasado a URDF mediante el correspondiente comando en terminal.

2.5.1.2. Movimiento de articulaciones

Para realizar el movimiento de las articulaciones se puede usar el movimiento utilizando el menú de articulaciones (Figura 13b) o seleccionando una articulación sobre el modelo (Figura 13c). Para mover las articulaciones dentro de los sistemas reales se ha cambiado el sistema de envío.

Se mantiene el envío de datos cada 0.3 segundos al servidor de la posición actual del slider si se ha modificado el valor cuando se esté usando la mano Allegro. En cambio, cuando se utiliza la mano Shadow, para evitar tener al planificador trabajando todo el tiempo y que se ralentice el sistema, únicamente se envían datos de las posiciones actuales al servidor cuando se hayan modificado las posiciones de la mano y se levante el dedo de la pantalla. Cuando se realizaba el envío de datos de manera muy seguida el sistema podía tardar varios minutos en realizar una acción sencilla.

2.5.1.2.1. Uso del menú de articulaciones

A la par que se van cargando el modelo, si procede, cuando se trate de una articulación móvil, se genera un slider que permitirá poder mover determinada articulación entre un máximo y un mínimo, en un eje determinado. Toda la información de este tipo de rotación viene almacenada en la articulación (*joint*) que se ha leído en el archivo URDF.

Los diferentes sliders que se generan aparecerán dentro del menú que contiene el menú de articulaciones. Cada slider está relacionado con una articulación directamente.

En la Figura 17 se muestra una secuencia de movimiento producida mediante el menú de articulaciones.

Figura 17: Movimiento utilizando el menú de articulaciones[20]

[20] Fuente propia

2.5.1.2.2. Movimiento mediante la selección de una articulación

La selección de determinado *joint* pulsando sobre él es posible gracias a que los elementos *link* poseen una *bounding box*, esta *bounding box* viene especificada en el elemento *collision* del URDF. Para poder seleccionarla se utiliza *raycasting*. Este mecanismo que viene implementado en las físicas de Unity3D consiste en lanzar un rayo, en este caso desde el lugar donde hayamos tocado, para saber cuál es el primer objeto en línea recta con el que se va a encontrar el rayo.

Cuando se mueve el *slider* que aparece en la parte inferior también se mueve automáticamente el que está oculto en el menú de *joints*. De esta forma se evitan inconsistencias entre ambos controles.

Para realizar la selección del *joint* ha habido un cambio significativo con respecto al trabajo de final de grado. Para la selección antes se usaba el *GameObjectFind* que busca el objeto por el nombre, esto podía producir errores a la hora de realizar la búsqueda cuando hay dos manos Allegro cargadas al mismo tiempo, ya que tienen nombres de articulaciones que se repiten, además que esta función es lenta. Para solventar esto se ha guardado dentro de los *gameObject* que colisionan una referencia del *gameObject* que tiene la articulación. De esta manera, cuando se selecciona un *joint* se sabe a qué articulación se está haciendo referencia para moverla.

En la Figura 18 se muestra una secuencia de movimiento producida mediante un *joint* seleccionado.

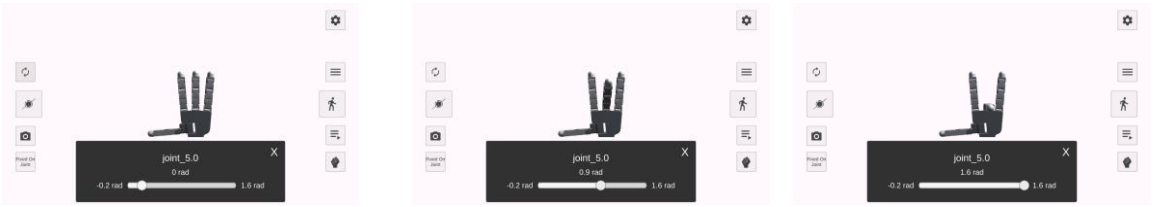


Figura 18: Movimiento mediante la selección de una articulación[21]

[21] Fuente propia

2.5.1.3. Secuencias y pasos

En esta aplicación se permite poder crear secuencias de movimientos. Esto consiste en ejecutar una interpolación entre diferentes posiciones de las articulaciones de las manos (pasos) para crear movimientos más complejos, en los siguientes apartados se detalla con mayor profundidad cómo se ha realizado.

La interpolación entre los pasos que se ha utilizado es una interpolación lineal (Figura 19), lo que asegura una velocidad constante entre posición y posición. Los datos que se utilizan durante toda la aplicación están en coordenadas articulares.

$$b = (q_{fin} - qin) / T$$

$$q(T) = q_{in} + bT$$

Figura 19: Fórmula interpolación lineal[22]

El dato de la velocidad b se recoge de los valores de las opciones. Esta velocidad es con la que se ejecuta dentro del simulador, en el caso de la mano Shadow la velocidad varía, es el controlador el que decide la velocidad a la que se mueve la mano.

2.5.1.3.1 Pasos

Cada vez que se pulsa al botón de crear un nuevo paso (Figura 13), se almacena de manera interna una lista con todas las posiciones actuales de las articulaciones. Los pasos indican las posiciones finales por las que tiene que pasar una secuencia en un momento dado, estos pasos se almacenan en orden.

La lista de pasos puede ser modificada antes y después de guardarse, pudiendo ser eliminado el último paso introducido o todos los pasos actuales del sistema mediante el menú de pasos (Figura 13e).

El funcionamiento de los pasos es prácticamente el mismo que en el trabajo de final de grado que realicé el año pasado. Cambia el sistema cuando se guardan los pasos para crear una nueva secuencia y la manera en la que se cargan. Cuando se guarda una secuencia de pasos se guarda también el identificador de la mano en el que se creó la secuencia.

[22] Apuntes asignatura de control y programación de robots

2.5.1.2.2 Secuencias

Las secuencias ya venían implementadas en el trabajo de final de grado pudiendo en el caso de la mano Allegro poder reproducir los movimientos creados en otra mano, esto era debido a que los *joints* cargados para la mano izquierda y derecha tenían el mismo nombre, de manera que usando esta información se pasaba los datos de uno a otro permitiendo ejecutar una secuencia creada en la mano izquierda en la derecha. Esto tenía varios puntos flacos, el principal es que la búsqueda por nombre de *GameObject*, que es lo que se usaba, es considerablemente lenta dentro del sistema de Unity3D, la segunda es que no funcionaba con la mano Shadow al no tener el mismo nombre de *joints* para la mano izquierda que para la derecha y la tercera es que se iba a permitir la ejecución de dos manos simultáneamente lo que produciría errores a la hora de realizar los movimientos en la mano Allegro ya que tiene los nombres repetidos.

En cada secuencia se almacena el nombre identificador de la secuencia, el identificador de la mano con la que se ha guardado la secuencia, una lista de los nombres de los *joints* y a continuación una lista con todos los pasos que conforman la secuencia.

Para cargar las secuencias y permitir su ejecución se hace lo siguiente, si el identificador de la mano es el mismo se cargan en el mismo orden que se ha guardado, si no coincide el número se lee la lista de nombres de los *joints* se busca equivalencias entre el nombre del *joint* y la lista guardada, para la búsqueda de equivalencias se utilizan las listas con los nombres del sistema. Se han creado dos grupos de manos del 0 al 2 corresponden a Allegro y del 3 al 5 a Shadow, de manera que no se pueden cargar las secuencias de Allegro en Shadow y viceversa.

El comportamiento que tendrá el sistema cuando se ejecute una secuencia dependerá de si es creada en una o en las dos manos. Será el mismo tanto para Allegro como para Shadow.

- Secuencia creada en una mano:
    - Cuando se ejecute en la misma mano que se creó la secuencia no variará.
    - Cuando se ejecute en la mano contraria donde se creó, los datos enviados a los dedos serán simétricos, es decir el movimiento referente a determinado dedo será el mismo.
    - Cuando se ejecute en dos manos al mismo tiempo el movimiento será simétrico. Lo que se ejecute en la mano derecha se ejecutará en la izquierda.

- Secuencia creada en dos manos
    - En la mano izquierda se reproducirá únicamente la secuencia de movimientos relacionada con la mano izquierda y en la derecha los movimientos correspondientes a esta mano.
    - Cuando se ejecute en las dos manos su ejecución será normal.

Con la finalidad de poder seguir alargando determinada secuencia, cuando se ejecute una de ellas, la lista de pasos se actualizará añadiendo todos los pasos que conforman la secuencia recién reproducida.

Las secuencias pueden ser reproducidas o eliminadas desde el menú de secuencias (Figura 13f).

En cuanto al envío de secuencias al servidor tenemos dos casos, en el caso de la mano Allegro se envían datos de las posiciones de las secuencias cada 0.05 segundos, en cambio las posiciones a las que tiene que llegar la mano Shadow sólo se envían una vez con la posición final de cada uno de los pasos por los que se quiere pasar. De esta manera, se evita tener que llamar demasiadas veces al planificador y el tiempo de ejecución disminuye, para la ejecución de una secuencia sencilla usando el mismo método que en el caso de la Allegro, la mano Shadow podía tardar más de 15 minutos, en el caso de la simulación.

2.5.1.4. Abrir y cerrar la mano, control con señales EMG

Los botones de abrir y cerrar la mano son muy similares a la ejecución de las secuencias.

Difieren en que se han introducido en un módulo que se llama cuando el servidor indica a la mano que se ha producido una orden mediante señales EMG. Cuando se recibe el valor cerrar la mano se cerrará y cuando se envíe abrir la mano se abrirá. También se pueden enviar órdenes de apertura y de cierre a las manos pulsando sobre éste botón. Los datos de las posiciones no se cargan de ficheros, están almacenados directamente en variables de la aplicación.

Los envíos de datos de las manos al servidor funcionan exactamente como en el caso de usar secuencias.

En la Figura 20 se muestra las secuencias de apertura y cierre de la mano para Shadow y para allegro enviando la orden desde el cliente.

a)2 Shadow cerrar *b)2 Shadow abrir* *c)1 Shadow cerrar* *d) 1 Shadow abrir*

e)2 Allegro cerrar *f)2 Allegro abrir* *g)1 Allegro cerrar* *h)1 Allegro abrir*

Figura 20: Movimientos de apertura y cierre generados al pulsar el botón de la interfaz a) Movimiento con dos manos Shadows de cierre b) Movimiento de dos manos Shadows de apertura c) Movimiento de una mano Shadow de cierre d) Movimiento de una mano Shadow de apertura e) Movimiento de dos manos Allegro de cierre f) Movimiento de dos manos Allegro de apertura g) Movimiento de una mano Allegro de cierre h) Movimiento de una mano Allegro de apertura[23]

El motivo por el que se ha decidido dejar una estructura similar entre las secuencias normales y las ejecutadas por el botón de cierre/apertura es porque así en una posible ampliación se podría utilizar cualquier secuencia definida por el usuario como secuencia de movimiento realizada por señales EMG.

2.5.1.5. Mejoras de visualización

En comparación con el trabajo realizado el año pasado ha sido necesario introducir determinadas modificaciones para mejorar la calidad de las sombras en los modelos cargados.

Teniendo en cuenta que el objeto cargado es a una escala muy pequeña con los valores por defecto no era posible poder realizar los cálculos de la sombra correctamente. Se han cambiado los valores mínimos para los cuales un objeto puede hacer sombras dentro de las opciones de Unity3D, produciendo una mejora de la calidad de visualización.

[23] Fuente propia

2.5.1.6. Mejoras en los métodos de colisión

Se ha modificado la colisión dentro del sistema. Actualmente el método de colisión sabe a qué altura del árbol se está, previniendo colisiones con aquellas estructuras con colisionables que están una posición con respecto a sí mismo o por debajo o por encima de él en una distancia de un *link* en la estructura de árbol, también se evitan colisiones con todos los colisionables que están a su misma altura dentro del árbol.

El acceso directo a los objetos están guardados en los *joints* mediante una variable pública, que permite su acceso directo, de ésta manera se evita tener que llamar el *GameObject find* o tener que estar buscando entre los padres del sistema mediante las ordenes *getParent* o *getChild* [número de hijo].

Además de ésta mejora también se prohíbe el envío de datos al servidor si se ha detectado una colisión dentro del sistema para prevenir errores.

2.5.1.7. Rotación y zoom de la cámara sobre una articulación y sobre el sistema general

En el trabajo de final de grado la rotación de la cámara se realizaba usando la posición del primer *link* del sistema, que quedaba bastante centrada para el caso de la mano Allegro. Sin embargo, al hacer lo mismo en la mano Shadow la cámara no enfocaba adecuadamente la escena.

Lo que se ha hecho para solucionar esto, teniendo en cuenta también la carga de dos manos simultáneamente y la carga de la mano Shadow, es crear un *GameObject* vacío y situarlo en el centro exacto mirando las coordenadas más pequeñas de las manos y las más grandes. El sistema rota entorno a este *gameObject* de referencia para asegurar que ambas manos caben en la pantalla. Para hacer que la cámara se fije en determinado *joint* se cambia el *GameObject* entorno al que se rota por el *joint* seleccionado. Al hacer esto ha sido necesario guardarse en el caso de guardar una *view* la información del *joint* sobre el que se estaba usando como pivote de rotación.

a)Pivote en el centro *b)Pivote en el centro con rotación* *c)Pivote en el centro con zoom*

d)Pivote en un joint *e)Pivote en un joint rotación* *f)Pivote en un joint con zoom*

Figura 21: Rotación y zoom en la mano Shadow rotando y realizando zoom sobre el centro de las posiciones y sobre uno de los joints a)Pivote en el centro b)Pivote en el centro con rotación c)Pivote en el centro con zoom d)Pivote en una articulación e)Pivote en una articulación con rotación f) Pivote en una articulación con zoom.[24]

Para seleccionar el *joint* sobre el que se quiere rotar se utiliza el menú de la Figura 13g.

Se muestra en la Figura 21 un ejemplo de rotación y de zoom sobre la mano Shadow centrados en el pivote del centro y después cambiando el punto del pivote a ser otro de los *joints*

2.5.1.8. Vistas

El sistema permite poder cargar guardar y eliminar vistas a partir del menú de vistas (Figura 13d). Cuando se guarda una vista se almacena en el dispositivo la información de la posición, el zoom y el pivote de la cámara. En el anterior proyecto no se tenía en cuenta el pivote.

Cuando se selecciona una vista se cambia la configuración de la cámara a los datos almacenados dentro de las vistas.

[24] Fuente propia

2.5.1.9. Menú de opciones

Se ha realizado un menú de opciones que dota al sistema de determinada personalización por parte del usuario. (Figura 13i). Cada vez que se modifican los datos dentro del menú de opciones se guardan en un archivo de preferencias que es cargado al inicio del programa permitiendo al usuario mantener sus opciones preferidas sin tener que cambiarlas entre ejecuciones.

De este menú se han conservado todas las funcionalidades desarrolladas en el trabajo de final de grado y se ha incluido una nueva que consiste en utilizar sobre las manos cargadas un *shader* de mano realista.

Las opciones disponibles en este menú son las siguientes:

- Selección de la unidad de medida, en grados o en radianes, esta información es meramente visual, al servidor los datos siempre le son enviados en radianes y se mantienen con esa información internamente.
- Cantidad de número de decimales con un máximo de 6 y un mínimo de 1, este dato también es visual.
- Velocidad de las secuencias, indica la velocidad a la que se desea mover las secuencias. De 0.3 a 1
- Velocidad a la que se mueve la cámara, este factor se multiplica por la rotación normal de la cámara. Este dato varía entre 1 y 15.
- Invertir ejes de la cámara, cuando están activos se multiplican las rotaciones de la cámara por -1.
- Activar la colisión, permite activar o desactivar la colisión entre los dedos.
- Activar la apariencia de piel humana. Esto consiste en cargar un *shader* optimizado para dispositivos telefónicos sobre las manos seleccionadas. El *shader* al estar optimizado y preparado para su uso en *smathphones* tiene una menor carga computacional y se asegura su correcto funcionamiento en estos sistemas. En la Figura 22 se muestra la carga de la mano robótica *shadow* y allegro con textura de piel.

a)Mano Allegro con textura de mano humana

b)Mano Allegro con textura de mano real con zoom

c)Shadow hand con textura de mano real humana

d)Shadow hand con textura de mano real y zoom

Figura 22: Carga de textura de mano real sobre las manos robóticas simuladas y zoom para apreciar las arrugas de la piel. a) Mano Allegro Hand b) Mano Allegro con zoom sobre uno de los dedos c) Shadow hand con textura de piel humana d) Shadow hand con textura de piel humana real y zoom sobre uno de los dedos[25]

- Cambiar de idioma, permite seleccionar entre inglés y castellano, para esta sección se implementó un módulo de idiomas que lee en un txt las traducciones posibles del sistema, permitiendo de manera sencilla ampliar la cantidad de idiomas del sistema.
- Cambiar de mano, te permite cambiar de mano de manera automática sin tener que cerrar la aplicación.
- Exit, cierra la aplicación
- About, da información del sistema.

[25] Fuente propia

2.5.1.10. Feedback visual

Otra implementación realizada es la creación de pop-ups que aportan información sobre una acción recién realizada. Estos pop-ups aparecen durante un segundo y desaparecen indicando que se ha realizado determinada acción. Este sistema de pop-ups no estaba en la aplicación realizada el año pasado y aporta información relevante para el correcto uso de la aplicación. En la Figura 23 se muestra un ejemplo de pop-up.

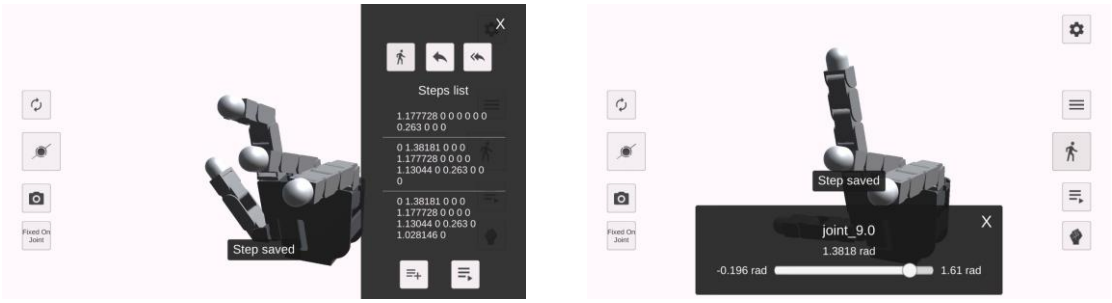


Figura 23: Pop-up mostrando paso guardado[26]

Los pop-ups por el momento avisan cuando guardas un paso o cuando pulsas el botón de reiniciar posiciones.

2.5.1.11. Interfaz

Para evitar que se solapen las interfaces cuando se tienen abiertas las lista de *steps*, la lista de secuencias o la lista de *joints*, si se selecciona alguno de los menús que aparecen en la parte de abajo estos se esconden y cuando se cierra el menú abierto se vuelven a abrir aquellos menús que estaban abiertos inicialmente.

Tanto la lista de *steps* como la lista de secuencias se pueden manejar al mismo tiempo.

Los controles de cámara y de resetear posiciones seguirán pudiéndose manejar con el menú de la lista de *joints* abiertos, puesto que aparecen dentro de esta interfaz, de esta manera no es necesario tener que ir abriendo y cerrando interfaces todo el rato.

Cuando se seleccionan las opciones o la introducción de nombres se ocultan los menús que estuvieran abiertos y se muestran en pantalla completa. De fondo, debido a las transparencias se puede ver la mano en la posición en la que la hayamos dejado, solo se esconden los menús.

Se ha seleccionado colores blancos y negros y fondos con transparencias para los diferentes menús. Los colores blancos y negros evitan problemas para aquellas personas que sean

[26] Fuente propia

daltónicas. Las transparencias permiten poder ver a través de los menús, sin embargo, no son muy transparentes para evitar que se pueda solapar el fondo con las letras.

La mayoría de iconos usados son de Google icons, algunos como por ejemplo el del menú de *joints* o el de apertura o cierre son de producción propia.

2.5.1.12. Las implementaciones que se conservan iguales al Trabajo realizado el año pasado

En este apartado se mencionan las diferentes opciones que se han mantenido iguales al trabajo de final de grado desarrollado al año pasado y el motivo por el cual se ha decidido mantener estas implementaciones.

Se ha mantenido igual la apariencia de las ventanas de guardado, así como la distribución general de los menús del sistema y los colores.

Se han añadido nuevos botones a la distribución pero manteniendo la idea principal del sistema, los botones más grandes de la interfaz hacen referencia a las opciones más usadas por parte del usuario, así como su distribución por la pantalla puesto que no entorpece el uso del robot.

Los colores se han mantenido ya que aseguran un correcto entendimiento y sin pérdidas de información a los usuarios que estén usando la aplicación y tengan problemas de confusión de colores o vean la escena en escala de grises. Además, aunque se utilice un color blanco, no es un color blanco puro, es un color que tiene tonos más rojizos lo que favorece y evita el cansancio visual por parte del usuario.

Las funcionalidades del menú de opciones son exactamente las mismas. Se ha añadido la carga de la textura de mano real.

El resto de funcionalidades se han modificado para que se puedan usar con una o varias manos simultáneamente, además del uso de la mano Shadow.

2.5.2. Implementación del Servidor con los nodos de ROS

La estructura dentro de los nodos de ROS se ha modificado para permitir la conexión de dos manos simultáneas, además de las tramas de datos y el parseador del sistema para saber la información que se está recibiendo. En este apartado se va a ver cómo se ha implementado la parte del servidor.

Lo primero que se hizo fue modificar las órdenes de las tramas de datos que nos llegan con una estructura que permitiese saber el dato que nos estaba llegando, la mano a la que se hacía referencia o el tipo de instrucción que se está dando. Teniendo siempre un indicador de inicio de comienzo de información para describirlo y de final.

El parseador se modificó para que fuera más rápido y capaz de interpretar las diferentes tramas de datos que se reciben.

El servidor envía las posiciones actuales en las que se encuentra el sistema a cada vez que se piden datos, una vez cada 0.05 segundos. Cuando se pide información se hace un *SpinOnce*() de ROS. Éste método nos permite recoger los valores de los *CallBakcs* de los *topics* a los que se está suscrito.

Se va a separar este apartado en dos una para la mano Allegro y otro para la Shadow.

2.5.2.1. Conexión de la mano Allegro

La mano Allegro permite realizar la conexión de dos manos de manera al mismo tiempo en un mismo ordenador (Véase 4.5. Ejecución de la mano Allegro en Linux). Para la conexión de las manos se ha definido la mano con NUM:=0 a la mano derecha y con NUM:=1 a la mano izquierda, de ésta manera se pueden ejecutar las manos derecha e izquierda.

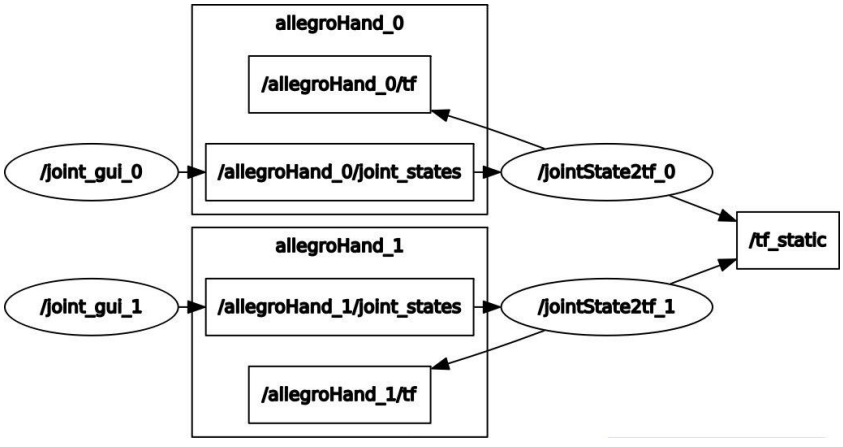


Figura 24: Conexión de dos manos Allegro simuladas sin el servidor en marcha[27]

[27] Fuente propia

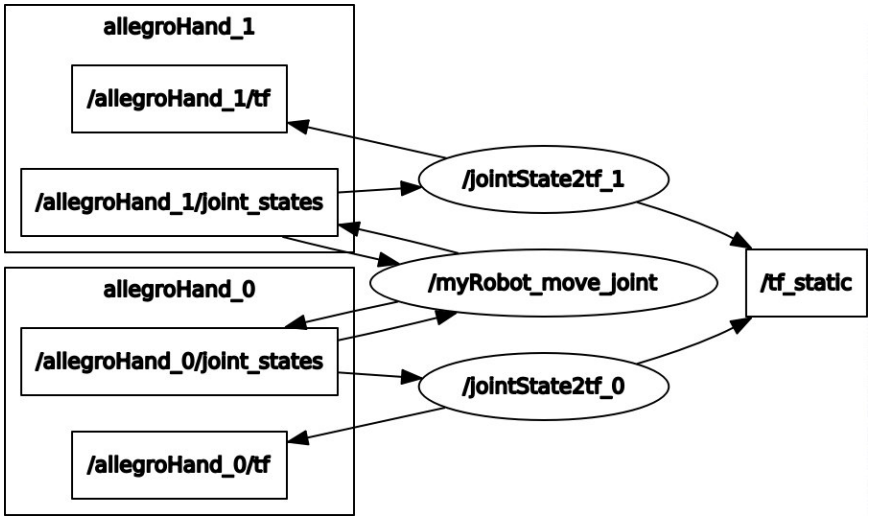


Figura 25: Conexión de dos manos Allegro simuladas con el servidor en marcha.[28]

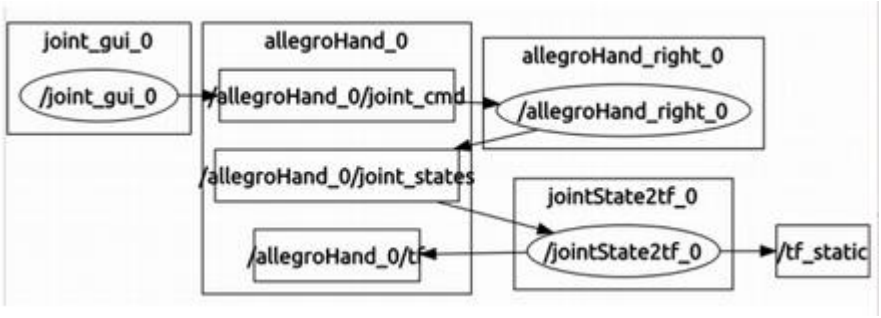


Figura 26: Conexión de una mano Allegro Real sin el servidor[29]

Para enviar las órdenes de control se ha sustituye el *nodo /joint_gui_1, /joint_gui_0* o ambos. Para ello primero se han matado esos nodos usando la orden *rosnode kill* (véase 4.1.1. Rosnode). Es importante eliminar esos nodos para evitar interferencias. Después se ha suscrito nuestro nodo (*/myRobot_move_joint*) al *topic* en el que estaba suscrito los */joint_gui* en el caso de ser la mano simulada a */allegroHand_identificador/jointStates* para la real */allegroHand_identificador/joint_cmd* (Figura 24, Figura 25, Figura 26). Esta es la principal diferencia entre las manos simuladas y reales. Para poder recibir la información de las posiciones actuales en las que se encuentra la mano se ha suscrito al *topic* */allegroHand_identificador/jointStates* tanto en la simulada o en la real.

[28] Fuente propia
[29] Fuente propia

Las figuras 24, 25 y 26 han sido generadas mediante *rqt_graph* de ROS, dónde se representa la interconexión entre los nodos y los *topics* del sistema. En los rectángulos vienen los nombres de los *topics* y en las elipses los nombres de los nodos del sistema.

2.5.3.2. Conexión de la mano Shadow

La mano Shadow no permite la conexión de dos manos de manera simultánea de por sí, cuando se lanza en ROS una se sustituye la otra. Para poder utilizar dos manos en caso de Shadow al mismo tiempo se utiliza una conexión utilizando sockets TCP/IP a un ordenador en el que se tiene un cliente con los nodos de la otra mano de Shadow. Este cliente programado en C++ recibe las posiciones y las envía al sistema con la mano conectada, cada cierto tiempo cuando el servidor lo requiere envía las órdenes actuales en las que se encuentra esta mano.

Para poder enviar las órdenes a la mano Shadow tanto simulada como real se ha utilizado lo que recomienda el laboratorio Shadow, que es el uso de un método implementado por ellos en python, que permite la conexión con sus sistemas utilizando el planificador. Es necesario, en el caso de la mano Shadow, utilizar el planificador de movimientos de la mano para evitar errores. Esto ralentiza el sistema.

Tanto el cliente que tenga la Shadow como el servidor siguen la misma estructura para realizar la conexión. Primero se reciben las órdenes del cliente con las posiciones, el servidor sabe qué mano se está ejecutando y si hay una segunda mano ejecutándose en otro cliente, ya que éste cuando se conecta le envía un aviso al servidor. Una vez recogidos los datos se publican dentro del sistema en un *topic* dependiendo de si se es la mano izquierda o la derecha.

Cuando el servidor recibe órdenes de control referentes a la mano Shadow publica en un *topic*, dependiendo de si se trata de la mano izquierda o derecha, de tipo *jointStates* la posición y los nombres de las articulaciones de la mano robótica. A ese *topic* está suscrito un nodo programado en python, ese nodo se encarga de usar la función de python realizada por Shadow para mover la mano haciendo inicialmente una planificación del sistema.

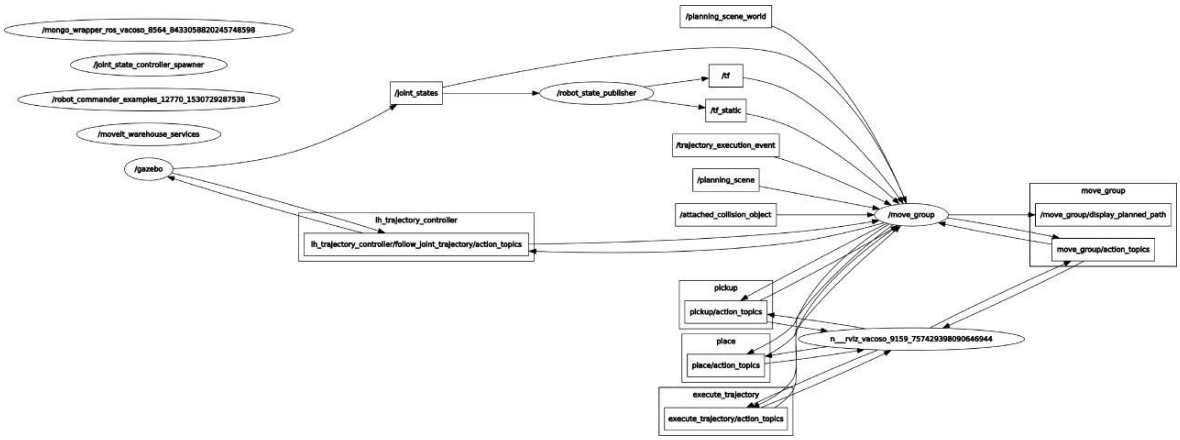


Figura 27: Grafo rqt_graph de la ejecución de la mano Shadow sin el servidor[30]

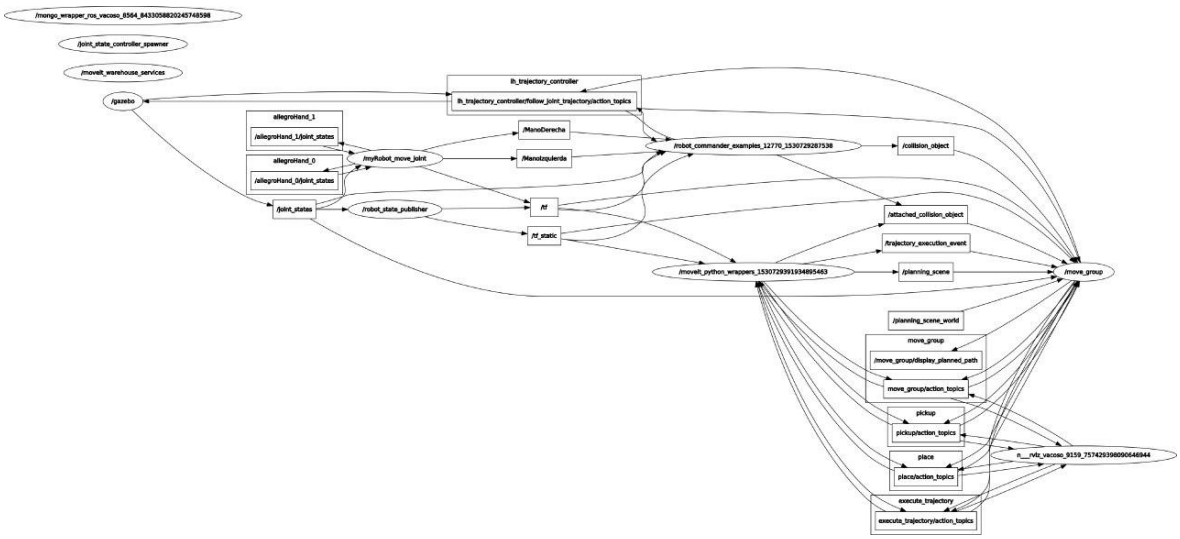


Figura 28: Grafo rqt_graph de la mano Shadow con servidor[31]

En la Figura 27 y Figura 28 se muestra la interconexión de los nodos antes de ejecutarse el sistema y con el servidor conectado. Como se puede comprobar, cuando se ejecuta el servidor el nodo del servidor *myRobot_move_joint* publica en dos *topics*, publicará en el de la mano derecha o en la izquierda según se use una u otra. El nodo de python recibe la información de estos nodos mediante la suscripción y realiza los movimientos en la mano

[30] Fuente propia
[31] Fuente propia

Shadow correspondiente. En cuanto al sistema real y el simulado no es necesario realizar ningún cambio en la estructura puesto que se suscribe al *topic* correspondiente directamente.

2.5.3.3. Módulo de señales EMG

Para este proyecto se ha facilitado un módulo que recibe e interpreta señales EMG utilizando Matlab en ROS y publica la información del movimiento en un *topic* de tipo cuaternion, donde se indican movimientos de brazo y apertura y cierre de la mano. Se publica en la variable W del *topic* un valor para representar la apertura y el cierre. Cuando el valor es 0 el servidor envía al cliente la orden para que se abra la mano, si ese valor es 1 la orden enviada es la de apertura.

El servidor está suscrito al *topic* que tiene la información que se publica desde el nodo de ROS en Matlab. Cuando se hace *SpinOnce* al haber recibido datos del cliente o una petición de actualización de la información se comprueba además de las posiciones actuales en las que se encuentra el sistema los valores publicados en este *topic*. Cuando se han modificado se envía la orden a los sistemas conectados, para ejecutar el movimiento correspondiente.

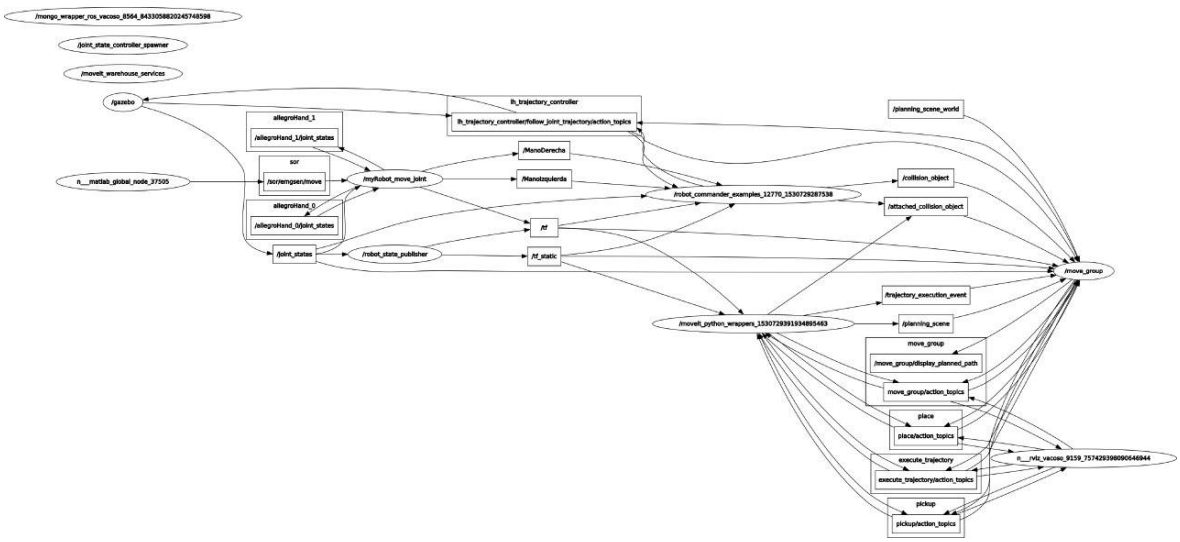


Figura 29: Grafo rqt:graph de la mano Shadow con el servidor y el módulo de señales EMG en una mano simulada[32]

[32] Fuente propia

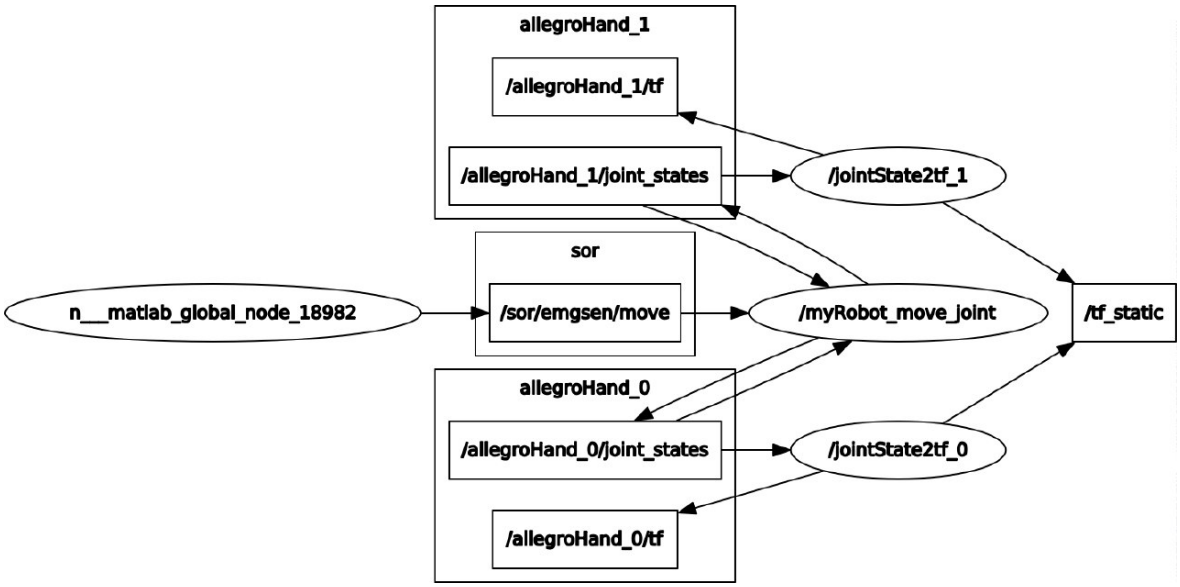


Figura 30: Grafo rqt_graph de la mano Allegro con el servidor y las señales EMG en una mano simulada[33]

En las anteriores figuras (Figura 29 y Figura 30) se muestra en nodo de Matlab publicando en el *topic* la información interpretada de las señales EMG. El nodo *myRobot_move_joint* se suscribe a este nodo y es el que se encarga de transmitir las órdenes a los clientes.

Cuando el cliente recibe una orden de apertura o de cierre se realiza la secuencia definida en el botón. El sistema que se encarga de reproducir esta orden es el cliente que hace de *updater*.

2.5.3. Interconexión entre los clientes y el servidor

Para realizar la interconexión entre el cliente y el servidor se han utilizado sockets TCP/IP, ya que aseguran la llegada de los paquetes y en el caso de la mano Shadow los datos de posición sólo se envían una única vez. Lo único que se conserva de la interconexión utilizada en el trabajo de final de grado es que son sockets TCP/IP. Las tramas se han modificado y se han solucionado errores relacionados con ellas. Además se ha añadido la posibilidad de poder realizar un control colaborativo sin enviar órdenes contradictorias al sistema, al modificar únicamente el dedo sobre el que se está ejecutando el sistema, en el caso de que se modifique la misma articulación de manera simultánea se hará caso al último cliente que envío la orden.

[33] Fuente propia

El paso de mensajes entre la interfaz de Unity3D y el servidor se ha realizado de la siguiente manera. El primer cliente que se conecta va a ser el que haga de *updater*. El *updater* se encargará de pedir refresco de información del sistema para él y para el resto de clientes cada 0.5 segundos, el refresco de la información realizará un *SpinOnce* dentro del sistema de ROS para recoger los datos actuales del sistema. Esto se ha elegido así ya que los nodos de ROS cuando hacen el *spin* son bloqueantes lo que puede producir errores a la hora de usar temporizadores dentro del servidor.

Una vez enviado al *updater* su información se ha implementado una trama de datos que se envía a los dispositivos telefónicos. La trama de datos da la información de las posiciones en las que se encuentran actualmente el robot mediante la subscripción al *topic* /*jointStates* de cada uno de los sistemas robóticos conectados:

```
inic Numero_de_mano Lista_de_posiciones_separadas_por_espacio end
```

Esa trama de datos se enviará por cada mano conectada al servidor a cada cliente conectado cuando se pida un *update* de información. El número de la mano será 0 para Allegro izquierda, 1 para Allegro derecha, 3 Shadow izquierda y 4 Shadow derecha.

El servidor también enviará en la trama de datos las órdenes “abrir” o “cerrar” cuando se registre en el *topic* de las señales EMG el valor que se corresponda (0 abrir, 1 cerrar). El servidor no envía tramas de datos que simbolicen ambas manos simultáneas.

Las tramas que pueden enviar los clientes son dos principalmente, o toda la posición de la mano lo que se muestra en el código que aparece a continuación o indicando el *joint* que se ha movido al servidor con el valor actual, esta última sólo en el caso de ser la mano Allegro. La trama de datos completa viene dada por:

```
inic Numero_de_mano Lista_de_posiciones_separadas_por_espacio end
```

Siendo el número de la mano 0 para Allegro derecha, 1 para Allegro izquierda, 2 para Allegro las dos manos al mismo tiempo, 3 para Shadow derecha, 4 para Shadow izquierda y 5 para dos Shadows simultáneamente.

La mano Allegro permite un ambiente colaborativo, ya que permite el envío de datos de manera instantánea. Cuando se modifica un único *joint* se envía una orden diferente al Servidor para indicar que *joint* se ha modificado, de esta manera se van sustituyendo de manera paralela las órdenes enviadas por un cliente, permitiendo controlar sin interferencias a varios clientes simultáneamente, esto no estaba implementado en el proyecto del año pasado.

Debido a que el sistema Shadow es más lento y no hay manera segura de saber si las órdenes enviadas son simultaneas o deben pasar por pasos intermedios se ha decidido enviar las órdenes de manera individual sin sumarlas al sistema, de esta manera se ejecutará una después de otra. Para evitar problemas dentro del envío de los datos por parte del cliente se refrescarán las posiciones a las posiciones actuales de los *jointStates* en el último cliente que envío datos cuando pase 0.5 segundos desde su última modificación.

Así pues, en Allegro tenemos un control colaborativo mientras que en Shadow se mantienen los datos de manera ordenada y se tiene la visualización del comportamiento de la mano bajo supervisión constante por parte de los clientes con un ligero retardo.

Es importante saber que en el caso de tener dos manos conectadas se envían por parte del cliente las posiciones de las manos derecha e izquierda seguidas en el misma lista de información, indicando que se trata de las dos manos Allegro o de las dos manos Shadow. En el caso de tener un control con las dos manos el servidor segmenta las posiciones sabiendo que en el caso de la mano Shadow se tendrán 24 articulaciones por cada mano un total de 48 en el mensaje recibido y 16 para el caso de la mano Allegro, 32 en total. El servidor separa en el caso de la Shadow los primeros 24 valores y en el caso de la Allegro los primeros 16 y los asigna como las articulaciones de la mano derecha y el resto serán de la mano izquierda. Cuando se cargan los sistemas con dos manos mediante el URDF se introducen en la lista de articulaciones primero las manos derechas y luego las izquierdas.

Cuando se segmentan los datos para la mano Shadow teniendo dos manos, se publican en el *topic* de la mano que se tiene cargada en el ordenador del servidor y la otra es enviada a un cliente especial en el que se tiene ejecutada la otra mano. Para realizar el control se utiliza el mismo nodo de python y la misma estructura que el servidor.

2.5.4. Ideas de programación descartadas

En un principio se planteó la idea de cambiar el servidor e implementarlo completamente en python, ya que la mano Shadow requería el uso de este lenguaje para comunicarse con ella. Se descartó esta idea ya que python es un lenguaje de programación considerablemente más lento que C++. De hecho, cuando se ejecutan conexiones utilizando python en realidad se está usando este lenguaje como interfaz y las operaciones complicadas las ejecutan en C++.

Python es un lenguaje de programación que te permite hacer pruebas muy rápidas, sin embargo, es poco recomendable su uso en aplicaciones que requieran tiempo real. Por ese motivo se decidió conservar el servidor implementado en C++ y dejar el nodo de ROS con acceso a python aparte.

2.5.5. Compilación para sistemas IOS

Cuando compilamos Unity3D para obtener la aplicación para Android lo único que es necesario es preparar el *player*, que es quien posee toda la información del paquete: nombre de la aplicación, orientación de la aplicación, nombre del desarrollador, etc. Una vez completado esto se puede generar una *.apk* que se guarda en el dispositivo y podemos ejecutarla para instalarla, después de aceptar poder instalar software de terceros, así como aceptar una alerta en algunos dispositivos, estás opciones te aparecen inmediatamente sin necesidad de buscarla dentro del menú de opciones.

Sin embargo, cuando compilamos para iOS, tras haber preparado el *player*, nos genera un proyecto para Xcode. Para poder ejecutar la aplicación en un dispositivo iOS es necesario tener un ordenador con macOS y Xcode instalado en él. Xcode recompilará el proyecto y generará la aplicación dentro del dispositivo iOS. No obstante, no podrás instalarlo ni tampoco ejecutarlo si no buscas la opción para permitirle a esa aplicación que se instale dentro del menú de opciones. Será Xcode quién te muestre la ruta dónde está esa opción.

La solución que prepara unity3D está pensada para que tras ser compilada en Xcode y que se ejecute en un dispositivo con iOS, por lo que si no dispones de uno no podrás probar la aplicación en una simulación sin obtener errores de arquitectura.

Es posible poder acceder a la aplicación instalada durante un tiempo limitado en lo que se supone que se testea la aplicación, tras esto la aplicación dejará de poder ser accesible.

2.6. Experimentos y discusión

2.6.1. Pruebas sobre usuarios

Se muestra a continuación las pruebas realizadas sobre usuarios, se ha testeado la aplicación probando los controles y la aplicación sobre un total de 11 usuarios, de ellos se tenía a 2 usuarios zurdos y el resto diestros, estos usuarios rondan los rangos de edad entre 20 y 43 años. Se les ha realizado una serie de preguntas sobre la comodidad de la interfaz, si les parecen intuitivos los iconos seleccionados y su opinión hacia como están ubicados los controles. A estos usuarios no se les ha dado una explicación básica de lo que iba la interfaz y han sido ellos los que han ido probando las diferentes opciones, de esta manera se puede ver si les parece también intuitivo el sistema y que opciones son más fáciles de descubrir dentro del sistema.

A continuación se muestran unos gráficos en la Figura 31 con la media de la opinión de los usuarios que han participado dando su opinión.

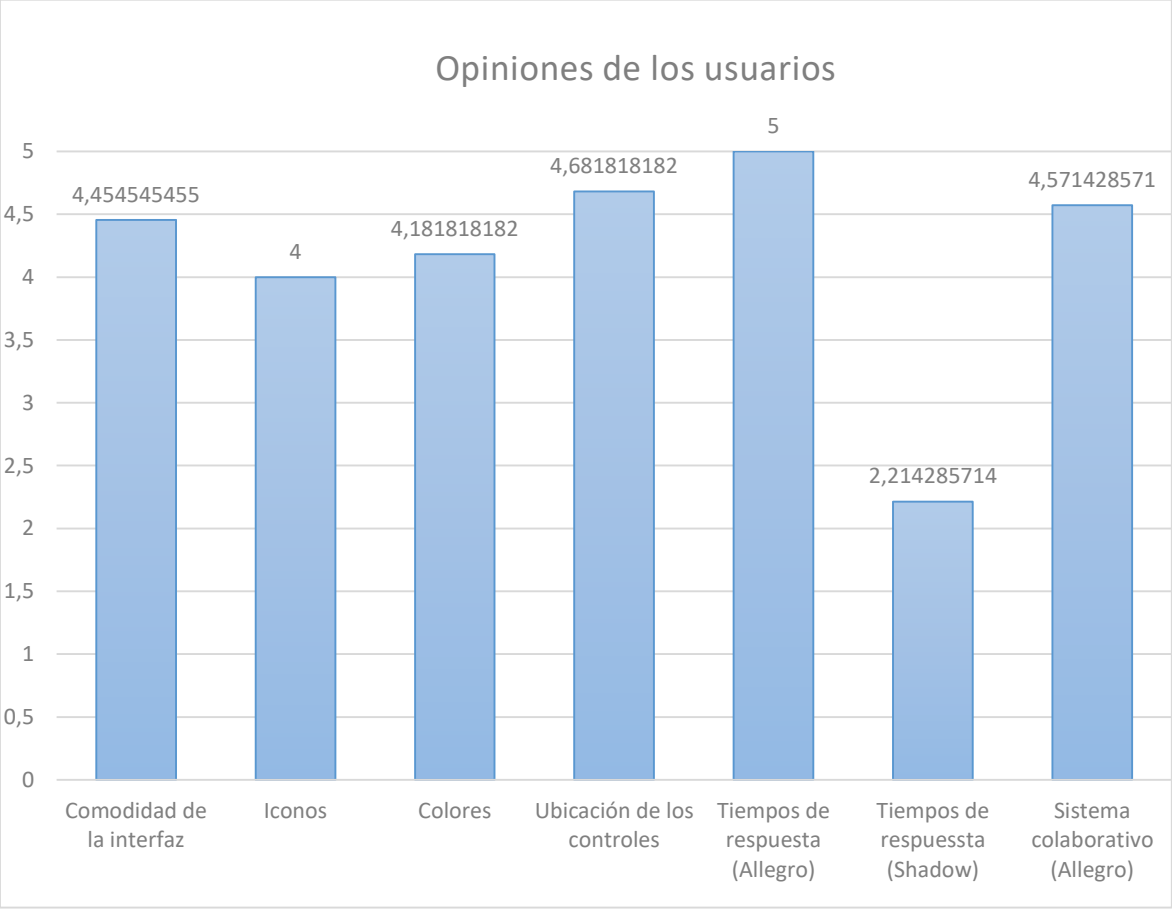


Figura 31: Opiniones de los ususarios[34]

[34] Fuente propia

En la Tabla 4 se muestra por usuario las propuestas de mejora del sistema.

Tabla 4: Propuestas de mejora por parte de los ususarios[35]

Usuario	¿Qué cambiarias?
1	Superposición de los menús laterales
2	Los iconos, algunos son poco intuitivos.
3	*Feedback* al pulsar botones, preferiría que los botones salieran desde la parte inferior en la interfaz
4	Pondría más margen en la barra del menú lateral
5	Iconos personales para la app. Asignaciones de colores para cada parte de la mano para que se vea más intuitivo
6	Pondría colores verdes y rojos para que sea más intuitivo y bonita, los entornos grises se me hacen monótonos, no hay que sacrificar diseño por funcionalidad.
7	Poner un botón de confirmación para eliminar las listas de pasos, ya que es eliminar un conjunto.
8	Poner más a la vista los controles de las articulaciones. *Feedback* luminosos sobre la articulación que estas
9	Nada
10	En los iconos agregaría algo de texto o algún símbolo más explicativo. La posibilidad de que el usuario cambie el color al gusto.
11	Añadir *feedback* visual para resaltar la articulación. Buscador de articulación.

Se ha podido instalar la aplicación en todos los usuarios a los que se les dado la apk para instalar.

Habría sido recomendable poder probar la aplicación sobre una mayor muestra de usuarios y abarcar rangos de edad mayores. La población se seleccionó pensando en que cualquier usuario podría ser capaz de utilizarla aunque no estuviera envuelto en el mundo de la robótica, de esta manera se podría ver cómo se desenvuelve todo tipo de usuario.

Los usuarios han dado *feedback* generalmente positivo resaltando que se ve intuitiva, clara y es posible poder llegar a las opciones utilizando pocos menús. Las funcionalidades que más han resaltado es la de poder escoger determinada articulación para moverla pulsando sobre ella, poder hacer zoom y rotar la cámara sobre determinada articulación. Varios

[35] Fuente propia

usuarios han indicado que preferirían que la interfaz tuviera una mayor cantidad de colores puesto que se está sacrificando diseño por funcionalidad. Lo que se ha indicado que necesita una mayor mejora son los iconos y que lo recomendable sería hacer iconos personalizados en lugar de utilizar iconos tan genéricos.

Lo que peor nota ha tenido con diferencia es el tiempo de respuesta con la mano Shadow. En este caso se ha mostrado el control con la mano simulada, en el caso de utilizar la mano real el tiempo de respuesta se reduce mucho.

2.6.2. Pruebas diferentes dispositivos

Tabla 5: Dispositivos en los que se ha probado la aplicación[36]

Dispositivo	**Sistema Operativo**	**Gama**
LG G6	Android 8	Alta
Huawei P9 Lite	Android 6 y Android 7	Media
Sony Xperia XA	Android 6 y 7	Media
Xioami Redmi note 4	Android 7.0	Media
Samsung Galaxy a5 2017	Android 6.01	Media
LG K10 4G	Android 6	Media
Moto G3	Android 6	Baja
Huawei P7 Lite	Android 5	Baja
Samsung S4	Android 5.01	Baja
Sony Xperia E	Android 4.1	Muy Baja
Sony Xperia Tipo	Android 4.0.4	Muy Baja (descatalogado)
Vodafone 890N	Android 4.4.4	Muy Baja (descatalogado)
Samsung Fame Lite	Android 4.12	Muy baja (descatalogado)
iPad Air	iOS 9.2.1	Baja

[36] Fuente propia

Esta aplicación ha sido probada sobre diferentes dispositivos telefónicos, algunos de ellos descatalogados desde hace varios años del mercado y siendo considerados actualmente de gama muy baja. En la Tabla 5 se muestra una lista de los dispositivos en los que se ha testeado la aplicación indicando el sistema operativo en el que se ha probado y la gama de dispositivos a los que hace referencia. Las pruebas se han realizado tanto a nivel general como conectando los dispositivos al servidor.

La aplicación testeada sobre todos estos dispositivos tiene una pantalla de carga de Unity3D que no se puede desactivar al estar usando una versión gratuita, esta pantalla de carga ralentiza la carga inicial de la aplicación, este tiempo de carga es bastante notable en los dispositivos de gama baja. Después de esto el movimiento de la cámara, el refresco de valores con el servidor y los envíos de datos son instantáneos y no hay retardo. En el caso de la mano Shadow, debido a tener que ejecutar un planificador para realizar el movimiento, se nota un cambio de tiempo entre el movimiento de la mano simulada en Linux y cuando se envió la orden (en ocasiones de segundos de diferencia). Este inconveniente no está relacionado con los dispositivos utilizados, también el envío de datos a la mano es rápido, la ralentización es debida principalmente a la planificación necesaria. La actualización de las posiciones de los *joints* en el resto de dispositivos conectados se realiza cuando la mano se está moviendo y los valores de los /*jointStates* se modifican.

En los dispositivos más antiguos el teclado del sistema es mucho más lento y tarda en reaccionar. En estos mismos tipos de dispositivos la cámara no se movía siempre con fluidez cuando se seleccionaba un *joint* concreto como pivote, sin embargo, los usuarios no tuvieron queja puesto que normalmente les van peor las aplicaciones que usan actualmente. La rotación de la cámara se podría arreglar realizando una interpolación y yendo una ejecución por detrás del sistema, de ésta manera el usuario no notaría el retardo y la aplicación se vería fluida siempre.

En cuanto a la interfaz se ha comprobado que se acomodan correctamente todos los botones y las ventanas en cualquier resolución.

En comparación con el proyecto realizado el año se puede afirmar que en los dispositivos se ha notado un incremento en la velocidad de carga al haber cambiado el lector URDF, siendo más notable cuanto más antiguo es el dispositivo. Con respecto a la aplicación anterior tarda un segundo menos en cargar el URDF, en el mejor de los casos.

Aunque el uso principal del sistema está pensado para aplicaciones móviles se ha comprobado que la aplicación es también ejecutable desde ordenadores de sobremesa, en particular se ha probado en ordenadores con Sistema operativo Windows 10 y Linux 16.04 Xenial Xerus. Sin tener que resaltar nada en particular, los ordenadores al tener una mayor potencia ejecutan la aplicación con gran fluidez y rapidez. En la Figura 32 se muestra un ejemplo en el que se ejecuta la aplicación en un ordenador sobremesa.

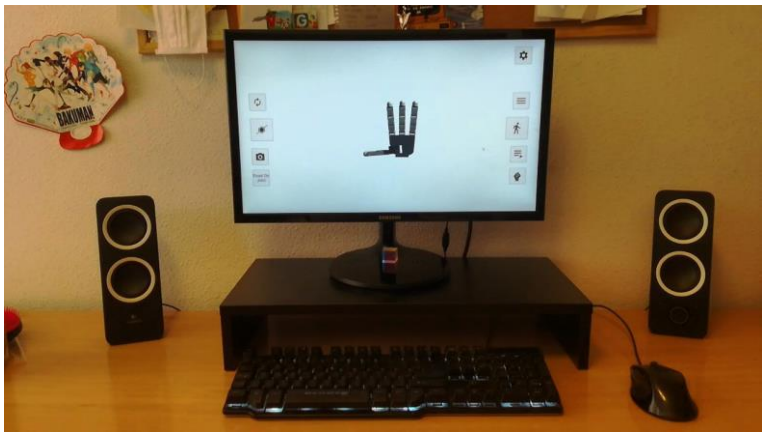


Figura 32: Aplicación ejecutándose en un ordenador de sobremesa

2.6.3. Pruebas de interconexión cliente servidor

En este apartado se muestran algunas pruebas de interconexión entre el cliente y el servidor utilizando los sistemas simulados y los reales. Primero utilizando un solo dispositivo, después usando varios y finalmente ejemplos moviendo las simulaciones mediante la publicación de las señales EMG.

En este apartado se recogen algunas pruebas de las que se han realizado durante el proceso de desarrollo con intención de ilustrar su funcionamiento de la manera más general posible.

2.6.3.1. Usando un solo dispositivo

A continuación se muestran pruebas del sistema usando un único dispositivo telefónico en la mano real de Shadow utilizando un Huawei P9 Lite (Figura 33) y en una simulación de Allegro utilizando un Motorola G3 (Figura 34).

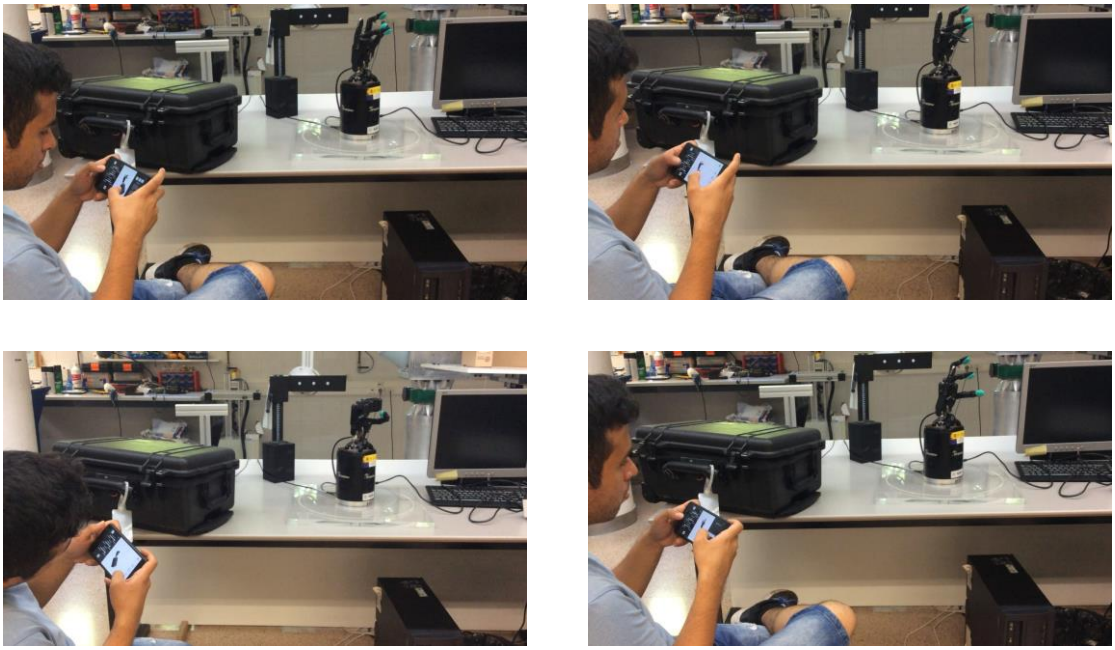


Figura 33: Movimiento de la mano Shadow desde un dispositivo telefónico por un usuario

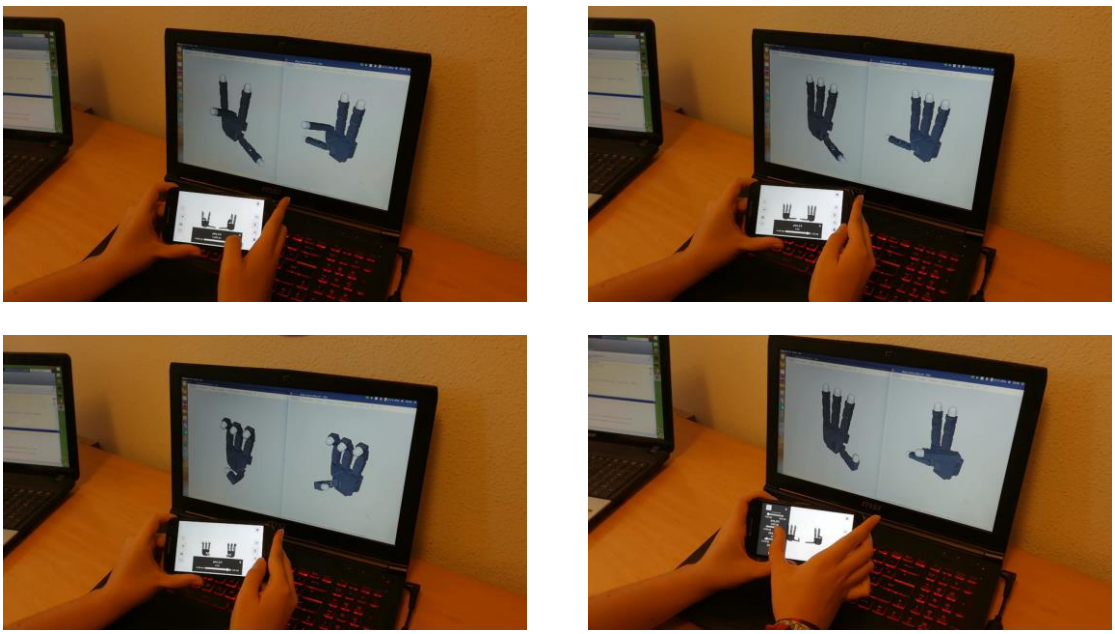


Figura 34: Movimiento de dos manos Allegro de manera simultánea con un dispositivo telefónico

2.6.3.2. Usando varios dispositivos en ambiente colaborativo Allegro

En este apartado se han probado varios dispositivos ejecutando órdenes a una mano Allegro en un ambiente colaborativo, como podemos comprobar en la Figura 35: Allegro en entorno colaborativo ambos usuarios envían datos a la mano y actualizan las posiciones actuales en las que se encuentra el sistema sin problema. Los sistemas usados son un Huawei P9 Lite y un ordenador con Windows 10.

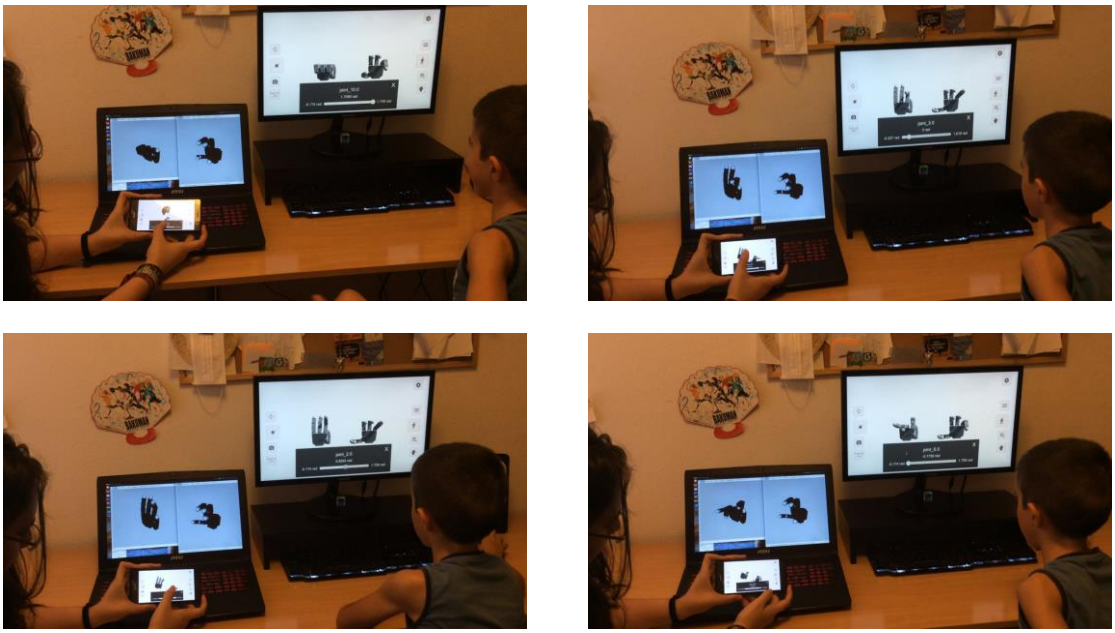


Figura 35: Allegro en entorno colaborativo

2.6.3.3. Usando varios dispositivos en Shadow

En este apartado se muestra varios dispositivos operando la mano Shadow en simulación. (Figura 36)

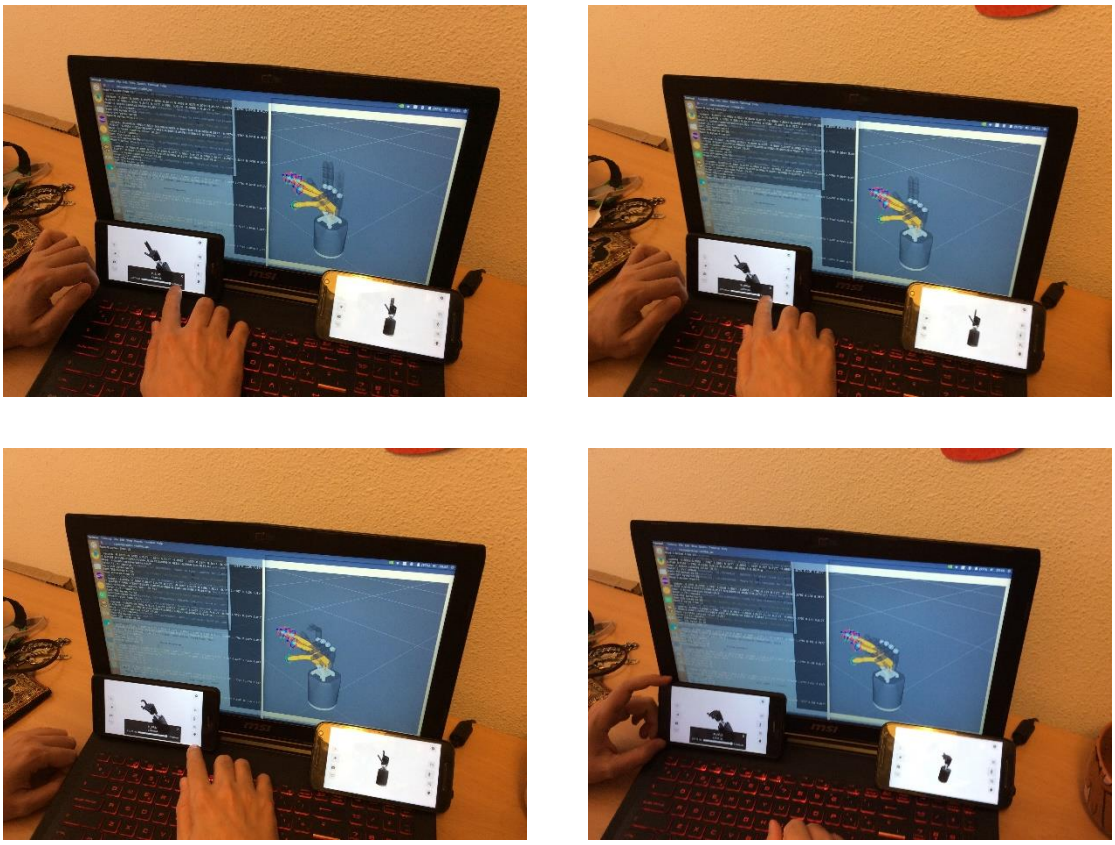


Figura 36: Uso de la mano Shadow con varios dispositivos

Cuando se ejecutan dos manos simultáneas en el caso de usar la mano Shadow, al ejecutarlo dentro de una máquina virtual con una mano simulada se nota una mayor ralentización en el sistema.

2.6.3.4. Prueba con la mano Allegro real

En este apartado se muestra un ejemplo de ejecución con la mano real (Figura 37), aunque esta prueba se grabó el año pasado y no se ha podido grabar una nueva por problemas técnicos, el sistema sigue publicando y suscribiéndose a los mismos *topics*. Lo único que cambia es el identificador con el que se llama, en el caso de la mano izquierda a 1 y en el de la derecha 0.

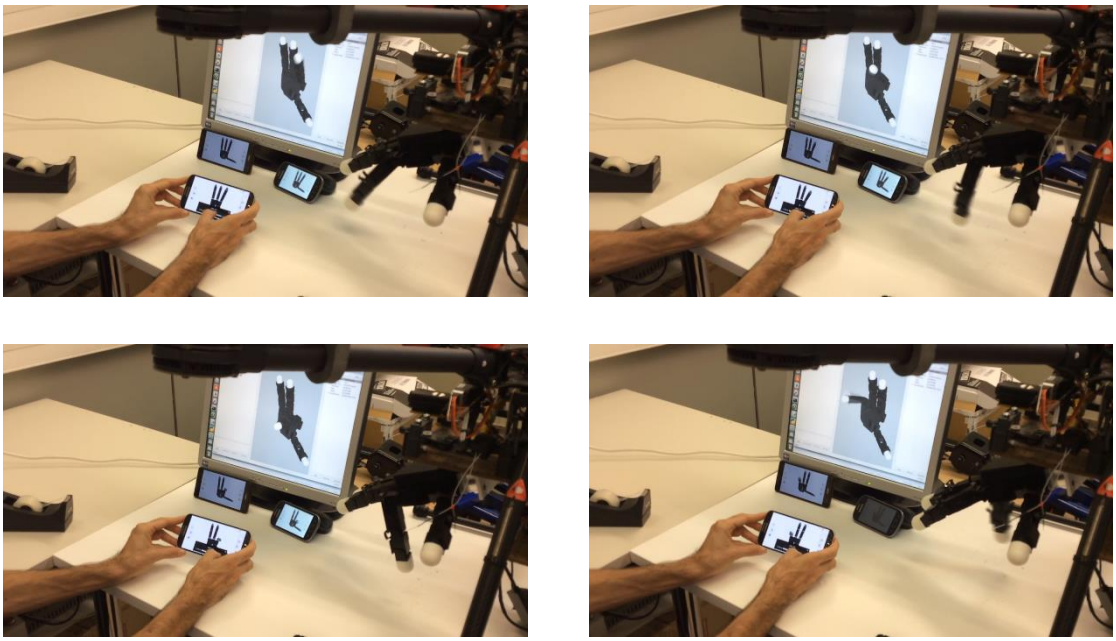


Figura 37: Ejecución en la mano Allegro real

2.6.3.5. Conclusiones de la primera parte de experimentos

Cuando se ejecuta la mano real Shadow podemos encontrar una mejora en cuanto al control y el refresco de información con respecto a la Shadow simulada.

En cuanto a la actualización de datos en la mano Allegro encontramos que los envíos y las actualizaciones son muy rápidos.

El sistema funciona correctamente en todas las pruebas realizadas tanto en manos simuladas como en manos reales.

Existe un error dentro del planificador de Shadow en el cual, la mano puede llegar a determinada posición según Rviz pero en Gazebo la posición no llega a realizarse correctamente. A partir de ahí es necesario volver a ejecutar el controlador. Este error no se produce con las manos reales. Es un error producido al intentar simular el +/-1º dentro de la simulación.

2.6.4. Pruebas usando el nodo de señales EMG

Las pruebas que se presentan en este apartado consisten en realizar el control utilizando los sistemas simulados y publicando en el *topic* que contiene la información de la apertura o el cierre de las señales EMG en Allegro (Figura 38) y en Shadow (Figura 39).

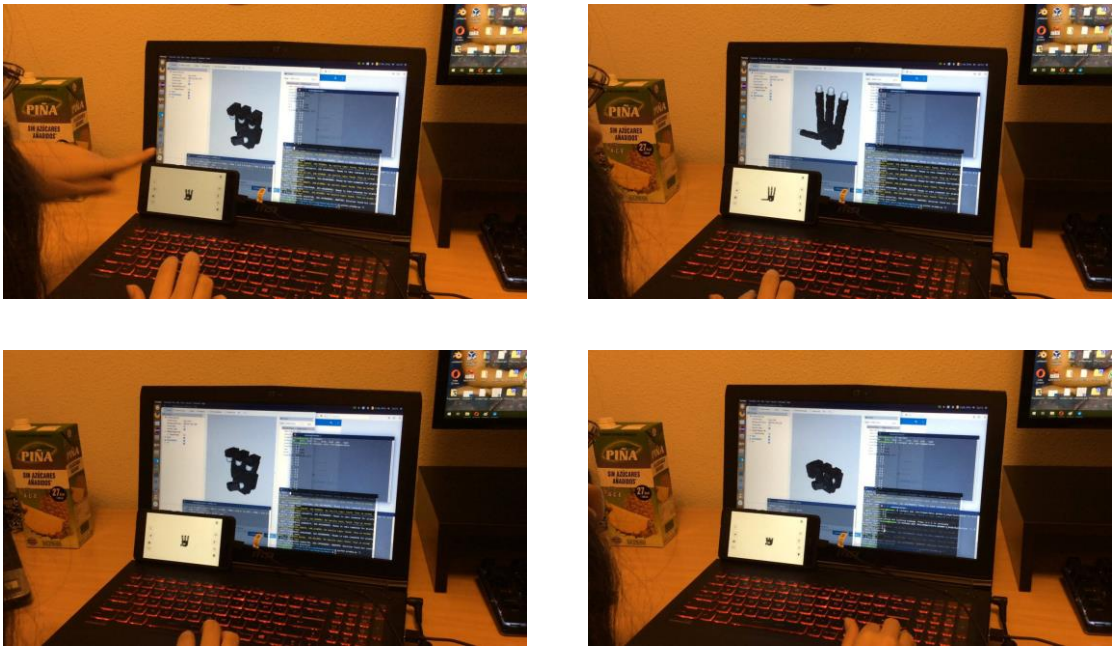


Figura 38: Secuencia de apertura y cierre mediante publicación en el topic de las señales EMG[37]

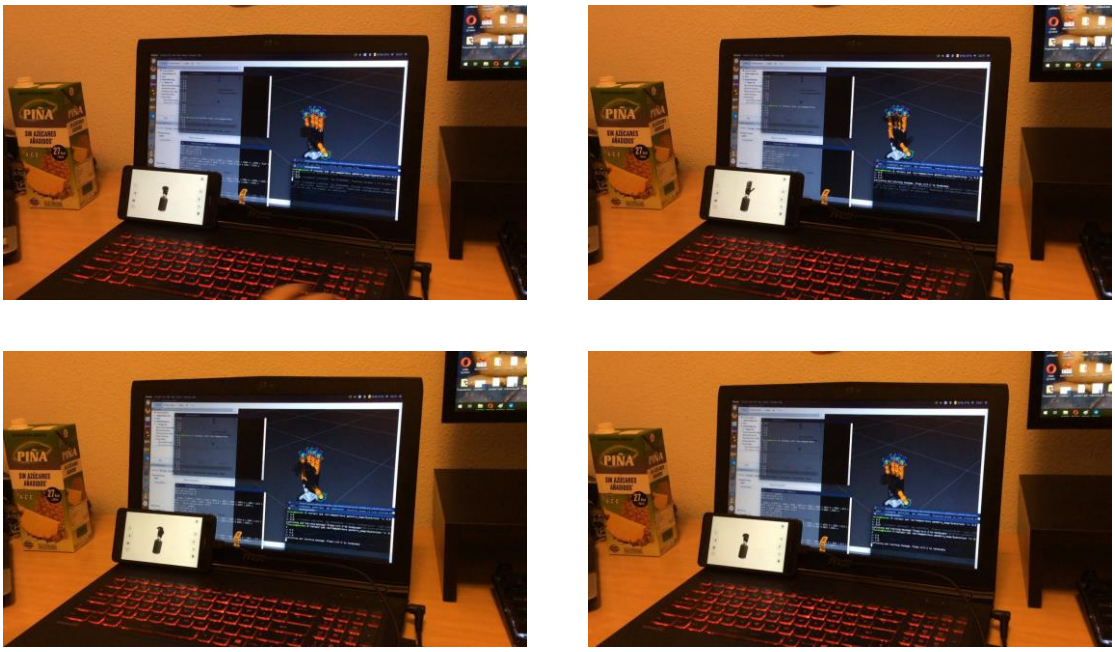


Figura 39: Secuencia de apertura y cierre mediante la publicación en el topic de las señales EMG

[37] Fuente propia

2.6.4.1. Conclusión de la segunda parte de experimentos

En las pruebas que se muestran actualmente son todas sobre sistemas simulados. En este caso se ha publicado en el *topic* de ROS que contiene la información de apertura y de cierre la información.

Como se aprecia en las diferentes pruebas realizadas todos los sistemas tienen una buena tasa de refresco y tienen actualizada la posición actual de la mano cuando se ejecutan los movimientos realizados por el envío de señales EMG. Esto también se realiza adecuadamente cuando hay más usuarios utilizando el sistema.

3. Conclusiones

Como conclusión se ha podido realizar un sistema multiplataforma que funciona en una gran variedad de dispositivos, en *smartphones* y en ordenadores de sobremesa. La interfaz se adapta perfectamente a cualquier tamaño de pantalla. Los usuarios fueron capaces de poder ejecutar sin problema la aplicación en sus dispositivos y se pudieron conectar directamente con las manos virtualizadas en los ordenadores. La interfaz desarrollada en Unity3D permite de manera bastante intuitiva poder realizar el control de una o varias manos. Como se puede ver en las pruebas en cuanto al diseño hay algunas mejoras que se podrían realizar y que se tienen en cuenta en el apartado de trabajos futuros.

El sistema desarrollado permite poder realizar el control de la mano Allegro y de la mano Shadow además de poder realizar el control de manera simultánea de dos manos utilizando el controlador de ROS. En cuanto el control de la mano Allegro tiene respuestas muy rápidas, permitiendo sin problemas el control en un ambiente colaborativo, en el que al mover dos dedos diferentes dos usuarios, se suman las posiciones de la mano. Se permite la visualización en tiempo real de las posiciones actuales en las que se encuentra el *jointStates*. En cuanto a la mano Shadow el sistema es más lento ya que necesita utilizar un planeador para realizar los movimientos. El control de la mano Shadow se realizó aparte y se envían las posiciones a las que actualizar mediante el paso de mensajes de ROS a un nodo en python. Shadow recomienda el uso de phyton para utilizar el control, sin embargo, para realizar el servidor en este lenguaje podría ralentizar el sistema. Como intento para ver si mejora la velocidad en la que se publican los datos en la mano Shadow cuando se está simulando se podría intentar desde los nodos del servidor publicar en el *topic* que se corresponde al planificador del sistema en Rviz, fijándose en la estructura actual de los nodos de la mano Shadow, de manera que se pueda seguir usando el planificador pero disminuya el tiempo de respuesta ligeramente, ya que se ha comprobado que el paso de mensajes no es lento. Aunque es posible realizar los movimientos sin utilizar el planificador esto no es nada recomendable debido a que se pueden producir choques a grandes velocidades rompiendo la mano real. En cuanto al sistema real la planificación y la actualización de las posiciones es mucho más rápida que en la mano simulada.

Aunque Shadow recomiende el uso de python y prepare los módulos específicos para los usuarios en python la compañía utiliza por debajo C++ puesto que es mucho más eficiente.

También se ha permitido de manera satisfactoria el envío de datos cuando se registre un cambio de posición en una señal EMG. Esto permite poder realizar el envío de datos dentro del sistema sin que falle, es posible poder tener conectado únicamente el servidor a los clientes sin ninguna mano y enviar las órdenes de control a uno de los dispositivos conectados. Recibir órdenes de apertura y de cierre mediante la publicación del módulo es útil para poder realizar entrenamientos de prótesis. La idea de incluir la posibilidad de poder cargar una textura de piel humana es para hacer sentir más cómodo al usuario que está realizando el entrenamiento de la prótesis permitiendo poder realizar el movimiento y visualizar el resultado final de la mano que se está usando como prótesis.

La mano Shadow real funciona mucho más rápido que la mano Shadow simulada, además que no se producen errores a la hora de realizar los movimientos.

Para probar el sistema en la mano Allegro ha habido problemas de disponibilidad, sin embargo, se sabe que el sistema funcionaría puesto que el año pasado se hicieron pruebas y la publicación se hace exactamente a los mismos *topics*. Por lo que se incluye una imagen del control de la Allegro realizado con anterioridad.

Se ha redactado varios artículos sobre este proyecto, actualmente hay uno enviado a las jornadas de automática y robótica, otro se está redactando en inglés para la revista Displays.

3.1. Trabajos futuros

En este apartado se mencionan una serie de trabajos futuros que enriquecería la aplicación y se explica a grandes rasgos cómo se desarrollarían para poder usarlo. Algunos de estos trabajos futuros no se han podido implementar por falta de tiempo.

- Poder interactuar con objetos tanto sólidos como deformables en la aplicación de simulación.

  Este punto lo permite Unity3D de por sí, por lo que no sería necesario tener que cambiar el sistema. Bastaría realizar la carga de objetos 3D ya sean deformables o sólidos y aplicarles las fuerzas correspondientes.

- Selección por parte del cliente de la interpolación utilizada en la aplicación de simulación.

  Para realizar esta mejora sería recomendable utilizar una librería que nos permitiría realizar cálculos matriciales complejos, se implementarían todos los interpoladores vistos en la asignatura de control y programación de robots.

- Creación de grupos de actuación dentro de la aplicación.

  Selección dentro de la aplicación de las articulaciones y carga de los sliders de control correspondientes para controlar únicamente un grupo de articulaciones.

- Poder seleccionar el punto sobre el que se rota

  Cambiar el punto de actuación sobre el que se rota la cámara haciendo. Para ello cuando se haga doble tap sobre determinada posición, se guardará ese espacio tridimensional sobre el que rotar, poniendo como profundidad la profundidad a media a la que se encuentre la mano.

- Mover los objetos insertados en la aplicación por el espacio 3D

  Mediante la carga de un objeto que contenga las líneas que correspondan a los vectores x, y y z mover el objeto en el plano marcado.

- Permitir el uso de realidad aumentada

  Unity3D permite su uso, sería necesario preparar la interfaz para su uso sin problemas

- Recibir información de señales de presión

  ROS puede enviar datos sobre la presión ejercida, así pues se podría recibir esa información desde la conexión establecida y poder reflejarla sobre los objetos cargados en la aplicación o mostrada en la interfaz a modo informativo. También realizar el control teniendo en cuenta esos datos de presión.

- Añadir más opciones de control como usar un joystick o por ejemplo usar guantes

  Se tendrían que usar los datos en forma articular si no vienen dados en esta forma se haría la transformación usando la información recibida en las asignaturas de control y programación de robots o en robótica, también sería necesario una transformación desde el sistema de control que se vaya a usar al robot real, utilizando el modelo matemático del robot. Tras esto, se enviarían los datos al dispositivo telefónico para que se realizase el control.

- Implementación de algoritmos de agarre dentro del sistema

  Implementar dentro del sistema alguno de los algoritmos de agarre que existen actualmente y poder reproducirlos frente a los diferentes objetos en Unity3D.

- Posibilidad de carga de diferentes sistemas robóticos en el caso de la aplicación de escritorio

  Actualmente se ha creado un URDF que puede cargar gran cantidad de sistemas robóticos, debido a las limitaciones de los dispositivos telefónicos no se ha podido añadir esta opción, sin embargo, en un ordenador se podría cargar seleccionando la carpeta donde está almacenado cualquier URDF y su correspondiente modelo. Sería necesario para luego realizar la conexión un pequeño ajuste para indicarle al servidor el robot con el que se está trabajando.

- Probar a utilizar los nodos de ROS para realizar agarres utilizando Graspit

  Se realizará la conexión con los nodos de Graspit para recibir las posiciones en las que se encuentra el sistema

- Arreglar posibles problemas que puedan surgir cuando el cliente que hace de *updater* de desconecte

  En el servidor se sabe cuándo un cliente se ha desconectado. Tendría que situarse en otro cliente de la lista de clientes y notificarle que es el nuevo *updater*.

- Permitir a los usuarios definir secuencias de control personalizadas para el control mediante señales EMG.

  Permitir que el cliente asigne una orden a una secuencia. Esta orden de control será enviada por el servidor cuando se cumpla determinadas condiciones dentro del *topic* en el que se publican los datos de las señales EMG.

- Utilizar una base de datos para sustituir los ficheros txt donde se almacena la información.

  Se utilizaría la base de datos que viene por defecto en .Net para almacenar los datos de las secuencias, las vistas y también las preferencias. Este cambio agilizaría el proceso de carga en caso de tener una gran cantidad de secuencias y de vistas.

- Marcar en el modelo 3D el *joint* que se está moviendo.

  Actualmente el slider de movimiento tiene los datos de la articulación a la que se hace referencia, sería seleccionar ese *gameObject* y resaltarlo cambiando la manera en la que se renderiza, podría ser cambiándola de color o iluminándola. El color para marcar la articulación debe ser un color que permita poderse distinguir teniendo en cuenta los problemas de daltonismo y de visión en escala de grises. Para ello se utilizará una web que permita comprobar el color de fondo y la articulación.

- Implementar las mejoras propuestas por los usuarios

  Implementar las mejoras propuestas con el fin de hacer más usable e intuitiva la interfaz. Una vez hecho esto volverá a probar la interfaz, con otros usuarios.

- Añadir seguridad mediante el cifrado de datos

  La información pasa del cliente al servidor sin cifrar, es fácil poder utilizar un software que te permita poder capturar paquetes y modificarlos para realizar otras acciones haciendo hasta que el robot falle o se rompa. Para hacer esto se podría utilizar un algoritmo de cifrado ya conocido que permitiera a los clientes comunicarse al ser los únicos que tienen las claves de cifrado para la comunicación. Para elegir un algoritmo se puede ver (Microsoft, s.f.)

- Sistema de usuarios con clave

  Actualmente para conectarse se utiliza la dirección IP. Esta dirección se puede saber muy fácilmente, para aumentar la seguridad se podría añadir un sistema de usuarios y claves dentro de una base de datos del servidor para comprobar el acceso. Estos datos dentro del servidor se almacenarían cifrados con el cifrado de la base de datos por defecto.

- Intentar mejorar la velocidad de la mano Shadow

  Se elimina el módulo realizado en python y se comunica directamente con el sistema que se encarga de realizar el planeador en Rviz.

- Posibilidad de poder guardar una IP conocida en una lista que se carga al inicio

  Permitiría al usuario poder guardar una IP conocida dentro de una lista para luego poder seleccionarla, además de poder eliminarlas o modificarlas. Sería recomendable almacenar los datos en una base de datos. Esto mejoraría la experiencia en los dispositivos descatalogados, en los cuales se ha notado un retardo a la hora de escribir las direcciones IP.

De estas mejoras las más importantes son aquellas que comprometen la seguridad y que mejoran la experiencia de usuario haciendo la interfaz más amigable y usable o mejorando la velocidad de respuesta de las conexiones de las manos.

4. Apéndices

4.1. Comandos de terminal para ROS

El paquete que nos permite moverse entre paquetes de manera muy similar a como nos moveríamos en el sistema de archivos de GNU/Linux se llama *rosbash*. Este permite realizar comandos como por ejemplo *roscd* para movernos a otra carpeta de ROS.

Es muy importante saber los comandos básicos en terminal para poder ver el grafo computacional del entorno. De esta manera podemos saber el tipo de mensajes que se envían, quienes son los *topics*, los nodos...

Esto facilita mucho el desarrollo para poder conectarse a determinado robot existente desde tu propio programa y poder realizar el control de éste.

4.1.1. Rosnode

Los comandos que comiencen por *rosnode* aportan información sobre los diferentes nodos que hay actualmente en el sistema.

• **rosnode info nombreDeNodo**: Da información sobre el nodo

• **rosnode kill nombreDeNodo**: Mata a un nodo en ejecución.

• **rosnode list**: Muestra por pantalla todos los nodos activos.

• **rosnode machine hostname**: Muestra por pantalla todos los nodos que están ejecutándose en determinada máquina o muestra una lista de las maquinas.

• **rosnode ping nombreDeNodo**: Prueba la conexión de un nodo.

• **rosnode cleanup**: Elimina la información de registro de los nodos que no son accesibles.

4.1.2. Rostopic

Los comandos que empiezan por *rostopic* dan información sobre los *topics*.

• **rostopic bw /topic**: Muestra el ancho de banda que usa determinado *topic*.

• **rostopic echo /topic**: Muestra el contenido de los mensajes por pantalla.

• **rostopic find tipoDeMensaje**: Encuentra *topics* por su tipo de mensaje.

• **rostopic hz /topic**: Muestra el ratio de publicación de determinado *topic*.

• **rostopic info /topic**: Muestra información sobre determinado *topic* activo, los *topics* a los que publica, los servicios, a quienes está suscrito.

• **rostopic list**: Muestra información de los *topics* activos.

• **rostopic pub /topic tipo parametros**: Publica en determinado *topic*, mediante línea de comandos.

• **rostopic type /topic**: Imprime por pantalla el tipo de mensaje que publica determinado *topic*.

4.1.3. Rosmsg

Este comando aporta información sobre los mensajes.

• **rosmsg show**: Muestra los campos que componen el mensaje.

• **rosmsg list**: Muestra una lista de todos los mensajes.

• **rosmsg package**: Muestra todos los mensajes en el paquete.

• **rosmsg packages**: Muestra todos los paquetes que tengan ese mismo mensaje.

• **rosmsg users**: Busca todos los archivos de código que usan ese tipo de mensaje.

4.1.4. rqt_graph

Este comando consiste en mostrar información sobre los nodos y sobre los *topics*. Pudiendo ver que nodos están comunicados usando determinados *topics*.

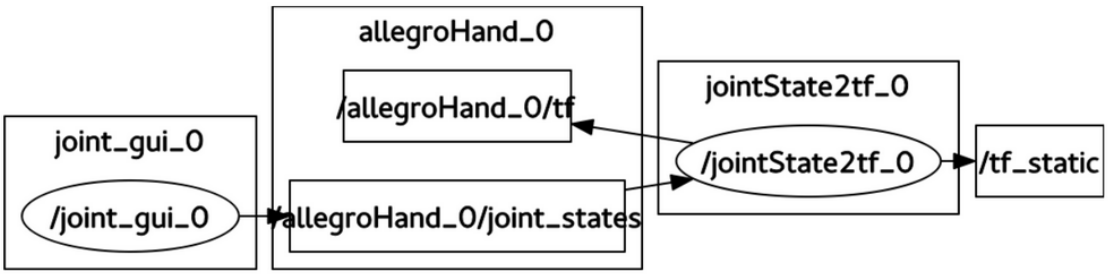


Figura 40: Con los nodos y los topics activos[38]

Figura 41: Con los nodos activos[39]

[38] Fuente propia
[39] Fuente propia

Se puede seleccionar ver los nodos y *topics* activos, como en la Figura 40, ver solo los nodos, tal y como se muestra en la Figura 41 o visualizar todos los nodos y los *topics*. Esto mostraría un grafo con una información similar a la Figura 40

En el caso de la Figura 40 los nodos están representados en elipses dentro de rectángulos y los *topics* están en rectángulos dentro de rectángulos. En la Figura 41 los *topics* están representados colocando su nombre sobre una flecha únicamente si hay un nodo que está suscrito y uno que publique en el.

4.1.5. Otros comandos

Hay una gran variedad de comandos en ROS que permiten realizar diferentes acciones desde terminal, todo ello se puede consultar en la wiki de ROS.

Vamos a explicar a continuación de manera muy general los comandos que aportan información sobre los servicios y los bags, puesto que se han mencionado anteriormente pero no se han usado durante el desarrollo de este proyecto.

4.1.6. Comandos para los Servicios

Existen dos líneas de comandos para obtener información de los servicios, rossrv y rosservice. Con rossrv se puede ver información de la estructura interna de los datos de los servicios, tiene la misma estructura que rosmsg. Rosservice se utiliza para mostrar y consultar servicios.

4.1.7. Comandos para Bags

Para trabajar con Bags se usan dos comandos: rxbag, que muestra los datos en el entorno gráfico y rosbag, que se usa para guardar, ejecutar, etc.

4.2. Entendiendo las etiquetas de un archivo URDF

URDF posee diferentes etiquetas, a continuación vamos a explicar cada una de ellas y se entrará en detalle en las más importantes: link, joint y robot. Para este anexo se ha tenido como referencia para realizar la maquetación y sacar la información (ROS.org, s.f.)

4.2.1. Elemento link.

Este elemento es el que se encarga de describir el cuerpo rígido, así como su inercia y su aspecto visual. Podemos ver una descripción gráfica de los elementos principales que componen un link en la Figura 42.

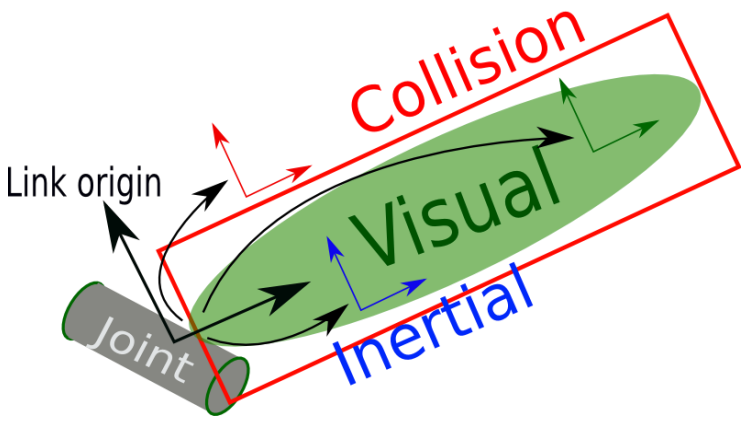


Figura 42: Elemento Link[40]

4.2.1.1. Atributos:

name (*obligatorio*)

- El nombre que tiene el *link*.

4.2.1.2. Elementos dentro de link

<inertial> (*opcional*)

- Las propiedades de inercia del link.

<origin> (*opcional: Por defecto identidad*)

- Esta es la posición del marco de referencia, relativo al marco de referencia del link. El origen del marco de referencia debe de estar en el centro de gravedad. Los ejes de la referencia inercial no necesitan estar alineados con los ejes principales de la inercia.

[40] http://wiki.ros.org/urdf/XML/link

xyz (*opcional: Por defecto Vector3 zero)*

- Posición en los ejes x, y, z.

rpy (*opcional: por defecto valor identidad*)

- Representa los ejes de rotación, están en radianes.

<mass>

- El valor de la masa del *link*.

<inertia>

- La matriz de rotación 3x3, representada en el marco de inercia. Ya que esta matriz es simétrica solo están especificados 6 elementos (ixx, ixy, ixz, iyy, iyz, izz).

<visual> (*opcional*)

- Son las propiedades visuales del link. Este elemento especifica la forma del objeto, es posible tener más de una etiqueta visual en un mismo link.

  name (*opcional*)

  - Especifica el nombre de una parte de la geometría del link. Es útil para hacer referencia a una parte específica de la geometría.

  <origin> (*opcional: por defecto la identidad*,)

  - El marco de referencia del elemento visual con respecto al marco de referencia del link. Posee en su interior las mismas etiquetas, que el de inertia.

  <**geometry**> (*obligatoria*)

  - La forma del objeto visual, puede ser una de las siguientes

    <box>

    size este atributo contiene las tres longitudes de cada lado del cubo. El origen del cubo está situado en su centro.

    <**cylinder**>

    Con **radius** y **length** se especifica el radio y el largo del cilindro. El origen del cilindro está en su centro.

    <**sphere**>

radius es el radio de la esfera. El origen esta en su centro.

<**mesh**>

- Viene especificado por un **filename** y opcionalmente por **scale** éste escala el tamaño usando los ejes del bounding box. El formato recomendado puesto que tiene mejor textura y color es Collada .dae, aunque también soporta .stl. El archivo debe ser local.

<**material**> (*opcional*)

- El material del elemento visual. Esta permitido especificar un material fuera del elemento link, en el nivel superior (elemento robot), y referenciar al material por el nombre.

  name nombre del material.

  <**color**> (*opcional*)

  rgba Color del material especificado en los canales rojo, verde, azul y alfa. Debe estar en un rango de 0 a 1.

  <**texture**> (*opcional*)

  La textura del material está especificada en un archivo, **filename**.

<**collision**> (*optional*)

- Propiedad de colisión de un link. Esta puede ser diferente de las propiedades visuales del link, por ejemplo los modelos de colisión simples son usados para reducir el tiempo de computación. Es posible tener varias etiquetas collision en un mismo link.

  name (*opcional*) Nombre de la geometría del link. Es útil para hacer referencia a la geometría específica del link.

  <**origin**> (*opcional: por defecto la identidad*)

  - El marco de referencia del elemento collision con respecto al marco de referencia del link. Posee en su interior las mismas etiquetas, que el de inertia.

  <**geometry**>

  - Tiene la misma descripción que el elemento visual.

4.2.2. Elemento Joint

El elemento joint es la articulación del robot, describe la cinemática y la dinámica del joint además de especificar los límites de seguridad del mismo. Podemos ver una descripción visual de elemento joint en la Figura 43.

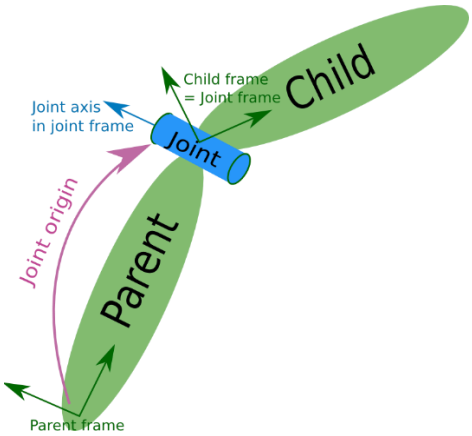


Figura 43: Elemento Joint[41]

4.2.2.1. Atributos

Este elemento posee dos atributos:

name (*obligatorio*)

- Especifica un nombre único para el joint

type (*obligatorio*)

- Especifica el tipo de joint que puede ser uno de los siguientes:

  revolute: Una articulación de bisagra que gira a lo largo del eje y está limitada por un rango especificado por los limites superior e inferior

  continuous: Una articulación de bisagra continua que gira al rededor del eje y no tiene límites superiores ni inferiores.

  prismatic: Es una articulación deslizante. Se desliza a lo largo del eje y tienen un límite superior e inferior.

[41] http://wiki.ros.org/urdf/XML/joint

fixed: Esto no es una articulación puesto que realmente no se mueve. Todos los grados de libertad están bloqueados. Este tipo de articulación no necesita ejes, calibración, dinámicas, límites o controles de seguridad.

floating: Permite movimiento en los 6 grados de libertad.

planar: Permite el movimiento en un plano perpendicular al eje.

4.2.2.2. Elementos

Estos son los diferentes elementos que puede poseer joint.

<**origin**> (*opcional: por defecto la identidad*)

- Es la transformación desde el link padre al link hijo. El joint está situado en el origen del link hijo. Tal y como se muestra en la Figura 43.

  xyz (*opcional: por defecto un vector cero*)

  - Representa el desplazamiento en x, y, z.

  rpy (*opcional: por defecto un vector cero*)

  - Representa la rotación alrededor de unos ejes fijos: primero se rota alrededor del eje x, después en el y y finalmente en el z. Todos los ángulos están especificados en radianes.

<**parent**> (*obligatorio*)

- El nombre del padre como atributo obligatorio:

  link

  - El nombre del link que es el padre de este joint, este estará situado en la estructura de árbol del robot.

<**child**> (*obligatorio*)

- El nombre del link hijo como atributo obligatorio.

  link

  - El nombre del link que es el link hijo de este joint.

<**axis**> (*opcional: por defecto (1,0,0)*)

- Este es el eje de rotación o traslación de los tipos de articulaciones que hemos visto anteriormente: revolute, prismatic, continuous y planar. El eje es especificado en un mismo marco de referencia. Como ya se dijo anteriormente las articulaciones fixed y floating no utilizan este campo.

  xyz (*obligatorio*)

  - Representa las componentes x, y, z de un vector. Este vector debe estar normalizado.

<**calibration**> (*opcional*)

- La posición de referencia de la articulación, se usa para calibrar la posición absoluta de esta.

  rising (*opcional*)

  - Cuando una articulación se mueve en una dirección positiva, esta posición de referencia desencadenará en un borde ascendente.

  falling (*opcional*)

  - Cuando un joint se mueve en una dirección positiva, esta posición de referencia desencadenará en un borde descendente.

<**dynamics**> (*opcional*)

- El elemento que especifica las propiedades físicas de la articulación. Estos valores se usan para especificar las las propiedades de la articulación, es algo muy útil para la simulación.

  damping (*opcional, por defecto 0*)

  - Para el valor de la amortiguación física de la articulación (N·s m para articulaciones de tipo prismatic, N·m·s rad para aquellas de tipo revolute).

  friction (*opcional, por defecto 0*)

  - El valor de la fricción estática de la articulación (N para las articulaciones prismatic, N·m para las articulaciones de tipo revolute).

<**limit**> (*obligatorio solo para articulaciones de tipo revolute y prismatic*)

- Puede contener los siguientes atributos:

lower (*opcional, por defecto 0*)

- Este atributo especifica el límite inferior de la articulación. Se utilizan radianes para las articulaciones de tipo revolute, metros para las prismatic. Se omite si es un joint de tipo continuous.

upper *(opcional, por defecto 0*)

- Especifica el límite superior del joint. Las unidades de medida son las mismas que en lower.

effort (*obligatorio*)

- Un atributo para el máximo esfuerzo de la articulación el esfuerzo aplicado siempre debe ser menor que este valor.

velocity (*obligatorio*)

- Atributo para mostrar la velocidad máxima a la que puede moverse la articulación.

<**mimic**> (*opcional*) (Nuevo a partir de ROS Groovy.)

- Esta etiqueta se usa para especificar que la articulación imita a otra articulación existente. El valor de esta articulación se puede calcular como valor=multiplicador∗otrovalordearticulacion+compensador .

joint (*obligatorio*)

- Especifica el nombre de la articulación que va a imitar.

multiplier (*opcional*)

- Especifica el multiplicador de la fórmula de arriba. Por defecto 1.

offset (*opcional*)

- Especifica la compensación que hay que añadir en la fórmula de arriba. Por defecto 0.

<**safety_controller**> (*opcional*)

soft_lower_limit (*opcional, por defecto 0*)

- Este atributo especifica el límite inferior de la articulación donde el controlador de seguridad comienza a limitar la posición de este. Este límite necesita ser más grande que el atributo lower, visto anteriormente.

soft_upper_limit (*opcional, por defecto 0*)

- Este atributo especifica el límite superior de la articulación donde el controlador de seguridad comienza a limitar la posición de este. Este límite necesita ser menor que el atributo upper visto anteriormente.

k_position (*opcional, por defecto 0*)

- Este atributo especifica la relación entre los límites de posición y velocidad.

k_velocity (*obligatorio*)

- Este atributo especifica la relación entre los límites de esfuerzo y velocidad.

4.2.3. Robot

Es la etiqueta principal del URDF, consiste en general en un conjunto de etiquetas joint y etiquetas link. Aunque también puede contener las etiquetas transmission y gazebo

4.2.4. Transmission

Es una extensión de la descripción del robot, se utiliza para describir la relación entre un transmisor de fuerza y una articulación. Esto permite al modelo conceptos como relaciones de engranajes y conexiones en paralelo. Muchos transmisores de fuerza pueden estar unidos con múltiples articulaciones a través de una compleja transmisión.

4.2.5. Gazebo

El elemento gazebo es una extensión del formato de descripción de robots URDF, usado para simular en el simulador Gazebo.

4.2.6. Sensor

Este elemento describe las propiedades básicas de un sensor visual. Sin embargo este elemento no ha sido utilizado en ninguna aplicación. El proyecto para mejorar esta etiqueta

ha sido abandonado. En la web de Ros se pide a cualquier persona que lo desee, seguir con el desarrollo y extenderlo para que funcione en aplicaciones que tengan un sensor hardware.

4.2.7. Elemento model_state

Este elemento describe el estado básico de un modelo URDF. Está en desarrollo actualmente. La representación de las configuraciones de un grupo particular de joints puede ser también especificado usando SDF.

4.3. Ejecución del Sistema

Para ejecutar el sistema lo primero que se hará es lanzar las manos Allegro o Shadow que se quieran lanzar en el servidor. Hay que tener en cuenta que en un ROS Master sólo se puede lanzar una mano Shadow. Los comandos de lanzamiento son los siguientes para el caso de usar manos simuladas en el caso de la Allegro:

```
roslaunch allegro_hand_joint_gui_virtual.launch HAND:=right GROOVY:=true polling:=false NUM:=0
```

```
roslaunch allegro_hand_joint_gui_virtual.launch HAND:=left GROOVY:=true polling:=false NUM:=1
```

Para lanzar la mano Shadow se ejecutarán los siguientes comandos para la mano derecha o izquierdas simuladas.

```
roslaunch sr_robot_launch srhand.launch
```

Los nodos de las manos Allegro se matan automáticamente al ejecutar la aplicación ya que internamente se llama a un archivo .sh que se encargará de ejecutar esas órdenes en terminal:

Una vez lanzadas las manos se tendrá que ejecutar en el caso de haber usado una mano Shadow nodo de python realizado en ROS, para ello habrá que dirigirse a la carpeta dónde está guardado y ejecutarlo.

```
cd /catkin_ws/src/python_programa/src
```

```
python prueba.py
```

Tras esto tenemos el sistema preparado y será necesario ejecutar nuestro servidor con las órdenes:

```
rosrun dprg_ros_real dprg_ros_real_node tipo shadowmano
```

Los parámetros usados para este apartado son los siguientes:

- Tipo: Esto se usa para discernir entre la mano real o simulada, recordemos que los nodos a los que suscribe el sistema en la mano Allegro serán diferentes dependiendo de si es una mano real o una simulada.
- Shadowmano: Este parámetro se usa para distinguir la mano del nodo en la que se está ejecutando, dependiendo de si es derecha o izquierda se publicará la información en un nodo u en otro.

Para lanzar el módulo de Matlab hay que tener ejecutando en el mismo ordenador Matlab y en una máquina virtual Ubuntu con el sistema de nodos. Se ejecuta el módulo proporcionado y ya se tendría el sistema de señales EMG.

Si se quieren lanzar dos manos Shadows simultáneamente se lanzara en el otro ordenador la mano Shadow correspondiente y el nodo de python con los mismos parámetros vistos anteriormente y por último se lanzará el cliente con la orden:

```
rosrun shadowmirror shadowmirror shadowmano serverdir
```

En esa orden es necesario indicarle al sistema la mano en que se está ejecutando, así como la dirección del servidor.

4.4. Manual de la aplicación

En este apartado se muestra con mayor detalle cómo utilizar las diferentes opciones de la interfaz desarrollada en Unity3D. Para ello se va a ver paso a paso las opciones implementadas.

Cuando se abre la aplicación aparecen las opciones de trabajar sin conexión o conectarse (Figura 44). Si se selecciona esta opción es posible introducir la IP del ordenador con el que se quiera conectar para realizar el control de las manos. Si se trabaja sin conexión se trabajará de manera simulada en el dispositivo telefónico, esta opción permite poder crear secuencias, mover los dedos y reproducir secuencias ya creadas sin tener la necesidad de estar conectado a la red.

Figura 44: Selección el método de trabajo[42]

Una vez seleccionada el método de trabajo se puede seleccionar el tipo de mano con la que se trabajará (Figura 45), la mano Shadow o la mano Allegro y con qué manos se va a trabajar, con la izquierda, la derecha o las dos simultáneamente. Es importante tener en cuenta que aplicaciones están conectadas actualmente ya que en el caso de no estar en el servidor las órdenes no verán reflejadas en el programa.

[42] Fuente propia

Figura 45: Mano/s con la/s que se va a trabajar[43]

Una vez hecho esto se cargará la ventana principal. En ésta ventana se han implementado diferentes controles de cámara (Figura 46). Es posible poder hacer zoom en la escena poniendo dos dedos sobre la pantalla y acercándolos o alejándolos entre sí. Para rotar la cámara se pulsa sobre la escena con un dedo y se desplaza éste hacia la dirección que se quiera mover la cámara.

Figura 46: Controles de cámara[44]

La cámara de manera general rota sobre el centro de los modelos cargados, se puede rotar la cámara sobre determinada articulación seleccionando el botón inferior derecho. Éste te abre una ventana con una lista de las articulaciones cargadas, si se pulsa sobre una de las opciones de la lista la cámara se fija en ellas (Figura 47). Es posible dejar el pivote como viene al inicio pulsando el botón inferior que aparece en ese menú.

[43] Fuente porpia
[44] Fuente propia

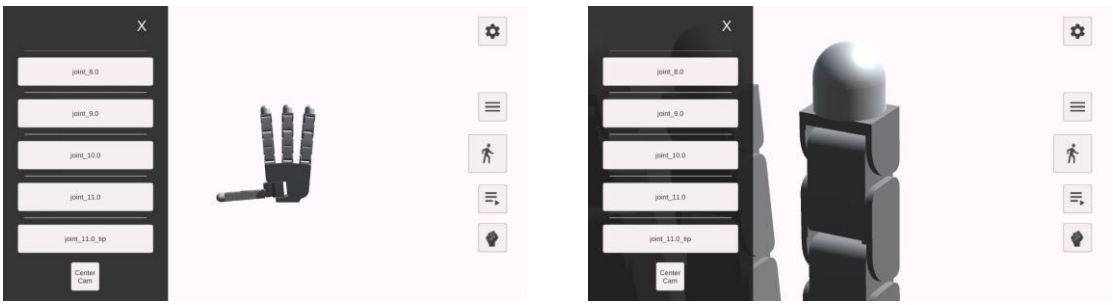


Figura 47: Selección de pivote de la cámra[45]

Por comodidad también se ha creado la opción de poder guardar, cargar o eliminar una vista que es la posición actual en la que se encuentra la cámara. Para gestionar las vistas está el menú de vistas (Figura 48) situado en la penúltima posición del lado izquierdo.

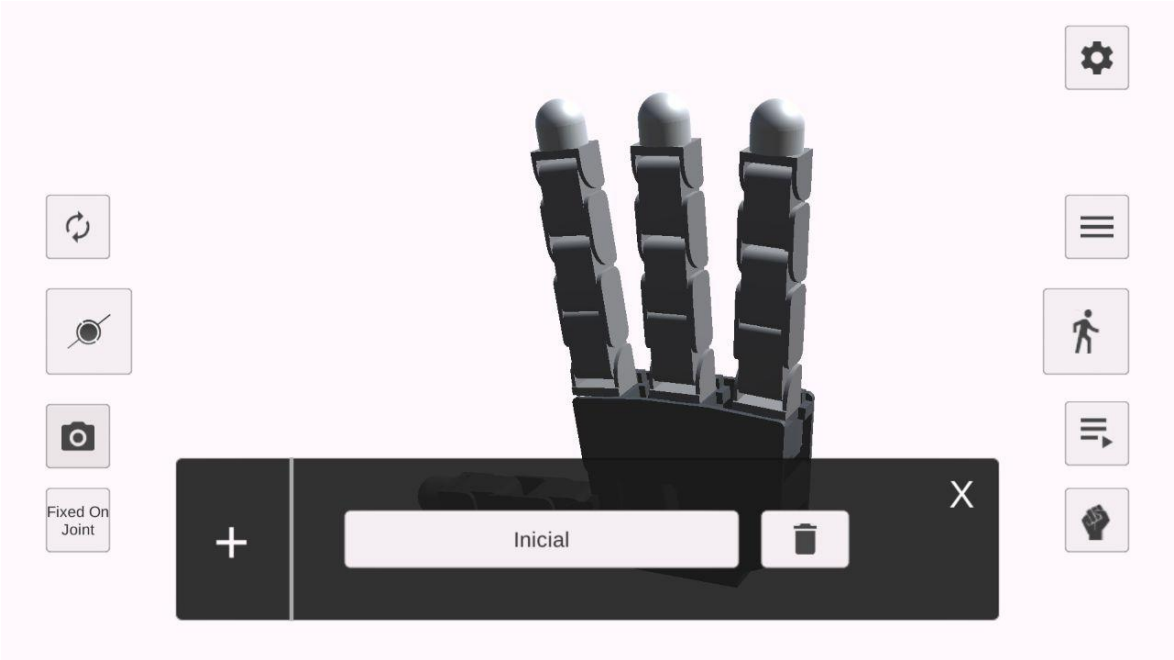


Figura 48: Menú de vistas[46]

En este menú se puede crear una nueva pulsando el botón de + para su creación será necesario introducir un nombre que no contenga espacios, el signo = y que no esté repetido (Figura 49).

[45] Fuente propia
[46] Fuente propia

Cuando se pulsa sobre una de las vistas cargadas la cámara se posicionará en la posición almacenada anteriormente.

Figura 49: Creación de una nueva vista[47]

Para mover las articulaciones de la mano existen dos maneras utilizando los deslizadores que se encuentran en el segundo botón del lado izquierdo de la aplicación (al seleccionar el menú de articulaciones) o al seleccionar la articulación que se quiera mover directamente de la interfaz en cuyo caso aparecerá un deslizador (Figura 50).

Figura 50: Ejemplos de movimientos de articulaciones[48]

[47] Fuente propia
[48] Fuente propia

Para resetear las posiciones a una mano con todos los dedos completamente extendidos se puede pulsar la opción de resetear las posiciones indicadas por el icono de las dos flechas.

En esta aplicación es posible poder crear y ejecutar secuencias para crear una secuencia se van guardando pasos por cada posición que se quiera tener en la secuencia. Para crear pasos se pulsará a los icónos que representan a una pesona andando. Después desde el menú de pasos se muestran los pasos que se han creado y se posibilita poder crear más, eliminar el último o eliminar todos los pasos existenes. También es posible poder guardar una secuencia. Para guardar una secuencia es necesario introducir un nombre que no contenga espacios, no esté repetido y no contenga el carácter =, además las secuencias deben contener al menos un paso. (Figura 51)

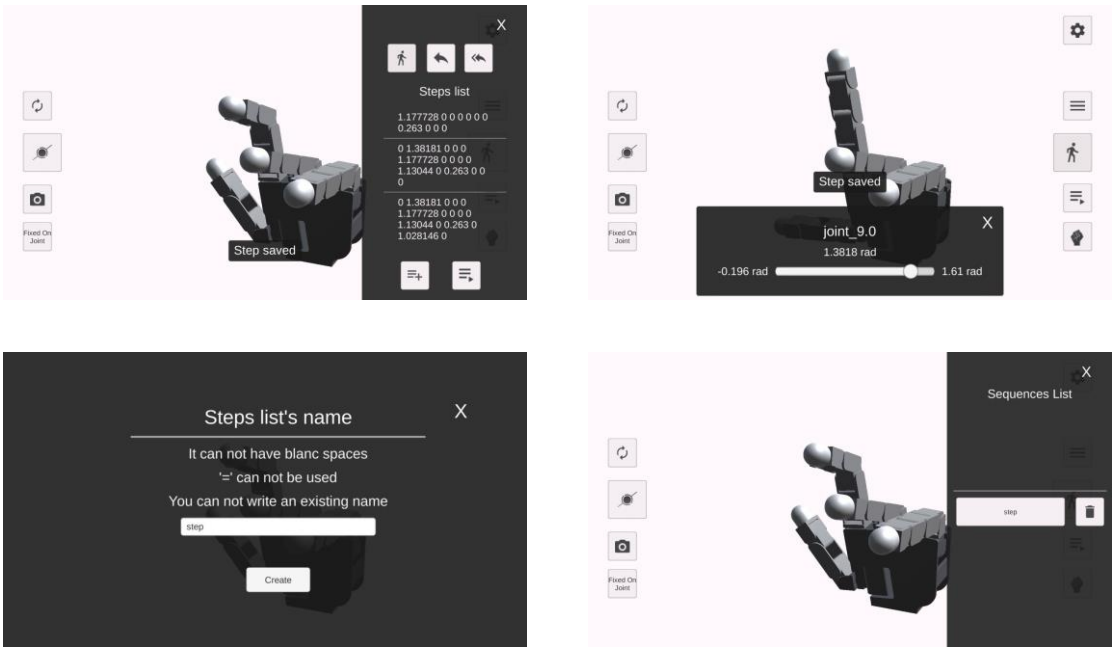


Figura 51: Creación de nuevos pasos, guardado y reproduccion de secuencias[49]

Para reproducir las secuencias se puede acceder mediante el menú de secuencias. Al pulsar sobre el nombre de la secuencia se reproducirá, si se vuelve a pulsar sobre el la secuencia se dejará de reproducir. Para poder reproducir una secuencia es imprescindible que este menú se mantenga abierto. Desde aquí es posible poder eliminar una secuencia.

[49] Fuente propia

En la parte inferior derecha se encuentra el botón que permite ejecutar una secuencia de movimiento predefinida en la que cuando se pulse por primera vez se ejecutará la secuencia que permite cerrar el puño y si se vuelve a pulsar se abrirá la mano. (Figura 52)

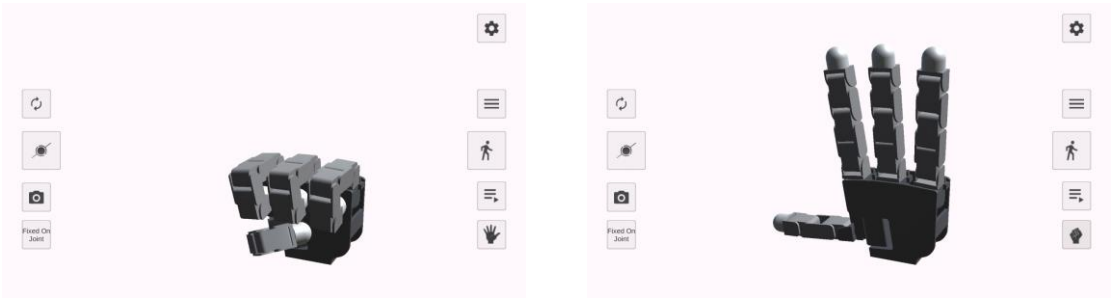


Figura 52: Ejecución de secuencias predefinidas[50]

Se ha implementado un menú de opciones, situado en el lado derecho con las funciones descritas en los anteriores apartados.

[50] Fuente propia

4.5. Ejecución de la mano Allegro en Linux

La visualización del entorno 3D en GNU/Linux de la mano Allegro se realiza con Rviz. Para poder ejecutar la mano primero será necesario ir a la carpeta allegro_hand_ros y ejecutar el setup.bash en la terminal de Ubuntu. De esta manera será capaz de saber dónde se localizan los archivos que necesita.

```
cd ~/allegro_ws/allegro_hand_ros
source ~/allegro_ws/setup.bash
```

Sólo es necesario ejecutar los comandos anteriores una vez por terminal.

Se van a explicar a continuación los diferentes parámetros para ejecutar la mano robot.

El primer comando que se debe escribir para lanzar la aplicación es roslaunch. Este se utiliza para lanzar las aplicaciones de ROS.

- **allegro_hand.launch**: Lanza la mano Allegro Hand con un controlador especifico por defecto grasp. Además de lanzar un controlador por teclado y el visualizador de Rviz
- **allegro_hand_noRviz.launch:** Es igual que el primero pero no lanza Rviz.
- **allegro_hand_joint_gui.launch:** Por defecto lanza el controlador PD. Además de la interfaz para controlar cada joint con sus límites y el visualizador de Rviz.
- **allegro_hand_joint_gui_virtual.launch**: Lanza el modelo cinemático de la mano Allegro, además de la interfaz para controlar los *joints* y el visualizador Rviz.

Todos los launchers anteriores incluyendo la mano virtual y real tienen los siguientes argumentos:

- HAND:= (Para seleccionar la mano que va a ser controlada, debe ser especificado siempre)
    - Right
    - Left
- NUM:= (Se usa para enumerar las manos cuando se van a usar varias/ Evita los conflictos)
    - 0 (por defecto)
    - Cualquier numero entero (1, 2, 3, ...)
- GROOVY:= (Especifica si es la distribución más antigua que GROOVY o no)
    - true (por defecto, usa ROS Fuerte)
    - false
- CONTROLLER:= (Especifica el controlador)
    - grasp (por defecto)
    - pd
    - velSat
- polling:= (Especifica si la comunicación CAN se lleva a cabo usando polling, es decir de manera síncrona y sin el uso de interrupciones)
    - true (por defecto)
    - false
- ZEROS:= (Especifica el codificador / el desplazamiento y las direcciones del archivo que contiene los parámetros
    - zero.yaml (por defecto)
    - Debe estar dentro la ruta actual /allegro_ws/allegro_hand_ros y tener en cuenta desde ahí donde esta guardado el archivo *.yaml (ejemplo
    - "parameters/zero_files/zero_SAH020CR020.yaml")

- CAN_CH:= (Especifica el canal CAN al que esta conectada la mano)
    - /dev/pcan32 (por defecto)
    - otro canal PEAK CAN (en /dev/*)

Un ejemplo de ejecución de los comandos es el siguiente:

```
roslaunch allegro_hand_joint_gui_virtual.launch HAND:=right GROOVY:=true
polling:=false
```

Este comando lanzaría el entorno de simulación virtual Rviz, de la mano derecha robótica en ROS, así como, el panel de control de las articulaciones que componen el modelo, tal y como se puede ver en la figura 5.

4.6. Ejecución de la mano Shadow en Linux

Para la ejecución de la mano en simulación basta con escribir por terminal:

```
roslaunch sr_robot_launch srhand.launch
```

Esto ejecutará una mano de tipo Shadow Motor. En la Tabla 6 se muestran todas las manos disponibles para ejecutar la simulación. Y en la Tabla 7 los diferentes .xacros que tendrían que ser llamados para su carga.

Tabla 6: Tipos de manos Shadow[51]

									
1	2	3	4	5	6	7	8	9	10

Tabla 7: Posibles ejecuciones de mano Shadow[52]

	Right	Left
1	shadowhand_motor.urdf.xacro	shadowhand_left_motor.urdf.xacro
2	shadowhand_motor_biotac.urdf.xacro	shadowhand_left_motor_biotac.urdf.xacro
3	shadowhand_motor_ff_biotac.urdf.xacro	
4	shadowhand_motor_btsp.urdf.xacro	
5	shadowhand_motor_ellipsoid.urdf.xacro	
6	shadowhand_motor_th_ff_rf_ellipsoid.urdf.xacro	
7	shadowhand_muscle.urdf.xacro	shadowhand_left_muscle.urdf.xacro
8	shadowhand_muscle_biotac.urdf.xacro	
9	shadowhand_lite.urdf.xacro	
10	shadowhand_extra_lite.urdf.xacro	

Para ejecutar una simulación de la mano se puede ejecutar:

```
roslaunch sr_robot_launch srhand.launch robot_description:=`rospack find sr_description`/robots/shadowhand_motor.urdf.xacro
```

[51] http://shadow-robot.readthedocs.io/en/latest/generated/sr_interface/sr_robot_launch/README.html
[52] http://shadow-robot.readthedocs.io/en/latest/generated/sr_interface/sr_robot_launch/README.html

El parámetro de robot_description puede ser cambiado para ejecutar cualquier Shadow que aparece en la Tabla 7.

Si se ejecuta una mano izquierda se tiene que añadir el parámetro hand_id:=lh

Para no utilizar moveit se puede cambiar el parámetro use_moveit:=false

5. Bibliografía

Adams, J. A., 2002. Critical Considerations for Human-Robot Interface Development. *AAAI Technical Report.*

Aguero , C. et al., 2015. Inside the virtual robotics challenge: Simulating real- time robotic disaster response.. *IEEE Transactions on Automation Science and Engineering,* Volume 12, pp. 494-506.

Andaluz, V. H. et al., 2016. Unity3D-MatLab Simulator in Real Time for Robotics Applications. In: De Paolis L., Mongelli A. *Augmented Reality, Virtual Reality, and Computer Graphics. AVR 2016. Lecture Notes in Computer Science,* Volume 9768.

Cecere, G., Corrocher, N. & Battaglia, R. D., 2015. Innovation and competition in the smartphone industry: Is there a dominant design?. *Telecommunications Policy,* 39(3 - 4), pp. 162-175.

Codd-Downey, R. & Jenkin, M., 2015. CON: Dynamic Mobile Interfaces for Command and Control of ROS-enabled Robots. *12th International Conference on Informatics in Control, Automation and Robotics (ICINCO).*

Codd-Downey, R. et al., 2014. From ROS to Unity: leveraging robot and virtual environment middleware for immersive teleoperation. *IEEE International Conference on Information and Automation (ICIA),* pp. 932-936.

Consorcio OPENSURG Proyecto iberoamericano para la docencia e investigación en robótica médica utilizando recursos de código abierto Acción CYTED 509AC0372, 2013. Capítulo 4 Tratamiento de señales biomédicas. In: CYTED, ed. *Robótica médica notas prácticas para el aprendizaje de la robótica en bioingeniería.* Elche: s.n., pp. 110-130.

Corke, P., 2015. Integrating ROS and MATLAB [ROS Topics]. *IEEE Robotics & Automation Magazine,* 22(2), pp. 18 - 20.

Craighead, J., Burke, J. & Murphy, R., 2008. *Using the Unity Game Engine to Develop SARGE: A Case Study.* s.l., Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008).

Gaikwad, V. T. & Sardeshmukh, M. M., 2014. Sing languaje recognition based on electromyography (EMG) signal using artificial neuronal network (ANN). *International Journal of Industrial Electronics and Electrical Engineering,* 2(6).

Graupe, D. & Cline, W. K., 1975. Functional Separation of EMG Signals via ARMA Identification Methods for Prosthesis Control Purposes. *IEEE,* SMC-5(2), pp. 252-259.

Green, S. A., Billinghurst , M., Chen, X. & Chase , J. G., 2008. Human-Robot Collaboration: A Literature Review and Augmented Reality Approach in Design. *International Journal of Advanced Robotic Systems,* Volume 5, pp. 1-18.

Hirt, J., Berns , K. & Mianowski, K., 2012. Designing Arms and Hands for the Humanoid Robot ROMAN. *Advanced Materials Research,* Volume 463, p. 1233–1237.

Hu, Y. & Meng, W., 2016. ROSUnitySim: Development and Experimentation of a Real-time Simulator for Multi-UAV Local Planning. *Simulation,* 92(10), pp. 931-944.

Hu, Y. & Wei, M., 2016. ROSUnitySim: Development and experimentation of a real-time simulator for multi-unmanned aerial vehicle local planning. *Simulation,* 92(10), pp. 931-944.

Iliukhin , V. N., Mitkovskii , K. B., Bizyanova , D. A. & Akopyan , A. A., 2017. The Development of Motion Capture System Based on Kinect Sensor and Bluetooth-Gloves. *Procedia Engineering,,* Volume 176, pp. 506-5013.

Meng , W. et al., 2015. *ROS+Unity: An Efficient High-fidelity 3D Multi-UAV Navigation and Control Simulator in GPS-Denied Environments.* s.l., Industrial Electronics Society, IECON 2015 - 41st Annual Conference of the IEEE.

Microsoft, n.d. *Microsoft Docs.* [Online]
Available at: https://docs.microsoft.com/es-es/sql/relational-databases/security/encryption/choose-an-encryption-algorithm?view=sql-server-2014
[Accessed 2018].

Miller, A. & Allen, P. K., 2004. Graspit!: A Versatile Simulator for Robotic Grasping. *IEEE Robotics and Automation Magazine,* Volume 11, pp. 110-122.

Quigley, M. et al., 2009. ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software.*

Ramaswamy, C. & Deborah, S., 2015. A Survey of Robotic Hand-Arm Systems. *International Journal of Computer Applications,* Volume 109, pp. 26-31.

ROS.org, n.d. *wiki de ros.* [Online]
Available at: http://wiki.ros.org/urdf/XML
[Accessed Julio 2018].

Salvietti, G. et al., n.d. HANDS.DVI: A DeVice-Independent Programming and Control Framework for Robotic HANDS. *Gearing up and accelerating cross-fertilization between academic and industrial robotics research in Europe: Technology transfer experiments from the ECHORD project,* pp. 197-215.

Virgala, I., Kelemen, M., Varga, M. & Kurylo, P., 2014. Analyzing, Modeling and Simulation of Humanoid Robot Hand Motion. *Procedia Engineering,* Volume 96, pp. 489-499.