Desarrollo y despliegue de un módulo OPC-UA en ROS.

Autor
ESTER LÓPEZ MORA
(esterlopezlm12@gmail.com)

Director:
JUAN FRANCISCO BLANES NOGUERA
(pblanes@ai2.upv.es)

TRABAJO FIN DE MÁSTER
MÁSTER DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Valencia, 6 Diciembre 2020

Dedicatoria

Quiero dedicar este TFM a mis padres, a mi hermana, a mi tía Llanos y a toda mi familia, sin ellos este TFM no habría sido posible. También a mis amigos, y a mi equipo, CFF Maritim, por todos los buenos momentos que me hacen pasar tanto dentro como fuera del campo.

AGRADECIMIENTOS

Quiero agradecer el desarrollo de este trabajo, a todos mis profesores del Máster en Automática e Informática Indsutrial y en especial a mi tutor, por participar de manera activa en mi formación y desarrollo académico y profesional.

TABLA DE CONTENIDO

LISTA DE ESQUEMAS

LISTA DE FIGURAS

RESUMEN EXTENDIDO

En este trabajo se ha desarrollado un sistema que permite la integración de los parámetros de un robot cualquiera en un servidor que utilice el protocolo de comunicación OPC-UA.

Se han estudiado las distintas tecnologías que intervienen, cuál es su estado del arte y una breve historia de las mismas.

El objetivo del trabajo era desarrollar una aplicación con el sistema operativo robótico ROS, que contara con todas las características necesarias para ser plenamente funcional. Sobre este proyecto ROS se ha añadido el protocolo de comunicación OPC-UA. Este protocolo OPC-UA cuenta con un servidor que es el eje central del proyecto. Este servidor se encarga de tomar los datos del robot que se quiera monitorizar, de almacenarlos en una dirección y puerto concretos que sean accesibles por el usuario de una manera correcta y así se evitan todos los posibles errores de compatibilidad de datos que puedan surgir entre el protocolo OPC-UA y el entorno robótico de programación ROS.

Además se ha incuido una manera de poder configurar el servidor sin necesidad de reprogramar el código cada vez que se quiera solicitar un dato nuevo, o se cambie el robot a parametrizar.

En resumen la aplicación servidor desarrollada tiene las siguientes funcionalidades:

1. Tomar los datos que se requieran de ROS
2. Autoconfigurarse mediante un archivo de configuración.
3. Almacenar los datos en una dirección IP y puerto concretos.
4. Refrescar los datos en los tiempos adecuados de acuerdo al tipo de dato con el que se esté trabajando.

En el transcurso del desarrollo del proyecto se han realizado pruebas intermedias que han permitido ir verificando la validez del sistema y la corrección de todos los errores que han surgido durante el desarrollo.

También se han realizado finalmente unas pruebas que han permitido comprobar el sistema desarrollado en su totalidad.

1. INTRODUCCION

Actualmente es mucho más frecuente encontrar robots colaborativos en los entornos industriales y de producción de las empresas. Esto se debe principalmente a tres razones: su versatilidad, su facilidad de programación y su facilidad de instalación. Además, es importante añadir que los robots colaborativos se caracterizan por el hecho de que no es necesario la instalación de barreras de seguridad u otras estructuras auxiliares al robot, más allá del propio robot. Por tanto, es muy habitual encontrar estos robots en líneas productivas realizando tareas que hasta ahora las hacían otras máquinas. Sin embargo, no tienen el grado de integración en los sistemas de información empresariales que pueden llegar a tener otros, como por ejemplo los PLCs, por lo que se mantienen en un aislamiento lógico sus parámetros de operación en el ámbito del controlador del robot, lo que dificulta su integración con los sistemas globales de supervisión y control de operaciones.

Bajo estas premisas y teniendo en cuenta que la integración en los sistemas de información de muchos dispositivos industriales se hace con el protocolo OPC-UA, debido a que es independiente de la tecnología de los mismos, en este trabajo se pretende hacer una integración similar de los robots colaborativos. Específicamente, la propuesta es integrar los parámetros de control de los robots usando el pseudo-sistema operativo ROS (Robot Operating System), mediante el que se puede acceder a la información correspondiente al robot e implementar dicha información en un módulo OPC-UA, ya que este protocolo es uno de los más usados a nivel industrial. Por ello, el objetivo de este trabajo es integrar la información de control de cualquier robot colaborativo en el sistema de información de una planta industrial para evitar así el aislamiento previamente descrito.

Este documento se compone de diez capítulos, más un último capítulo donde se incluyen los anexos correspondientes al proyecto. El primer capítulo es el actual e incluye una breve introducción. En el segundo capítulo se explica la justificación que tiene el proyecto teniendo en cuenta la situación actual. En el tercer capítulo se describen los objetivos del proyecto, tanto el objetivo general, como la descomposición del mismo en objetivos específicos. El cuarto capítulo versa sobre el actual estado del arte de la distintas tecnologías que intervienen en el proyecto, así como una breve historia de la mismas. En el capítulo cinco se recoge todo el desarrollo del proyecto, que se ha estructurado de una manera similar a la filosfía en la que se basa la ciencia de las matemáticas: partir de conocimientos básicos, para poco a poco, y apoyándose en esos conocimientos, llegar a estructuras mucho más complejas. El capítulo seis se reserva para exponer las pruebas realizadas para poder verificar la validez del sistema desarrollado y mostrar los resultados; mientras que en el capítulo siete se discutirán y compararán esos resultados con los resultados que se pretendían obtener teniendo en cuenta la bibliografía consultada. En el capítulo ocho se encuentran todas las conclusiones a las que se han llegado durante el desarrollo del proyecto, así como aquellas que se obtuvieron tras realizar las pruebas pertinentes, y la evaluación de si se han cumplido o no los objetivos del proyecto. En el capítulo nueve se realizan unas breves recomendaciones en caso de que alguien quisiera seguir mejorando este sistema. Por último, el capítulo diez recoge toda la bibliografia utilizada.

2. JUSTIFICACION

Este proyecto nace de la necesidad que existe en los entornos industriales de poder acceder a toda la información de los robots colaborativos; y de que esta información no sólo sea accesible por los operarios de planta sino por cualquier otro usuario de la empresa que así lo precise, así como por otros dispositivos que puedan necesitar la información derivada de los robots colaborativos para llevar a cabo sus tareas.

El desarrollo realizado beneficia a todas aquellas empresas que integren entre sus disposistivos de trabajo robots colaborativos y otros dispositivos no compatibles con la tecnología ROS pero sí con OPC-UA, de manera que la información, tanto de unos como de otros, pueda ser recogida en un servidor de la red del entorno industrial y, por tanto, accesible por cualquier usuario o por otros dispositivos que precisen de dicha información.

La metodología del trabajo está basada en tres fases básicas:

1. Análisis del estado del arte
2. Desarrollo de la propuesta
3. Redacción de la memoria

La primera fase fue una recopilación de información sobre el estado del arte de las distintas tecnologías que se han utilizado en este proyecto para tener un mejor conocimiento de éstas, para facilitar los futuros desarrollos. Tanto la segunda fase como la tercera fase han sido realizadas concurrentemente, puesto que cada vez que se desarrollaba un contenido del proyecto se redactaba y explicaba en la memoria la manera en qué se había desarrollado. Gracias a ello, se ha podido plasmar con claridad y precisión el trabajo llevado a cabo. Durante el desarrollo de los contenidos se han ido realizando pruebas intermedias que se encuentran también documentadas en esta memoria, de manera que si se encontraban fallos en lo que se había desarrollado hasta el momento o se planteaba una manera más óptima de obtener los mismos o mejores resultados se aplicaba.

La elección de utilizar OPC-UA para la integración se justifica por su amplio uso en entorno industriales y por la facilidad de integración gracias a que permite una capa de abstracción respecto a la integración de dispositivos.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

El objetivo general de este proyecto es el desarrollo y despliegue de un módulo OPC-UA en ROS para integrar robots en entornos industriales. Es decir, desarrollar un sistema que permita obtener la información de los *topics* de ROS y compartirla mediante el estándar OPC-UA con otros dispositivos que estén fuera de ROS.

3.2 OBJETIVOS ESPECÍFICOS

El objeto general descrito se descompone en los siguientes objetivos específicos:

- Analizar y comprender el funcionamiento del sistema ROS. Se pretende ser capaz de comprender las funcionalidades de ROS, así como tener la capacidad suficiente para poder aplicarlas en proyectos de aplicación real; ser capaz de realizar un proyecto mínimamente funcional que permita, más adelante, desarrollar otros proyectos más complejos.
- Analizar y comprender el estándar OPC-UA. Entender cuál es el problema que este estándar vino a solucionar, así cómo su filosofía de funcionamiento. Evaluar el SDK de desarrollo cliente servidor de open62541. Durante el desarrollo de esta aplicación se pretende también adquiririr los conocimientos suficientes para poder añadir funcionalidades a esta aplicación, así como aprender a trabajar con la documentación asociada a *open62451*.
- Realizar un proyecto que combine las partes de ROS y las partes del servidor. Ser capaz de trasladar todo lo aprendido en ROS para poder crear un proyecto que integre la funcionalidad de *open62451*, y desde ahí comenzar a desarrollar una aplicación servidor dentro del entorno ROS.
- Acceder al servidor desarrollado desde otro tipo de aplicaciones clientes ya existentes. La aplicación servidor creada no cuenta *a priori* con la capacidad de acceder a la misma si no es con una aplicación cliente creada específicamente para ello. Se pretende, por tanto, otorgarle esta funcionalidad, por lo que es necesario hacer uso de los conocimientos adquiridos previamente para poder trabajar con la documentación asociada.
- Modificar el proyecto para poder configurar desde un fichero de tipo *.xml* de qué *topics* de ROS se mostrará la información. Para que la aplicación sea verdaderamente funcional es necesario otorgarle la capacidad de poder configurarse a través de un fichero *.xml*, donde se especifique cuáles van a ser los datos que el usuario pueda consultar a través del servidor. Se hace ahora necesario usar la introspección, y para ello, de manera indirecta, aprender cómo funciona y cuáles son las modificaciones que es necesario hacer en los archivos auxiliares del proyecto, además de en el archivo *.cpp* (archivo que contiene el código).

- Aprender a programar y desarrollar proyectos en *Python* dentro de un sistema operativo *Linux*. Desarrollar una interfaz gráfica que permita la construcción de los ficheros *.xml* donde se muestren los *topics* existentes para suscribirse. Con los conocimientos de *Python* ya adquiridos, desarrollar una aplicación que cuente con interfaz gráfica que permita al usuario, de manera intuitiva y sencilla, crear los archivos de configuración.
- Diseñar y realizar una serie de pruebas tanto teóricas como prácticas que pongan de manifiesto la validez del sistema. En primer lugar, se realizarán una serie de pruebas teóricas donde se verificará si todo el sistema funciona correctamente o es necesario realizar cambios. Posteriormente se realizarán pruebas prácticas con un brazo robótico real, donde se pondrán de manifiesto todas las ventajas e incovenientes del sistema desarrollado cuando se usa en un entorno de trabajo real. Se extraerán todas aquellas mejoras que se podrán realizar a posteriori.
- Examinar todos los resultados obtenidos. Analizar y discutir todos los resultados de manera que se adquieran conocimientos y experiencia en la realización de diserciones sobre proyectos científicos y de ingeniería. Así como dar los motivos sufientes para determinar si el proyecto ha tenido el éxito esperado, o si por el contrario, no lo ha hecho.

4. ESTADO ACTUAL

Antes de comenzar el desarrollo del proyecto es necesario conocer las partes principales que intervienen en él, así como un poco de historia de las mismas y, sobre todo, el actual estado del arte en el que se encuentran.

4.1 ROS. ROBOT OPERATING SYSTEM

ROS es un sistema que proporciona bibliotecas y herramientas para poder desarrollar el software necesario para llevar a cabo aplicaciones robóticas.

Se puede definir como un middleware robótico, es decir, una colección de frameworks para el desarrollo de software de robots.

A pesar de no ser un sistema operativo propiamente dicho, provee los servicios estándar de los mismos como son la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de la funcionalidad de uso común, el paso de mensajes entre procesos, la gestión y el mantenimiento de paquetes, las bibliotecas, los visualizadores, y otros muchos.

ROS fue originalmente diseñado en la Universidad de Stanford, en su laboratorio de inteligencia artificial, en 2007. El objetivo principal de este proyecto fue crear un sistema base que proporcionase el punto de partida y estructuras básicas que facilitaran el proceso de desarrollo de robots. La filosofía de ROS se basa principalmente en los siguientes puntos:

- Par a par: Programas individuales pueden comunicarse sobre APIs definidas (ejemplo. Mensajes ROS, servicios, etc).
- Distribuido: Los programas pueden ejecutarse en distintos computadores y comunicarse a lo largo de la red.
- Multi-lenguaje: Los módulos de ROS pueden estar escritos en cualquier lenguaje de programación para el que existan librerías. Entre ellos los más usados son C/C++ y Python.
- Peso ligero: Se busca que las librerías independientes no ocupen demasiado espacio.
- Software libre: La mayoría de las distribuciones de ROS son gratuitas y de software libre.

ROS es un herramienta que puede usarse en numerosas aplicaciones donde intervienen robots, como por ejemplo aplicaciones de planificación, reconocimiento facial, identificación de objetos, robots móviles, etc.

A lo largo de la historia de ROS se han ido desarrollando distintas versiones, algunas de ellas son incompatibles con otras. La primera de ellas fue *Box Turtle*, que se lanzó el 2 de marzo de 2010, mientras que la última lanzada ha sido ROS *Noetic Ninjemys* (lanzada el 23 de marzo de 2020). La versión escogida para desarrollar el proyecto ha sido ROS *Kinetic Kame* (fecha de lanzamiento 23 de mayo de 2016) puesto que se trata de una versión que ya tiene suficientes años como para que haya mucha documentación sobre ella, pero no demasiados años como para que haya quedado obsoleta.

4.2 SISTEMA OPERATIVO LINUX Y UBUNTU.

Linux es la denominación que reciben los sistemas operativos de tipo *Unix* en *Unix*. Se trata de un sistema operativo extremadamente flexible, fácil de extender y modificar, además de ser una plataforma ideal para desarrollar nuevas aplicaciones. Es muy potente pero más complejo de entender. Se trata de un sistemas código abierto ya que todo su código puede ser utilizado, modificado y redistribuido libremente por cualquier persona, empresa o institución bajo los términos de la licencia pública general de GNU. La idea de desarrollar *Linux* vino encabezada por el programador finlandés Linus Torvalds.

Este tipo de sistemas operativos cuentan con una gran capacidad para compilar código fuente de distintos lenguajes de programación como *C, C++, Java, Ada, Pascal,* etc. Además soporta diversas arquitecturas de procesador mediante la compilación cruzada, lo que propicia que sea el entorno adecuado para desarrollos de proyectos heterogéneos como es el caso de este proyecto.

En concreto se va a eligir el sistema operativo *Ubuntu 16.04*, debido a que ROS *Kinetic Kame* sólo está disponible para *Ubuntu 15.*10 y *Ubuntu 16.04*, por lo que mejor escoger una versión más actualizada.

Ubuntu es un sistema opetativo de software libre y código abierto basado en *Debian*. Es muy usado para el desarrollo de servidores, está orientado el usuario, debido a su enfoque en la facilidad de uso y la mejora de la experiencia de usuario.

4.3 OPC-UA. OPEN PROCESS CONTROL UNIFIED ARCHITECTURE Y OPEN62541.

Un problema que aparece continuamente en las plantas industriales es la comunicación entre los distintos dispositivos que existen. Estos dispositivos suelen usar tecnologías software distintas entre sí, pero tienen unos *drivers* propietarios asociados para que la comunicación sea efectiva. Sin embargo, esto no resulta útil, pues se hace necesario comprar un software o unas pasarelas para cada uno de los fabricantes. En este contexto surgió la tecnología OPC, para poder resolver este problema. OPC es un estándar de interfaz de acceso a los servidores de datos de PLC´s, dispositivos de campo, bases de datos, etc. Permite a un único servidor manejar varios dispositivos, todo ello bajo una interfaz uniforme.

El OPC (antiguamente conocido como *OLE for Process Control*), está basado en la tecnología de Microsoft y se concibió en un principio para funcionar solo en los sistemas operativos de *Windows*. Este protocolo está basado en la famosa aplicación cliente-servidor, donde el servidor es la fuente de datos a la que se conectan el resto de los dispositivos (clientes). Con esta aplicación se asegura el intercambio de información con independencia del fabricante del dispositivo.

En las siguientes figuras se ejemplifica el problema sin la tecnología OPC, y la solución que esta aporta.

Figura 1. Problema sin tecnología OPC (fuente: es.wikipedia.org)

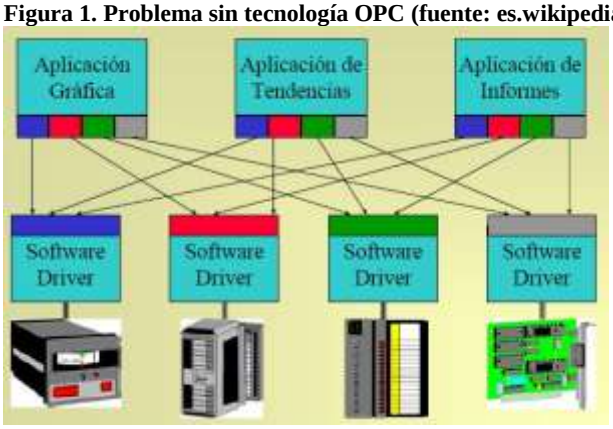


Figura 2. Solución con tecnología OPC (fuente: es.wikipedia.org)

El *driver* todavía es necesario, pero se le oculta el cliente, además el software se desarrolla de forma independiente del hardware subyacente.
OPC tiene tres módulos principales:

- OPC – DA = Data Access: Permite la definición de las variables que determinan el estado de la planta. Sus valores se obtienen de los sensores o de las posiciones de memoria de PLC's. Los valores pueden ser leídos mediante tres modalidades distintas: bajo demanda, cuando cambian o cada cierto tiempo. El servidor gestiona la actualización de los valores de acuerdo a la política escogida. Los principales consumidores de estos datos son las aplicaciones de visualización y control/supervisión.
- OPC – AE = Alarms and Events: Los eventos son cambios que se producen en la planta y que deben ser registrados. Las alarmas son estados anormales de la planta que requieren la atención del usuario. En OPC-AE se especifica cómo los clientes se suscriben a las alarmas y eventos definidos y bajo qué condiciones son filtradas y enviadas por sus correspondientes notificaciones.
- OPC – HDA = Historical Data Access: Contiene la información de históricos de los dispositivos. Su función principal es proporcionar información gráfica.

Con el paso del tiempo comenzaron a surgir algunos inconvenientes con la tecnología de OPC:

- Dependencia de *Windows.*
- Problemas con los *firewalls*.
- Poca alineación con las tecnologías web.
- Modelo orientado al objeto.
- Problemas de seguridad.
- Cada uno de los componentes (A&E, HDA, DA) define un espacio de objetos disjuntos.

Para solventar todos estos problemas se evolucionó al estándar OPC-UA: *unified architecture*. Esta tecnología incluye nuevas funcionalidades y ofrece la posibilidad de funcionar en distintos sistemas operativos, al contrario que OPC que sólo podía funcionar en *Windows*, permitiendo de esta manera portabilidad e interoperabilidad. También se ha mejorado la seguridad, (el OPC tenía la limitación de los *firewalls*), el modelo de datos y además todas las funciones A&E, HDA, DA, juntas en una, para evitar el problema de los objetos disjuntos. Sus principales características son:

- Independencia del fabricante, del lenguaje de programación e incluso de del sistema operativo empleado. Por tanto, se puede integrar el protocolo OPC-UA, en distintos dispositivos.
- Modelo Cliente-Servidor para el intercambio de información entre procesos.
- Mejora de la seguridad frente a OPC, se garantiza la protección contra accesos no autorizados, ofrece opciones para autentificar al usuario y cifrado de los datos a transmitir.
- Busca una transmisión fiable entre los sistemas distribuidos, por lo que se trata de un sistema robusto configurable.
- Se emplea un protocolo binario y optimizado de transmisión por TCP para el intercambio de datos y además soporta los protocolos para servicios web y HTTP.

Dentro del estándar de OPC existen distintas implementaciones algunas de pago y otras gratuitas. Puesto que una de las máximas de la ingeniería es la búsqueda de soluciones a bajo coste, se va a elegir una implementación de código abierto, en este caso será '*open62541*', que está escrita bajo el compilador *C99* y se encuentra en un proceso de actualización constante, por lo que la no tiene el problema de la obsolescencia.

4.4 LENGUAJE DE PROGRAMACIÓN C/C++

C es un lenguaje de alto nivel orientado a la implementación de sistemas operativos. Se trata de un lenguaje de propósito general, que contiene las estructuras típicas de un lenguaje de alto nivel, pero a su vez a su vez dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Sus propiedades principales son:

- Núcleo de lenguaje simple con funcionalidades añadidades importantes.
- Lenguaje estructurado.
- Usa un lenguaje de preprocesado para tareas como definir macros e incluir múltiples archivos de código fuente.

- Conjunto reducido de palabras clave.
- Lenguaje muy eficiente puesto que se diseñó de manera que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas.

C++ es un lenguaje de programación que permite la manipulación de objetos, tiene facilidades de programación genérica como son la programación estructurada y la programación orientada a objetos, es de lenguaje híbrido, considerado de programación multiparadigma. Se trata básicamente de una extensión de C, por lo que un compilador de C++ puede compilar tanto ficheros de C como de C++, incluso en muchos ficheros se mezclan ambos lenguajes en un mismo fichero, como es el caso de este proyecto. Sus características básicas son:

- Sintaxis heredada del lenguaje C.
- Programación orientado a objetos.
- Permite la agrupación de instrucciones.
- Lenguaje didáctico.
- Portable, tiene un gran número de compiladores en distintas plataformas y sistemas operativos.
- Permite la separación de un programa en módulos que admiten compilación independiente.
- Lenguaje de alto nivel.

4.5 LENGUAJE DE PROGRAMACIÓN PYTHON

Python es un lenguaje de programación interpretado cuya máxima principal es la legibilidad del código. Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. Posee una licencia de código abierto. Sus características son:

- Lenguaje interpretado, no compilado.
- Fuertemente tipado.
- Tipado dinámico
- Multiplataforma, puede ejecutarse en distintos sistemas operativos.
- Lenguaje de programación multiparadigma.
- Simplificado y rápido.
- Elegante y flexible.
- Fácil de aprender.
- Ordenado y limpio.

Tiene una amplia aplicación en las áreas de Data Science, Big Data e Inteligencia artificial.

4.6 ECLIPSE

Es una plataforma de software compuesta por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama “Aplicaciones de Cliente Enriquecido”. Esta plataforma ha sido usada típicamente para crear entornos de desarrollo integrados (IDE: Integrated Development Enviroment). Es un editor de texto con analizador sintáctico que

permite la compilación en tiempo real. Es el entorno que se va a usar para desarrollar los fichero de código C/C++.

4.7 VISUAL STUDIO CODE

Es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control de Git, resaltado de sintaxis, finalización de código, fragmentos y refactorización de código. Es gratuito y de código abierto, aunque exista una versión privativa. Es un editor compatible con varios lenguajes de programación y un conjunto de características que pueden estar o no disponibles para un lenguaje dado.

5. DESARROLLO DEL PROYECTO

En este apartado se describe la metodología utilizada para llevar a cabo este proyecto, así como la presentación de los resultados intermedios en las distintas partes del mismo. En el 'Anexo 5. Configuración de los entornos de programación y del software utilizado' se han detallado y especificado los distintos comandos y los resultados de las verificaciones y pruebas que se describen en este apartado.

5.1 CONFIGURACIÓN DEL ENTORNO Y EL ESPACIO DE TRABAJO

Para el desarrollo del proyecto hay que instalar ROS y verificar que la instalación se ha hecho de forma correcta, comprobando que las variables del entorno ROS han sido instaladas.

Es necesario también crear un espacio de trabajo, que será el lugar donde se guarden los programas desarrollados para este proyecto.

La creación del espacio de trabajo hace que en el directorio especificado (en este proyecto *catkin_ws*) se creen las carpetas requeridas. En concreto, se habrán creado tres carpetas:

- *src*: carpeta donde se va a trabajar. Aquí se encuentra el código fuente; es el lugar donde crear, clonar y editar códigos para los distintos paquetes que se quieran construir.
- *build*: en este espacio se invoca el *CMake* para poder compilar los paquetes que se encuentran en src. La información caché y otros archivos intermedios se guardan aquí.
- *devel*: (*development*) aquí se guardan los *built targets*, es decir, los ejecutables de los proyectos desarrollados.

5.2 PRIMERAS PRUEBAS

Las primeras prueban requieren que se arranque un *master* (habilita de manera individual los nodos). Tras esto, el único nodo que hay en marcha en el sistema de ROS es */rosout* que es el proceso necesario para que ROS esté ejecutándose. Una vez ejecutado, se puede abrir otro terminal y ejecutar programas de ROS.

Con el objetivo de verificar el funcionamiento del sistema, la primera prueba de funcionamiento de ROS se ha hecho ejecutando los programas *talker* y *listener*, que son una aplicación de publicador (lanza un mensaje) y suscriptor (escucha un mensaje). Tanto el *talker* y el *listener* son dos nodos distintos, que tienen en común un *topic* (o tópico en español) por el que se están comunicando denominado *chatter*. Estos programas se han obtenido de tutoriales de ROS.

Se han hecho distintas pruebas con estos nodos, con el objetivo de comprobar cómo se lanzan, cómo se puede listar información, cómo se puede publicar un mensaje y cómo se pueden graficar los nodos y *topics* que están funcionando.

5.3 CREACIÓN DE PAQUETES.

Los paquetes contienen el código fuente de cada uno de los proyectos que se hagan. Para el proyecto se ha creado un paquete para el que se han especificado las siguientes dependencias:

- *std_msgs*, indica un tipo de mensajes;
- *rospy*, para poder incluir archivos programados en el lenguaje *phyton*;
- *roscpp* para poder incluir archivos programados en *C/C++.*

La creación de un paquete hace que se creen automáticamente dos carpetas en su interior: una denominada *include*, donde se deben guardar los archivos .h y otra denominada *src*, donde se guardarán los archivos .cpp. También se crea un documento denominado '*CMakeLists.txt'*, que contiene las dependencias que se añadieron en el momento de su creación. Se pueden añadir más dependencias modificando el *CMakeLists*.txt; este documento es esencial para la compilación de los paquetes.

5.4 PROGRAMA EN ROS PARA PUBLICAR UNA VELOCIDAD EN UN *TOPIC* ASOCIADO A UN ROBOT Y LEER SU ODOMETRÍA DE OTRO *TOPIC*.

Para la creación de este programa, se va a utilizar una herramienta de ROS denominada *gazebo*. *Gazebo* es un simulador de entornos 3D con el que se puede evaluar el comportamiento de un robot en el mundo virtual. Permite diseñar robots de forma personalizada, crear mundos virtuales usando herramientas CAD e importar modelos ya creados.

No es necesario instalar ningún paquete nuevo, puesto que este simulador ya viene integrado en el paquete *kinetic,* instalado previamente. El robot que se va a usar también está ya instalado por lo que sólo es necesario lanzarlo. Se va a usar un robot ya creado para realizar estas pruebas, puesto que el objetivo de este proyecto no es la dominación de esta herramienta, sino la de tener los suficientes conocimientos sobre ROS para poder implementar el módulo OPC-UA.

En la figura 3 se muestra el robot utilizado.

Figura 3. Robot husky en gazebo.

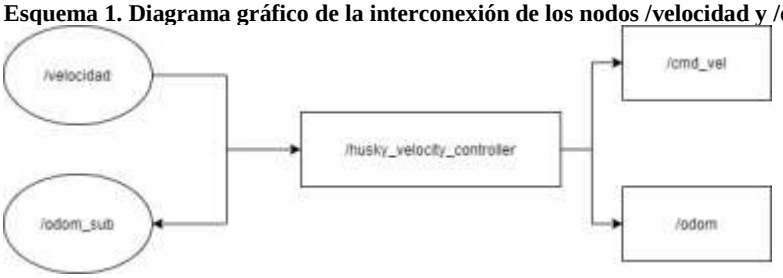
En la figura 4 se listan los *topic*s que hay activos en ROS al poner en marcha el robot *Husky*.

Figura 4. Listado de topic activos en ROS al poner en marcha el robot Husky

```
ester@ester-VirtualBox:~$ rostopic list
/clock
/cmd_vel
/diagnostics
/e_stop
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/husky_velocity_controller/cmd_vel
/husky_velocity_controller/odom
/husky_velocity_controller/parameter_descriptions
/husky_velocity_controller/parameter_updates
/imu/data
/imu/data/accel/parameter_descriptions
/imu/data/accel/parameter_updates
```

El programa que se va a desarrollar publicará una velocidad y leerá la odometría del robot, según el Esquema 1. No se ha usado la herramienta explícita de ROS para crear este esquema porque ésta muestra todos los nodos y *topics* activos del robot, no sólo sobre los que se está trabajando.

Esquema 1. Diagrama gráfico de la interconexión de los nodos /velocidad y /odom_sub.



Para la primera tarea, enviar una velocidad al robot, se ha creado un nodo que publique en */husky_velocity_controller/cmd_vel*. Para la segunda tarea, leer la odometría del robot, se ha creado un nodo que se suscribe a */husky_velocity_controller/odom*.

Para realizar este programa es necesario crear también un paquete, al que se ha llamado *odom_vel*:

```
$catkin_create_pkg odom_vel roscpp geometry_msgs nav_msgs
```

Las dependencias que se han creado en el *CMakeLists*, son *roscpp*, para indicar que se va a programar en *C/C++* y el tipo de mensajes que se van a publicar o se van a recibir. En el caso de la velocidad son de tipo *geometry* y en el caso de la odometría son de tipo *nav*. En el fichero *source,* tipo *cpp,* desde eclipse, se incluyen las librerías necesarias para que el proyecto funcione:

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "nav_msgs/Odometry.h"
#include <cstdlib>
#include <ctime>
#include "std_msgs/String.h"
#include <sstream>
```

Se detallan a continuación la solución propuesta para llevar a cabo las dos tareas descritas.

5.4.1 Publicar velocidad.

Para publicar la velocidad se ha seguido la filosofía del nodo *talker*. Cada segundo se publicará una velocidad en el *topic* /husky_velocity_controller/vel. Primero es necesario declara el nodo e iniciarlo, además de declarar dos variables de tipo double que serán los valores de la velocidad lineal (forwardvel) y la velocidad angular (rotatevel).

```
int main(int argc, char **argv) {
	ros::init(argc, argv, "velocidad");
	ros::Publisher comandoVel;
	double forwardVel;
	double rotateVel;
	ros::NodeHandle nh;
	comandoVel =
nh.advertise<geometry_msgs::Twist>("husky_velocity_controller/cmd_vel", 1);
```

Para especificar la frecuencia con la se publica cada mensaje se usa *rate*. Una vez hecho, además antes de publicar la velocidad es necesario convertir las variables en mensajes de tipo linear en su coordenada. Esta aplicación se ejecutará en segundo plano cada segundo.

Las instrucciones incluidas en el bucle *while* se ejecutarán mientras el motor (*master)* de ROS esté en marcha.

```
ros::Rate rate(1);
while (ros::ok()) {
	geometry_msgs::Twist msg;
	msg.linear.x = forwardVel;
	msg.linear.z = rotateVel;
	comandoVel.publish(msg);
	ros::spinOnce();
	rate.sleep();
}
return 0;
};
```

5.4.2 Escuchar odometría.

Básicamente este programa se suscribirá al nodo de */husky_velocity_controller/odom* y publicará el mensaje leído por la consola. Para ello usará una función denominada *counterCallback*, que se ejecuta cada vez que se escucha un nuevo mensaje.

```
void counterCallback(const nav_msgs::Odometry::ConstPtr& msg)
{
	ROS_INFO_STREAM("Odometry x: " << msg->pose.pose.position.x);
	ROS_INFO_STREAM("Odometry y: " << msg->pose.pose.position.y);
	ROS_INFO_STREAM("Odometry angz: " << 2 * atan2(msg->pose.pose.position.z,
msg->pose.pose.orientation.w));
}


int main(int argc, char **argv) {
	ros::init(argc, argv, "odom_sub");
	ros::NodeHandle nh;
	ros::Subscriber sub = nh.subscribe("husky_velocity_controller/odom", 1000,
counterCallback);//sub= nh.subscriber();
	ros::spin();
	return 0;
}
```

5.4.3 *CMakeLists*.

Para que los programas se puedan compilar sin ningún problema (siempre y cuando el código sea correcto) es necesario indicar en el *CMakeLists* el nombre del ejecutable que se va a añadir, las librerías, en caso de haber librerías externas, que hay que unir con el proyecto principal y añadir las dependencias necesarias. Para este programa, estas líneas son:

```
add_executable(odom_vel  src/ odom_vel.cpp)
target_link_libraries(odom_vel  ${catkin_LIBRARIES})
add_dependencies(odom_vel  odom_vel_generate_messages_cpp)
```

5.4.4 Compilar.

Para compilar se usa:

```
$catkin_make
$source devel/setup.bash
```

Otra forma adicional a la anteriormente detallada para ejecutar un programa es llamarlo desde el directorio en el que se encuentra.

```
$cd ~/catkin_ws/devel/lib/odom_vel
$./odom_vel
```

Todos los resultados que se obtengan durante el desarrollo de este proyecto se encuentran recogidos en el apartado 'RESULTADOS. ANÁLISIS Y DISCUSION DE RESULTADOS' donde se comentan y se comparan entre ellos.

5.5 APLICACIÓN CLIENTE SERVIDOR CON OPC-UA

Para poder implementar un servidor y/o un cliente OPC-UA en un proyecto es necesaria la instalación previa del paquete OPEN62541; en el 'Anexo 5. Configuración de los entornos de programación y del software utilizado' se describe cómo se hace esta instalación. En este anexo también se describe cómo se hace la instalación de un servidor OPC-UA, necesaria para desarrollar aplicaciones cliente servidor con OPC-UA.

5.5.1 Descripción.

Lo primero que se hace es lanzar un servidor. Para ello se puede especificar una dirección ip y un puerto donde escuchar o, bien, se pueden dejar los valores por defecto.
Desde otra terminal se ejecuta un cliente que solicita al servidor la siguiente información:

- Fecha actual.

Accederá a un nodo llamado '*Room1*', donde podrá leer la temperatura del sensor que hay en esa habitación, el nombre del vendedor del sensor y su número se serie. Tanto el número de serie como el nombre del vendedor serán constantes, mientras que la temperatura será variable, es decir, para dos demandas distintas de un cliente (o varios clientes) las lecturas del *tag* de temperatura deben, a priori, ser distintas, aunque también puede darse la casualidad de que sea la misma, pues el cambio será aleatorio y puede haber dos cambios que hagan la temperatura actual y la anterior sean iguales.

5.5.2 Servidor.

La estructura que tendrán los nodos que se van a añadir es la siguiente:

- *Room 1*:
  - Nodo Temperatura (nombre *r1_tempsens_Id*)
    - Temperatura (tipo *UA_Double*)
    - Nombre vendedor (tipo *UA_String*)
    - Número de serie (tipo *UA_Int32*)

Primero para crear el objeto '*room 1'* se usa:

```
//Creación de un objeto en el servidor
UA_Int16 ns_room1 = UA_Server_addNamespace(server, "Room1");
UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "New Namespace added with Nr
%d\n", ns_room1);
```

A este espacio, hay que añadir el sensor de temperatura que será el nodo al que se accede:

```
//Añadir el sensor de temperatura al nodo
UA_NodeId r1_tempsens_Id; /* Obtener el nodeID asignado por el servidor */
UA_ObjectAttributes oAttr = UA_ObjectAttributes_default;

UA_Server_addObjectNode(server, UA_NODEID_NULL,
       UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
       UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
       UA_QUALIFIEDNAME(2, "Temperature Sensor"), UA_NODEID_NULL,
       oAttr, NULL, &r1_tempsens_Id);
```

Los datos pertenecientes al sensor de temperatura se agregan de la siguiente forma:

```
//Añadir nombre del vendedor del sensor
UA_VariableAttributes vnAttr = UA_VariableAttributes_default;
UA_String vendorName = UA_STRING("Sensor temperature King Ltd.");
UA_Variant_setScalar(&vnAttr.value, &vendorName, &UA_TYPES[UA_TYPES_STRING]);
// vnAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ManufacturerName");
UA_Server_addVariableNode(server,   UA_NODEID_STRING(2,   "R1_TS1_VendorName"),
r1_tempsens_Id,
       UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
       UA_QUALIFIEDNAME(2, "VendorName"),
       UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), vnAttr, NULL, NULL);


//Añadir número de serie del sensor
UA_VariableAttributes snAttr = UA_VariableAttributes_default;
UA_Int32 serialNumber = 12324342;
UA_Variant_setScalar(&snAttr.value, &serialNumber, &UA_TYPES[UA_TYPES_INT32]);
UA_Server_addVariableNode(server,  UA_NODEID_STRING(2,  "R1_TS1_SerialNumber"),
r1_tempsens_Id,
       UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
       UA_QUALIFIEDNAME(2, "SerialNumber"),
       UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), snAttr, NULL, NULL);
```

```
//Añadir la variable temperatura
UA_VariableAttributes tpnAttr = UA_VariableAttributes_default;
Temperature = 12; //Es variable global para poder cambiar su valor.
UA_Variant_setScalar(&tpnAttr.value, &Temperature, &UA_TYPES[UA_TYPES_DOUBLE]);
UA_Server_addVariableNode(server,  UA_NODEID_STRING(2,  "R1_TS1_Temperature"),
r1_tempsens_Id,
       UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
       UA_QUALIFIEDNAME(2, "Temperature"),
       UA_NODEID_NUMERIC(0,  UA_NS0ID_BASEDATAVARIABLETYPE),  tpnAttr,  NULL,
NULL);
```

La manera de acceder a cada nodo es usando *UA_NODEID_STRING(2, "R1_TS1_SerialNumber");* es necesario conocer esto para poder desarrollar la parte cliente de manera adecuada.

Para poder actualizar la variable de temperatura, al no estar conectada a ningún sensor real, se hace mediante software. Para ello se usa una función que se ejecuta justo antes de leer el sensor, de manera que antes de cada lectura estará actualizado su valor.

```
static void beforeReadTemperature(UA_Server *server,
       const UA_NodeId *sessionId, void *sessionContext,
       const UA_NodeId *nodeid, void *nodeContext,
       const UA_NumericRange *range, const UA_DataValue *data) {
       srand48(time(NULL));
       double random;
       random = 1.0 *(rand() % 100) / 100 - 0.5;;
       Temperature += random;

       //Actualizar variable

       UA_Variant value;
       UA_Variant_setScalar(&value, &Temperature, &UA_TYPES[UA_TYPES_DOUBLE]);
       UA_Server_writeValue(server,  UA_NODEID_STRING(2,  "R1_TS1_Temperature"),
value);
```

Básicamente, lo que se ha hecho, ha sido, crear un número aleatorio (con la función *random*) que produzca pequeñas variaciones en la temperatura actual. Siempre que se vaya a leer la variable temperatura para mandar su valor a un cliente se ejecutará antes esta función, por lo que siempre tendrá un valor diferente al anterior.

Por último, es necesario añadir el código de una función llamada *callback* para poder actualizar el valor de la variable. Lo que se hace es que cuando hay una petición de un cliente se actualizan los valores y no antes gracias a la ejecución de la función *callback*.

```
//Añadir callback a la variable temperatura
UA_ValueCallback callback;
callback.onRead = beforeReadTemperature;
callback.onWrite = NULL;
UA_Server_setVariableNode_valueCallback(server, UA_NODEID_STRING(2,
"R1_TS1_Temperature"), callback);
}
```

Existe también otra función similar a '*BeforeRead*' que se llama *'AfterRead'* se usa por si quieren llevar cabo acciones una vez que se ha leído la variable, se puede usar con la siguiente línea de código:

```
callback.onWrite = AfterRead;
```

Se puede añadir una última parte al código para especificarle dirección IP y puerto al servidor, en lugar de que se conecte siempre al mismo valor:

```
//Comprobar número de argumentos
if (argc > 2) {
	UA_Int16 port_number = atoi(argv[2]);
	UA_ServerConfig_setMinimal(UA_Server_getConfig(server), port_number, 0);

}
else
UA_ServerConfig_setDefault(UA_Server_getConfig(server));
if (argc > 1) {
	//Comprobar si están disponibles IP o puerto
	//Copiar hostname desde char * a una variable open62541
	UA_String hostname;
	UA_String_init(&hostname);
	hostname.length = strlen(argv[1]);
	hostname.data = (UA_Byte *)argv[1];

	//Cambiar configuración
	UA_ServerConfig_setCustomHostname(UA_Server_getConfig(server),
hostname);
}
```

5.5.3 Cliente.

Al igual que en la parte servidor, se va a partir de un código base que se irá modificando para añadir los distintos *tags*. El código base es:

```c
#include <open62541/client_config_default.h>
#include <open62541/client_highlevel.h>
#include <open62541/plugin/log_stdout.h>

#include <stdlib.h>

int main(void) {
    UA_Client *client = UA_Client_new();
    UA_ClientConfig_setDefault(UA_Client_getConfig(client));
    UA_StatusCode retval = UA_Client_connect(client, "opc.tcp://localhost:4840");
    if(retval != UA_STATUSCODE_GOOD) {
        UA_Client_delete(client);
        return (int)retval;
    }

    //Leer el valor atributo del nodo.
    UA_Variant value; //Las variables pueden ser arrays o escalares
    UA_Variant_init(&value);

    //NodeID de la variable de tiempo
   const UA_NodeId nodeId = UA_NODEID_NUMERIC(0,
UA_NS0ID_SERVER_SERVERSTATUS_CURRENTTIME);
    retval = UA_Client_readValueAttribute(client, nodeId, &value);

    if(retval == UA_STATUSCODE_GOOD &&
       UA_Variant_hasScalarType(&value, &UA_TYPES[UA_TYPES_DATETIME])) {
        UA_DateTime raw_date = *(UA_DateTime *) value.data;
        UA_DateTimeStruct dts = UA_DateTime_toStruct(raw_date);
        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "date is: %u-%u-%u
%u:%u:%u.%03u\n",
                    dts.day, dts.month, dts.year, dts.hour, dts.min, dts.sec,
dts.milliSec);
    }

    //Limpiar
    UA_Variant_clear(&value);
    UA_Client_delete(client); //Desconectar cliente internamente

    return EXIT_SUCCESS;
}
```

A este código es necesario añadirle los *tags* que se van leer, en principio sólo se lee la fecha y la hora. Inicialmente se añaden los *tags* siguiendo la misma filosofía que cuando se lee el *tag* de la fecha y hora. Primero se declaran las variables que se van a leer.

```c
//Variables a las que se accede
UA_String VendorName;
UA_Int32 SerialNumber;
UA_Double Temperature;
```

Para cada lectura se sigue siempre la misma estructura:

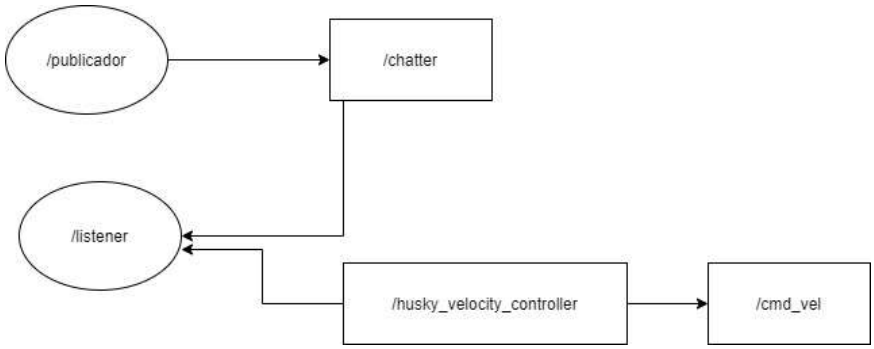
```
//Leer el nombre del fabricante

retval      =      UA_Client_readValueAttribute(client,      UA_NODEID_STRING(2,
"R1_TS1_VendorName"), &value);

//Comprobar si ha habido éxito
if    (retval    ==    UA_STATUSCODE_GOOD    &&    UA_Variant_hasScalarType(&value,
&UA_TYPES[UA_TYPES_STRING])) {
       VendorName = *(UA_String *)value.data;
       UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "The Vendor Name is:
%.*s\n", VendorName.length, VendorName.data);
}

//Leer número de serie

retval      =      UA_Client_readValueAttribute(client,      UA_NODEID_STRING(2,
"R1_TS1_SerialNumber"), &value);

// Comprobar si ha habido éxito
if    (retval    ==    UA_STATUSCODE_GOOD    &&    UA_Variant_hasScalarType(&value,
&UA_TYPES[UA_TYPES_INT32])) {
       SerialNumber = *(UA_Int32 *)value.data;
       UA_LOG_INFO(UA_Log_Stdout,  UA_LOGCATEGORY_USERLAND,  "The  Serial  number
is: %d\n", SerialNumber);
}
//Leer temperatura

retval      =      UA_Client_readValueAttribute(client,      UA_NODEID_STRING(2,
"R1_TS1_Temperature"), &value);

// Comprobar si ha habido éxito
if    (retval    ==    UA_STATUSCODE_GOOD    &&    UA_Variant_hasScalarType(&value,
&UA_TYPES[UA_TYPES_DOUBLE])) {
       Temperature = *(UA_Double *)value.data;
       UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "The Temperature is:
%f degrees\n", Temperature);
}
```

5.6 SERVIDOR Y CLIENTE OPC-UA EN MÓDULO ROS

Una vez creado y tras un desarrollo inicial del proyecto ROS, en la siguiente fase, se trabajará sobre este proyecto para integrar en el mismo el desarrollo que se acaba de realizar de aplicación cliente servidor. La comunicación entre los topics de ROS y el servidor y cliente OPC-UA se puede realizar gracias a las librerías de open62541. Para que los topics se lean de ROS sólo habrá que seguir los pasos descritos previamente. El flujo de datos viene definido por una aplicación en ROS que lee los datos de un robot, y posteriormente usando el protocolo OPC-UA crea un servidor y que permite acceder a los datos que se van leyendo de ROS a la vez que se van refrescando los datos del servidor conforme se van leyendo de ROS.

Por tanto, a partir de las dos aplicaciones anteriores se ha desarrollado una aplicación cliente servidor que accede a los datos de odometría del robot *husky*, además de al *topic /chatter*, donde otro programa denominado publicador está publicando mensajes de tipo *string*. Se ha decidido usar dos *topics* distintos para acceder a variables de distinto tipo y mostrar cómo un único servidor puede ofrecer datos provenientes de *topics* diferentes.
En él se muestra la interconexión de los nodos y los *topics*:

Esquema 2. Diagrama gráfico de la interconexión de los nodos */publicador* y /listener con sus respectivos topics



El programa publicador, que contiene un código muy sencillo y similar en estructura al del nodo *'talker'*, es el siguiente:

```
int main(int argc, char **argv) {
	ros::init(argc, argv, "publicador");
	ros::NodeHandle n;
	ros::Publisher pub = n.advertise<std_msgs::String>("chatter", 1000);
	ros::Rate loop_rate(100);
	while (ros::ok())
	{
		std_msgs::String msg;
		msg.data = "Prueba TFM";
		pub.publish(msg);
		ros::spinOnce();
		loop_rate.sleep();
	}
	ros::spin();
	return 0;
}
```

Este nodo, estará continuamente publicando mientras el motor de ROS esté en marcha o no se aborte el proceso.

5.6.1 Parte Servidor.

El programa que se ha desarrollado tiene dos hilos de ejcución, uno para suscribirse a los nodos y otro que se encarga de realizar las tareas del servidor.

Es necesario suscribirse a los distintos nodos que hay y actualizar las variables. Estas variables son variables globales y de acceso compartido, al tener el programa tendrá dos hilos de ejecución. Para evitar problemas derivados del acceso compartido, como por ejemplo la condición de carrera, la propuesta es usar mutex que es un protocolo de acceso a variables compartidas. Las variables que contienen los datos de los topics son:

- 'x', odometría en x, de tipo *UA_Double*.
- 'y', odometría en y, de tipo *UA_Double*.
- 'z', odometría en z, de tipo *UA_Double*.
- 'mensaje', mensaje del nodo publicador, de tipo *UA_String*.

Se declararán esas variables globales, el *mutex* asociado a cada una de ellas, un *struct* - para guardar los argumentos del *main*, y el booleno *running* para controlar el encendido/apagado del servidor:

```
struct dato_ini {
	int argc;
	char** argv;
};
pthread_mutex_t mutex_mensaje = PTHREAD_MUTEX_INITIALIZER; //Declaración del
mutex para evitar el acceso simúltaneo a varibales compartidadas
pthread_mutex_t mutex_odometria_x = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_odometria_y = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_odometria_z = PTHREAD_MUTEX_INITIALIZER;
//Declaración de funciones que están debajo del main
static void beforeReadOdom_x(UA_Server *server,
	const UA_NodeId *sessionId, void *sessionContext,
	const UA_NodeId *nodeid, void *nodeContext,
	const UA_NumericRange *range, const UA_DataValue *data);
static void beforeReadOdom_y(UA_Server *server,
	const UA_NodeId *sessionId, void *sessionContext,
	const UA_NodeId *nodeid, void *nodeContext,
	const UA_NumericRange *range, const UA_DataValue *data);
static void beforeReadOdom_z(UA_Server *server,
	const UA_NodeId *sessionId, void *sessionContext,
	const UA_NodeId *nodeid, void *nodeContext,
	const UA_NumericRange *range, const UA_DataValue *data);


static void beforeReadMensaje(UA_Server *server,
	const UA_NodeId *sessionId, void *sessionContext,
	const UA_NodeId *nodeid, void *nodeContext,
	const UA_NumericRange *range, const UA_DataValue *data);

static volatile UA_Boolean running = true;
UA_Double x;
UA_Double y;
UA_Double z;
UA_String mensaje;
```

En el *main* se realiza la inicialización de las variables, y se lanza el hilo que se ocupará de suscribirse a los nodos de ROS.

```
int main(int argc, char *argv[]) {
	signal(SIGINT, stopHandler);
	signal(SIGTERM, stopHandler);

	pthread_t hilo_ros;
	pthread_attr_t attr;
	pthread_attr_init(&attr);
	dato_ini inicializacion;
	inicializacion.argc = argc;
	inicializacion.argv = argv;
	pthread_create(&hilo_ros, &attr, nodos_ROS, &inicializacion); //hilo
para lanzar los suspcriptres de ros
	UA_Server *server = UA_Server_new();
	//Inicialización de variables
	mensaje = UA_STRING("Sin informacion.");
	x = 0;
	y = 0;
	z = 0;
```

Al igual que se hizo en la aplicación cliente servidor anterior, se puede introducir el código para permitir cambiar la dirección IP y el puerto del servidor mediante los argumentos del *main*.

La función que ejecuta el hilo encargado de suscribirse a los *topics* incluye la declaración de nodos, y la llamada a sus funciones de *callback* cada vez que se produce un cambio de valor en el topic al que se están suscritos:

```
	void *nodos_ROS(void *param) {
		dato_ini ini = *(dato_ini*)param;
		ros::init(ini.argc, ini.argv, "listener");
		ros::NodeHandle nh;
		ros::Subscriber sub_mensaje = nh.subscribe("chatter", 1000,
chatterCallback);
		ros::init(ini.argc, ini.argv, "leer_odometria");
		//ros::NodeHandle n_odometria;
		ros::Subscriber sub_odometria =
nh.subscribe("husky_velocity_controller/odom", 1000, OdometriaCallback);
		ros::spin();
	}
```

```
	//Callback del nodo listener
	void chatterCallback(const std_msgs::String::ConstPtr& msg)
	{

		ROS_INFO("I heard: [%s]", msg->data.c_str());


		/* String */
		UA_String s;
		UA_String_init(&s); /* _init zeroes out the entire memory of the
datatype */
		char *test = (char *)msg->data.c_str();
		s.length = strlen(test);
		s.data = (UA_Byte*)test;
		pthread_mutex_lock(&mutex_mensaje);
		mensaje = s;
		pthread_mutex_unlock(&mutex_mensaje);



	}
	//callback del nodo leer odometría
	void OdometriaCallback(const nav_msgs::Odometry::ConstPtr& msg)
	{
		ROS_INFO_STREAM("Odometry x: " << msg->pose.pose.position.x);
		ROS_INFO_STREAM("Odometry y: " << msg->pose.pose.position.y);
		ROS_INFO_STREAM("Odometry angz: " << 2 * atan2(msg-
>pose.pose.position.z, msg->pose.pose.orientation.w));
		double aux = (double)msg->pose.pose.position.x;
		pthread_mutex_lock(&mutex_odometria_x);
		x = (UA_Double)aux;
		pthread_mutex_unlock(&mutex_odometria_x);
		aux = (double)msg->pose.pose.position.y;
		pthread_mutex_lock(&mutex_odometria_y);
		y = (UA_Double)aux;
		pthread_mutex_unlock(&mutex_odometria_y);
		pthread_mutex_lock(&mutex_odometria_z);
		z = (UA_Double)(double)2 * atan2(msg->pose.pose.position.z, msg-
>pose.pose.orientation.w);
		pthread_mutex_unlock(&mutex_odometria_z);

	}
```

Y sus respectivas funciones de *callback:*

Además se utiliza el manejador de señal para controlar el encendido y apagado del servidor:

```
static void stopHandler(int sig) {
	UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "received ctrl-c");
		running = false;
	}
```

Para este programa, al espacio que se ha creado se ha denominado *ROS,* el objeto *ROS_robot. S*e han configurado también las variables pertenecientes a este objeto.

```
	//Añadimos el espacio ROS
	UA_Int16 ns_ros = UA_Server_addNamespace(server, "ROS");
	UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "New Namespace added
with Nr %d\n", ns_ros);

	//Añadir objeto llamado ROS_robot
	UA_NodeId ROS_robot;
	UA_ObjectAttributes oAttr = UA_ObjectAttributes_default;

	UA_Server_addObjectNode(server, UA_NODEID_NULL,
		UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
		UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
		UA_QUALIFIEDNAME(2, "ROS_Robot"), UA_NODEID_NULL,
		oAttr, NULL, &ROS_robot);

	//Añadir la variable 'mensaje' al server
	UA_VariableAttributes mensajenAttr = UA_VariableAttributes_default;

	UA_Variant_setScalar(&mensajenAttr.value, &mensaje,
&UA_TYPES[UA_TYPES_STRING]);
	// vnAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ManufacturerName");
	UA_Server_addVariableNode(server, UA_NODEID_STRING(2, "Robot_mensaje"),
ROS_robot,
		UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
		UA_QUALIFIEDNAME(2, "Mensaje"),
		UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
mensajenAttr, NULL, NULL);

	//añadir la odometría en x al servidor
	UA_VariableAttributes x_nAttr = UA_VariableAttributes_default;

	UA_Variant_setScalar(&x_nAttr.value, &x, &UA_TYPES[UA_TYPES_DOUBLE]);
	UA_Server_addVariableNode(server, UA_NODEID_STRING(2, "odometria_x"),
ROS_robot,
		UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
		UA_QUALIFIEDNAME(2, "Odometria_x"),
		UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), x_nAttr,
NULL, NULL);
```

```
       // añadir la odometría en y al servidor
       UA_VariableAttributes y_nAttr = UA_VariableAttributes_default;

       UA_Variant_setScalar(&y_nAttr.value, &y, &UA_TYPES[UA_TYPES_DOUBLE]);
       UA_Server_addVariableNode(server, UA_NODEID_STRING(2, "odometria_y"),
ROS_robot,
              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
              UA_QUALIFIEDNAME(2, "Odometria_y"),
              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), y_nAttr,
NULL, NULL);
       // añadir la odometría en z al servidor
       UA_VariableAttributes z_nAttr = UA_VariableAttributes_default;

       UA_Variant_setScalar(&z_nAttr.value, &z, &UA_TYPES[UA_TYPES_DOUBLE]);
       UA_Server_addVariableNode(server, UA_NODEID_STRING(2, "odometria_z"),
ROS_robot,
              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
              UA_QUALIFIEDNAME(2, "Odometria_z"),
              UA_NODEID_NUMERIC(0,   UA_NS0ID_BASEDATAVARIABLETYPE),   z_nAttr,
NULL, NULL);
```

También se ha añadido una función callback para que antes de que un cliente intente acceder al valor de estas variables se actualice su valor.

```
       //Añadir función callback a Mensaje - Para actualizar su valor:
       UA_ValueCallback callback_mensaje;
       callback_mensaje.onRead = beforeReadMensaje;
       callback_mensaje.onWrite = NULL;
       UA_Server_setVariableNode_valueCallback(server, UA_NODEID_STRING(2,
"Robot_mensaje"), callback_mensaje);

       // Añadir función callback a odometria x - Para actualizar su valor:
       UA_ValueCallback callback_x;
       callback_x.onRead = beforeReadOdom_x;
       callback_x.onWrite = NULL;
       UA_Server_setVariableNode_valueCallback(server, UA_NODEID_STRING(2,
"odometria_x"), callback_x);

       // Añadir función callback a odometria y - Para actualizar su valor:
       UA_ValueCallback callback_y;
       callback_y.onRead = beforeReadOdom_y;
       callback_y.onWrite = NULL;
       UA_Server_setVariableNode_valueCallback(server, UA_NODEID_STRING(2,
"odometria_y"), callback_y);

       // Añadir función callback a odometria z- Para actualizar su valor:
       UA_ValueCallback callback_z;
       callback_z.onRead = beforeReadOdom_z;
       callback_z.onWrite = NULL;
       UA_Server_setVariableNode_valueCallback(server,      UA_NODEID_STRING(2,
"odometria_z"), callback_z);
```

La última parte de la función *main* es el lanzamiento y el borrado del servidor de acuerdo al valor de la variable *'running'*:

```
	UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Starting new
Server...");
	UA_StatusCode retval = UA_Server_run(server, &running);
	UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Server was shut
down");


	UA_Server_delete(server);
	return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

Se añaden las funciones *'beforeRead*…' en las que se actualiza el valor de las variables del servidor, con los valores que obtenidos de la suscripción a los *topics* de *'chatter'* y *'husky_velocity_controller/odom'*.

```
	static void beforeReadMensaje(UA_Server *server,
		const UA_NodeId *sessionId, void *sessionContext,
		const UA_NodeId *nodeid, void *nodeContext,
		const UA_NumericRange *range, const UA_DataValue *data)
	{
		//Actualizar variable
		UA_Variant value;
		pthread_mutex_lock(&mutex_mensaje);
		UA_Variant_setScalar(&value, &mensaje,
&UA_TYPES[UA_TYPES_STRING]);
		pthread_mutex_unlock(&mutex_mensaje);

		UA_Server_writeValue(server, UA_NODEID_STRING(2,
"Robot_mensaje"), value);
	}

	static void beforeReadOdom_x(UA_Server *server,
		const UA_NodeId *sessionId, void *sessionContext,
		const UA_NodeId *nodeid, void *nodeContext,
		const UA_NumericRange *range, const UA_DataValue *data) {


		UA_Variant value;
		pthread_mutex_lock(&mutex_odometria_x);
		UA_Variant_setScalar(&value, &x, &UA_TYPES[UA_TYPES_DOUBLE]);
		pthread_mutex_unlock(&mutex_odometria_x);
		UA_Server_writeValue(server, UA_NODEID_STRING(2, "odometria_x"),
value);
}
```

```
	static void beforeReadOdom_y(UA_Server *server,
		const UA_NodeId *sessionId, void *sessionContext,
		const UA_NodeId *nodeid, void *nodeContext,
		const UA_NumericRange *range, const UA_DataValue *data) {

		UA_Variant value;
		pthread_mutex_lock(&mutex_odometria_y);
		UA_Variant_setScalar(&value, &y, &UA_TYPES[UA_TYPES_DOUBLE]);
		pthread_mutex_unlock(&mutex_odometria_y);
		UA_Server_writeValue(server, UA_NODEID_STRING(2, "odometria_y"),
value);


	}
	static void beforeReadOdom_z(UA_Server *server,
		const UA_NodeId *sessionId, void *sessionContext,
		const UA_NodeId *nodeid, void *nodeContext,
		const UA_NumericRange *range, const UA_DataValue *data) {

		UA_Variant value;
		pthread_mutex_lock(&mutex_odometria_z);
		UA_Variant_setScalar(&value, &z, &UA_TYPES[UA_TYPES_DOUBLE]);
		pthread_mutex_unlock(&mutex_odometria_z);
		UA_Server_writeValue(server, UA_NODEID_STRING(2, "odometria_z"),
value);


	}
```

5.6.2 Parte cliente.

El cliente demanda información de los nodos que se detallan:

```
int main(int argc, char *argv[]) {
	UA_Client *client = UA_Client_new();
	UA_ClientConfig_setDefault(UA_Client_getConfig(client));
	UA_StatusCode retval = UA_Client_connect(client,
"opc.tcp://localhost:4840"); //Puerto donde vamos a publicar

	if (retval != UA_STATUSCODE_GOOD) {
		UA_Client_delete(client);
		return (int)retval;
	}
	UA_Variant value;
	UA_Variant_init(&value);



	//Vaiables para leer un dato tipo string y la odometría que se guardará
en variables tipo double
	UA_String mensaje;

	UA_Double x;
	UA_Double y;
	UA_Double z;

	//Leer Mensaje

	retval = UA_Client_readValueAttribute(client, UA_NODEID_STRING(2,
"Robot_mensaje"), &value);

	//Comprobar si ha habido éxito
	if (retval == UA_STATUSCODE_GOOD && UA_Variant_hasScalarType(&value,
&UA_TYPES[UA_TYPES_STRING])) {
		mensaje = *(UA_String *)value.data;

		UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "El mensaje
leido es: %.*s\n", mensaje.length, mensaje.data);
	}

	//Leer Odometría

	retval = UA_Client_readValueAttribute(client, UA_NODEID_STRING(2,
"odometria_x"), &value);
	//Comprobar si ha habido éxito
	if (retval == UA_STATUSCODE_GOOD && UA_Variant_hasScalarType(&value,
&UA_TYPES[UA_TYPES_DOUBLE])) {
		x = *(UA_Double *)value.data;

		ROS_INFO_STREAM("Odometry x: " << x); //Se saca la información
con ROS_INFO por que ofrece más decimales
	}
```

```
	retval = UA_Client_readValueAttribute(client, UA_NODEID_STRING(2,
"odometria_y"), &value);
	//Comprobar si ha habido éxito
	if (retval == UA_STATUSCODE_GOOD && UA_Variant_hasScalarType(&value,
&UA_TYPES[UA_TYPES_DOUBLE])) {
		y = *(UA_Double *)value.data;

		ROS_INFO_STREAM("Odometry y: " << y);
	}


	retval = UA_Client_readValueAttribute(client, UA_NODEID_STRING(2,
"odometria_z"), &value);
	//Comprobar si ha habido éxito
	if (retval == UA_STATUSCODE_GOOD && UA_Variant_hasScalarType(&value,
&UA_TYPES[UA_TYPES_DOUBLE])) {
		z = *(UA_Double *)value.data;

		ROS_INFO_STREAM("Odometry z: " << z);
	}



	//Limpiar
	UA_Variant_clear(&value);
	UA_Client_delete(client); /* Desconectar cliente internamente */
	return EXIT_SUCCESS;
}
```

5.6.3 Compilación.

En el fichero *CMakeLists.txt*, previamente a la compilación, se han añadido las líneas de código correspondientes a los archivos ejecutables y añadir las librerías externas *pthread* y *open62541* en aquellos archivos que lo requieran:

```
include_directories(include/ros_opcua)
#la librería esta en la carpeta include

 add_library(open62541 src/open62541.c) #linkar con librería de OPC-UA
 add_executable(client src/client.cpp)
 target_link_libraries(client ${catkin_LIBRARIES} open62541)

 add_dependencies(client client_generate_messages_cpp)

 add_executable(server src/server.cpp)
 target_link_libraries(server ${catkin_LIBRARIES} open62541 pthread)
 add_dependencies(server server_generate_messages_cpp)


 add_executable(publicador src/publicador.cpp)
 target_link_libraries(publicador ${catkin_LIBRARIES})
 add_dependencies(publicador publicador_generate_messages_cpp)
```

5.6.4 Pruebas de funcionamiento.

Para las pruebas de funcionamiento, se ha lanzado un programa previamente implementado para que cambie la velocidad del robot a un valor distinto de cero, y así poder observar cambios en la posición del robot y, por tanto, cambios significativos en su odometría.

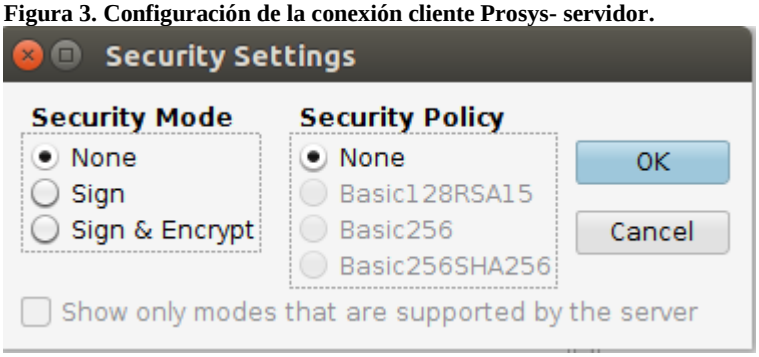
5.7 ACCESO AL SERVIDOR DESDE EL CLIENTE OPC DE PROSYS.

Un cliente OPC *Prosys* es un software que ayuda a solucionar problemas de conexiones OPC y probar servidores OPC.
Para la conexión es necesario abrir la aplicación (una vez instalado el cliente *Prosys*) e introducir el *endpoint*. Previamente es necesario cambiar la configuración del servidor para que envíe una marca de tiempo en el encabezado de la solicitud. Para cambiar la configuración se crea una variable '*config*' que hace referencia la configuración del servidor, y se accede directamente al campo *'verifyrRequestTimestamp'*. Para ello se añaden las líneas de código:

```
//Variable config para cambiar el verifyTimeRequest y poder conectarse con
//client Prosys
UA_ServerConfig* config = UA_Server_getConfig(server);
config->verifyRequestTimestamp = UA_RULEHANDLING_ACCEPT;
```

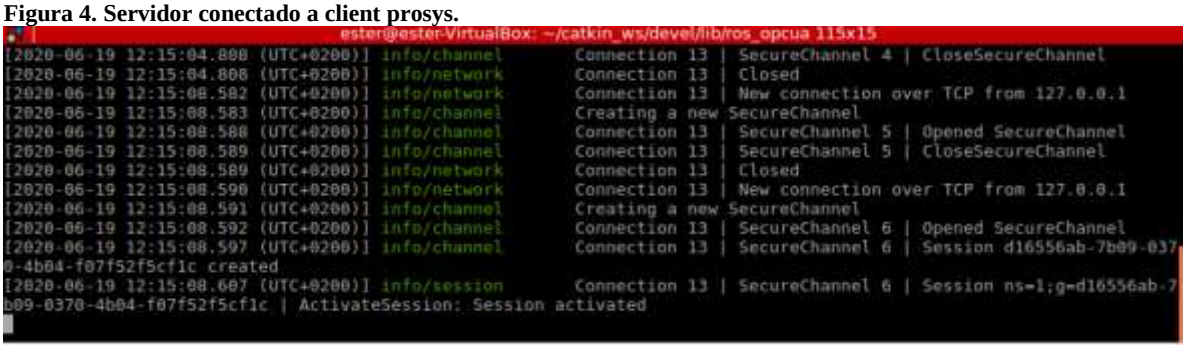
En la configuración del cliente *Prosys*, es necesario especificar que tanto la política seguridad como el modo son *'none'*, puesto que no se ha establecido en el código del servidor ninguna de las dos.

Figura 3. Configuración de la conexión cliente Prosys- servidor.



Una vez realizada esta conexión se obtiene en el terminal donde se ejecuta el servidor:

Figura 4. Servidor conectado a client prosys.



Mientras que en el cliente *prosys* se puede acceder a los distintos nodos del servidor y observar cómo cambia su valor, en función de si el '*publicador*' está enviado mensajes o no, o en qué momento se refresca el cliente.

Figura 5. Cliente Prosys conectado con servidor I.

Figura 6. Cliente Prosys conectado con servidor II.

Figura 7. Cliente Prosys conectado con servidor III.

Figura 8. Cliente Prosys conectado con servidor IV.

5.8 COMUNICACIÓN ENTRE CLIENTE PROSYS EN WINDOWS Y SERVIDOR EN MÁQUINA VIRTUAL LINUX.

Para poder acceder al servidor que se encuentra en la máquina virtual *Linux*, desde una máquina virtual *Windows* o desde el sistema nativo, es necesario reconfigurar la máquina virtual para que pueda acceder a la red local. Para ello se accede a la configuración de la máquina en el apartado de 'Red' y se selecciona la opción de 'Adaptador puente' en el campo 'conectado' y en nombre una *red wi-fi* o *'wireless'*, la configuración queda de la siguiente manera:

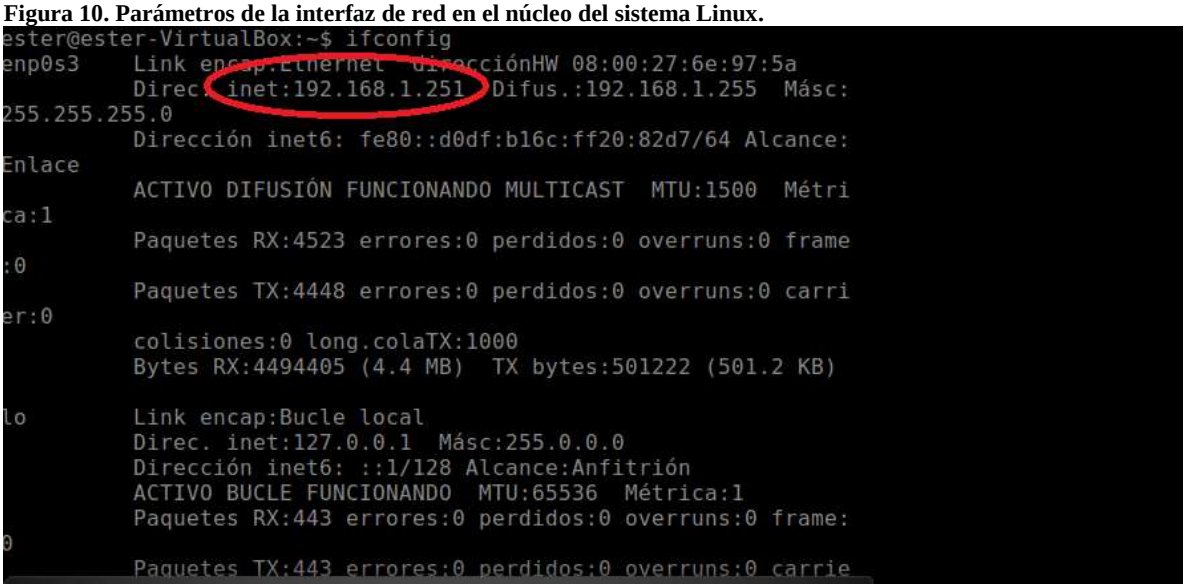
Figura 9. Configuración de red de la máquina virtual.

De esta manera, abriendo una terminal y usando el comando:

```
$ifconfig
```

Se muestra la configuración de red, y la dirección IP que se le ha asignado a la máquina virtual:

Figura 10. Parámetros de la interfaz de red en el núcleo del sistema Linux.



El servidor en lugar de ponerlo a escuchar en la dirección del host local, se pondrá en la misma IP que la máquina virtual, puesto que esta IP es accesible por todos los dispositivos que se encuentren en la red local. Como previamente se diseñó el código de manera que se pudieran cambiar la dirección IP y el puerto en el momento de ejecutar el programa, simplemente es necesario teclear:

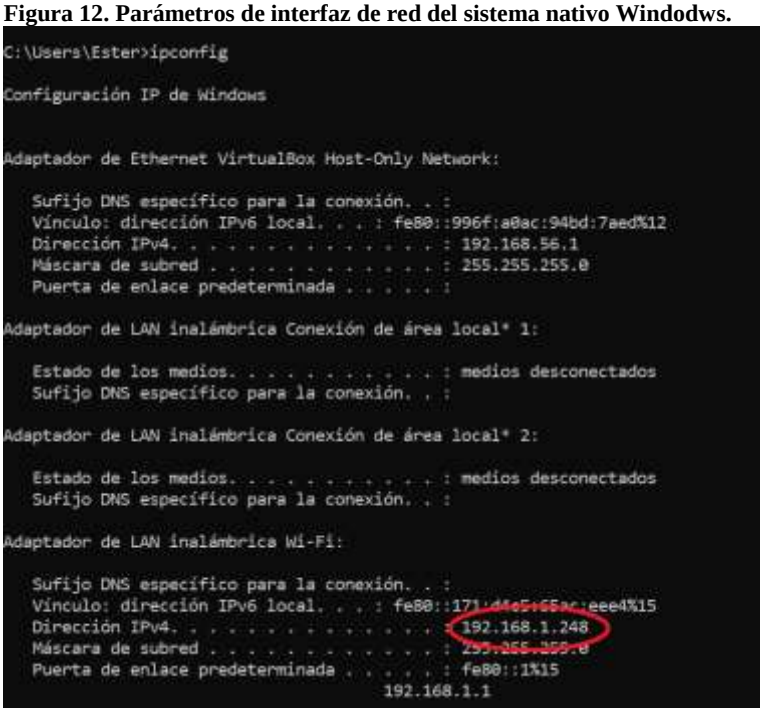
```
$./server 192.168.1.251 4048
```

Figura 11. Servidor ejecutándose en la dirección IP 192.168.1.251 y el puerto 4048

Se observa que el servidor ya no escucha en el *host local*, sino que ahora está en la dirección y puerto que se ha especificado.

En la Figura 12 se muestra la dirección IP del sistema nativo (sistema Windows):

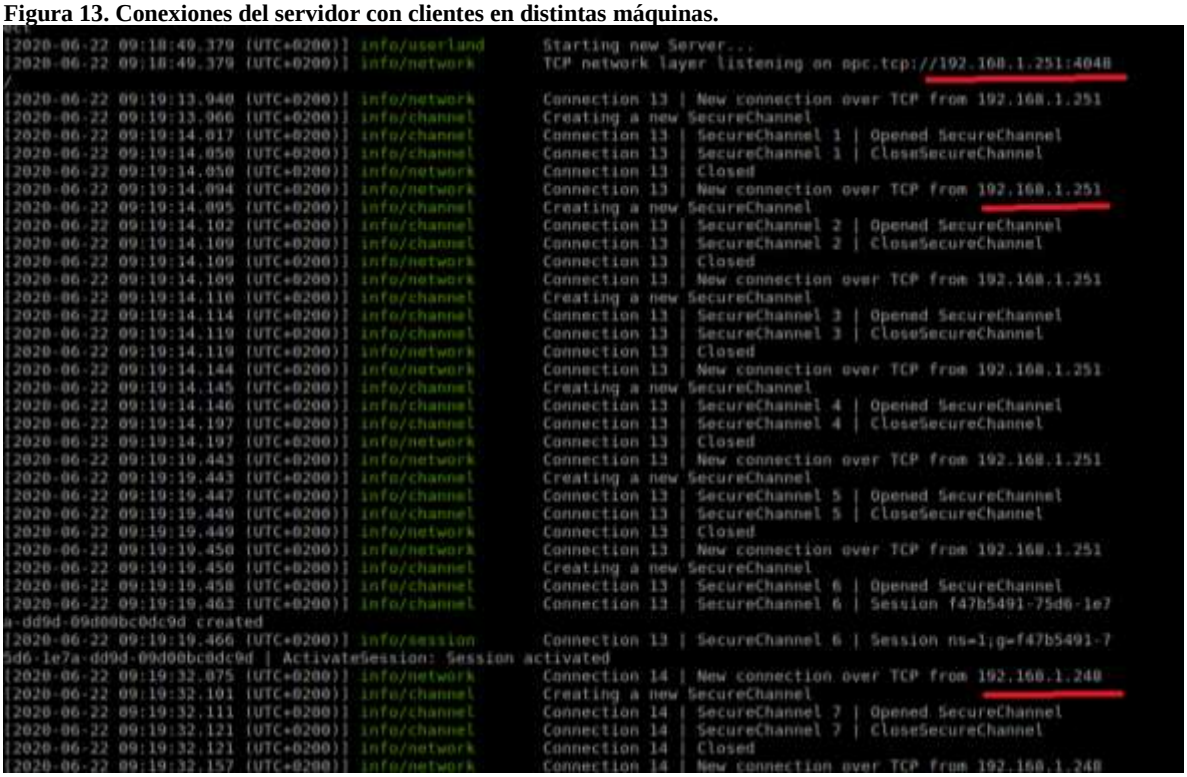
Figura 12. Parámetros de interfaz de red del sistema nativo Windodws.



A continuación, se va a conectar con el cliente de *Prosys* desde *Linux* y con el cliente de Prosys desde Windows.

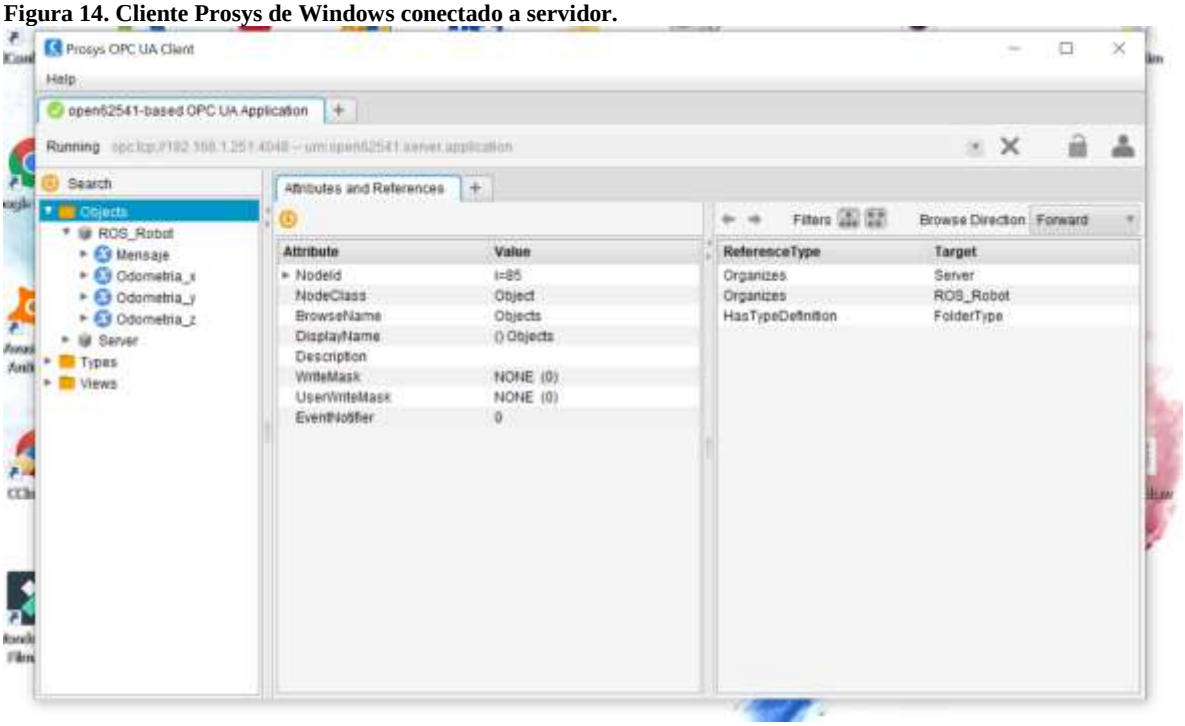
Cuando se realizan las conexiones con los clientes se muestra que cada cliente proviene de una IP distinta; en el caso de Linux tendrá la IP 192.168.1.251, que es la de la máquina virtual, y en el caso de Windows la dirección IP que se acaba de mostrar, es decir, 192.168.1.248.

Figura 13. Conexiones del servidor con clientes en distintas máquinas.



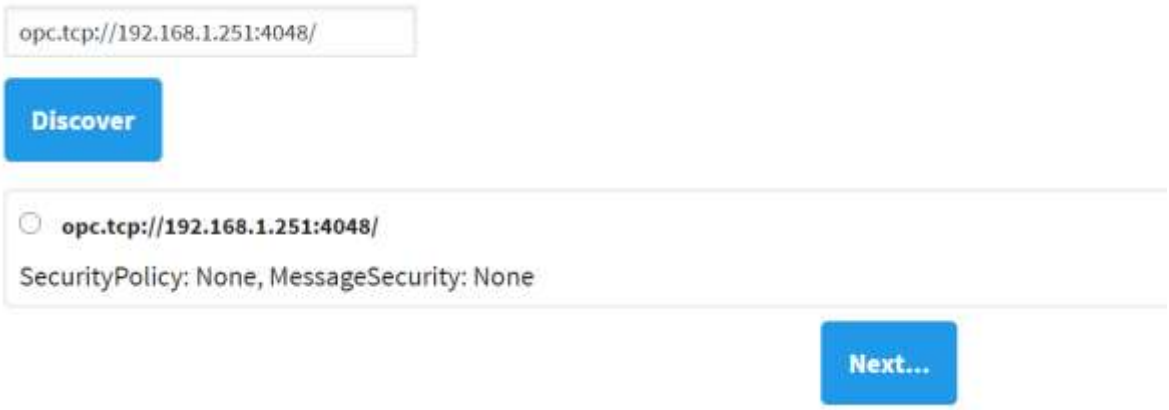
En la Figura 13 se han subrayado las IPs de cada conexión para que observe que las conexiones han sido tal y como se había descrito previamente. Se muestra por último una imagen donde se puede observar la aplicación del cliente *Prosys* de *Windows* conectado:

Figura 14. Cliente Prosys de Windows conectado a servidor.



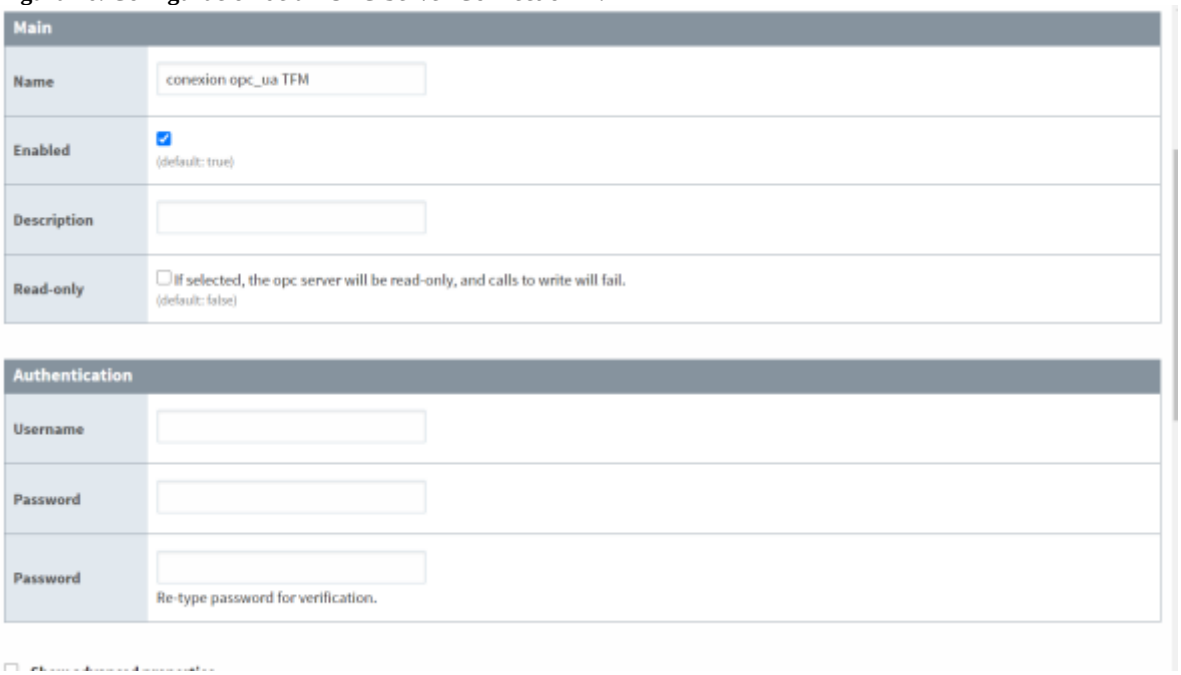
Por último, se va a conectar al servidor desde Ignition, creando una conexión OPC. Para ello primero es necesario acceder a la parte de *'OPC CONNECTIONS* > Servers'; una vez dentro se configura un nuevo dispositivo, lo primero que se pide es introducir el *endpoint* del servidor y seleccionar la política de seguridad y la seguridad de mensajes.

Figura 15. Configuración de un OPC Server Connection I.



La configuración del dispostivo se hace como se muestra en la Figura 16.

Figura 16. Configuración de un OPC Server Connection II.

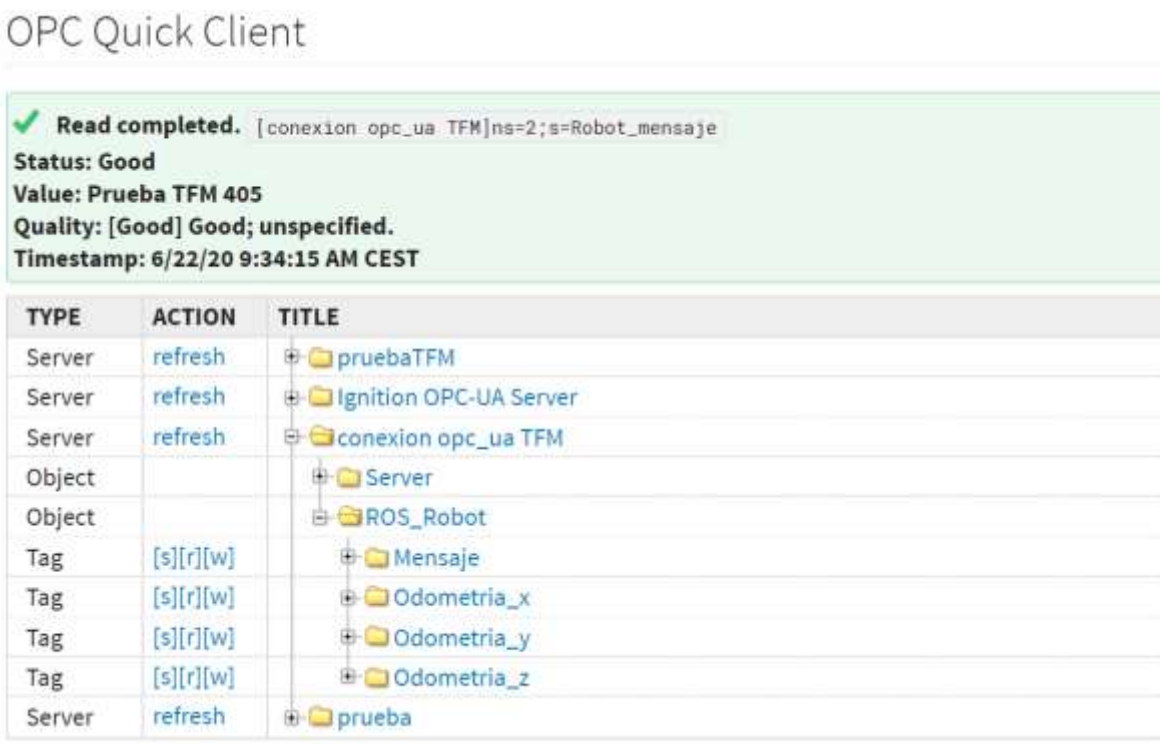


Cuando ya se ha configurado el dispositivo se puede comprobar si se ha establecido la conexión.

Figura 17. Configuración de un OPC Server Connection III.

Se puede acceder al servidor desde *Ignition* con una funcionalidad denominada *'Quick Client'*, que también se encuentra en el apartado de '*OPC CONNECTIONS'*. Al igual que con el Cliente de *Prosys* se puede leer el valor de los diferentes *tags*:

Figura 18. Funcionalidad Quick client de Ignition para acceder al servidor.



5.9 CONFIGURACIÓN DEL SERVIDOR A TRAVÉS DE UN ARCHIVO .XML

El objetivo de este proyecto es exportar los resultados obtenidos a robots reales por lo que se ha utilizado en este parte del trabajo un brazo robot. El robot que utilizado es una simulación predefina de ROS, denominada *UR_Gazebo*. Para poder utilizarlo se ha descargo y configurado.

```
$sudo apt-get install ros-kinetic-ur-gazebo ros-kinetic-ur5-
moveit-config ros-kinetic-ur-kinematics
```

Una vez instalado, para ejecutar la simulación:

```
$roslaunch ur_gazebo ur5.launch
```

Para poder controlar el brazo desde RViz se utilizará otro otro terminal:

```
$roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch
sim:=true
$roslaunch ur5_moveit_config moveit_rviz-launch config:=true
```

También se va a cambiar la filosofía de refresco de las variables. Antes, las variables se refrescaban cada vez que un cliente accedía a ellas, es decir, en el

momento en que un cliente realizaba una petición las variables se refrescaban para tener su valor actual. En este caso se ha utilizado un refresco periódico. Se han usado tres hilos distintos. Cada uno refresca unas variables concretas. Un hilo refresca las variables cada 200 milisegundos, por lo que esas variables cambian su valor en el servidor cada 200 milisegundos. Los otros dos hilos se encargarán de refrescar el resto de variables, uno cada 500 milisegundos y otro cada segundo.

En este trabajo también se pretende usar un archivo tipo xml para realizar la configuración del servidor; es decir, este archivo contendrá los topics a los que se pretende suscribir y los subtopics de este, además de la frecuencia de refresco de cada variable.

Al inicializarse el fichero *server.cpp* leerá el fichero xml, de manera que se configurará en base a él.

Se parte de un fichero .xml básico al que se irán añadiendo más funcionalidades. El contenido del fichero .xml inicial es:

```
< ? xml version = "1.0" encoding = "UTF-8" ? >
              <TFM>
              <Titulo>
              Topics del robot UR en ROS
              < / Titulo>

              <datos>
              <dato topic = "/joint_states" variable = "position" label =
"posicion_articulacion_1" freq = "200" tipo = "double" articulacion = "1" / >
              <dato topic = "/joint_states" variable = "velocity" label =
"velocidad_articulacion_1" freq = "500" tipo = "double" articulacion = "1" / >
              <dato topic = "/joint_states" variable = "name" label =
"nombre_articulacion_1" freq = "1000" tipo = "string" articulacion = "1" / >
              < / datos>

              < / TFM>
```

Se indica el *topic*, el *subtopic* dentro de ese *topic* que se denominará variable, el *label* que usado para nombrar a la variable dentro del servidor, la frecuencia de refresco, el tipo de dato al que se está suscribiendo, y por último, como el robot tiene 6 articulaciones (de la 0 a la 5) la articulación de la que se quiere obtener información. No todos los atributos van a ser necesarios para todos los *topics.*

Para poder leer archivos xml desde C/C++ es necesario usar una librería para realizar las funciones de *parser*. En este caso se ha elegido la librería *'pugixml.hpp'*, que se puede descargar de la siguiente manera:

```
$git clone https://github.com/zeux/pugixml.git
```

De los archivos que se descargan sólo se usarán *'pugiconfig.hpp', 'pugixml.hpp'* que se copiarán en la carpeta '*include'* del proyecto, y '*pugixml.cpp'* que se copiará en la carpeta 'src' del proyecto. Estas librerías externas se enlazan desde el *CMakeLists.txt*.
Se van a realizar varias modificaciones en el fichero *server.cpp* explicado

```
 add_library(pugixml src/pugixml.cpp)
 target_link_libraries(server   ${catkin_LIBRARIES}   open62541
pthread  pugixml)
```

previamente. Estas modificaciones se describen a continuación.

En primer lugar se incluye la librería que se va usar, así como las librerías que se utilizan en C/C++ para poder abrir fichero externos; también se define una constante MAX, que será el máximo de datos que puede ofrecer:

```
//Para poder leer archivos XML
#include <pugixml.hpp>
#include <iostream>
#include <fstream>
using namespace std;
#define MAX 100 //Máximo de datos del servidor
```

Se crea una estructura donde se encontrarán todos los datos necesarios para definir una variable en el servidor. En principio se va a trabajar con dos tipos de datos, *string* y *double*, por lo que para las variables que sean de tipo *string* se usará un dato y para las de tipo double el otro. También contará con un *mutex* propio para evitar problemas de condición de carrera ya que se dará el caso de acceso simultáneo a la variable que contiene el valor. El *struct* queda definido por tanto de la siguiente forma:

```
	struct dato_robot {
		char* topic;
		char* variable;
		char* label;
		int freq;
		char* tipo_dato;
		pthread_mutex_t mutex;
		UA_Double valor;
		UA_String valor_string;
		UA_VariableAttributes nAttr;
		int articulacion;
	};
```

Para poder acceder a las siguientes variables sin ningún problema se declaran como globales; la variable del servidor, la variable que contará el número total de datos de los distintos topics a los que se ha suscrito, la variable *dato_robot*, que es la estructura definida anteriormente y la variable *boolena running* que se usar

para controlar el encendido apagado del servidor, así como el final de los hilos de refresco.

```
	//Para controlar el apagado y encendido del servidor
	static volatile UA_Boolean running = true;
	//Declaración del servidor
	UA_Server *server;
	//Variables
	dato_robot ROS_datos[MAX];
	int total_datos;
```

Para cada *topic* distinto al que se desee suscribirse es necesario una función de *callback* nueva. En este primer fichero, sólo se va a suscribir al *topic /joint_states* del robot *UR_Gazebo*. Cada función de *callback*, accederá a todos los campos del *topic*, pero sólo guardará el valor de aquellos que se han pedido mediante el fichero xml.

```
//Función  topic /joint_states
void StatusCallback(const sensor_msgs::JointState::ConstPtr& msg) {
		for (int i = 0; i < total_datos; i++) {
			if (!strcmp(ROS_datos[i].variable, "position")) {
				pthread_mutex_lock(&ROS_datos[i].mutex);
				ROS_datos[i].valor = (UA_Double)msg-
>position[ROS_datos[i].articulacion];
				pthread_mutex_unlock(&ROS_datos[i].mutex);
			}
			if (!strcmp(ROS_datos[i].variable, "velocity")) {
				pthread_mutex_lock(&ROS_datos[i].mutex);
				ROS_datos[i].valor = (UA_Double)msg-
>velocity[ROS_datos[i].articulacion];
				pthread_mutex_unlock(&ROS_datos[i].mutex);
			}
			if (!strcmp(ROS_datos[i].variable, "effort")) {
				pthread_mutex_lock(&ROS_datos[i].mutex);
				ROS_datos[i].valor = (UA_Double)msg-
>effort[ROS_datos[i].articulacion];
				pthread_mutex_unlock(&ROS_datos[i].mutex);
			}
			if (!strcmp(ROS_datos[i].variable, "name")) {
				UA_String s;
				UA_String_init(&s); // _init zeroes out the entire
memory of the datatype
				char *test = (char *)msg-
>name[ROS_datos[i].articulacion].c_str();
				s.length = strlen(test);
				s.data = (UA_Byte*)test;
				pthread_mutex_lock(&ROS_datos[i].mutex);
				ROS_datos[i].valor_string = s;
				pthread_mutex_unlock(&ROS_datos[i].mutex);
			}
		}
	}
```

La función nodos_ROS, es la función que realiza el hilo asociado a ROS: crea un nodo denominado *'listener'* para escuchar los distintos nodos a los que se suscribe.

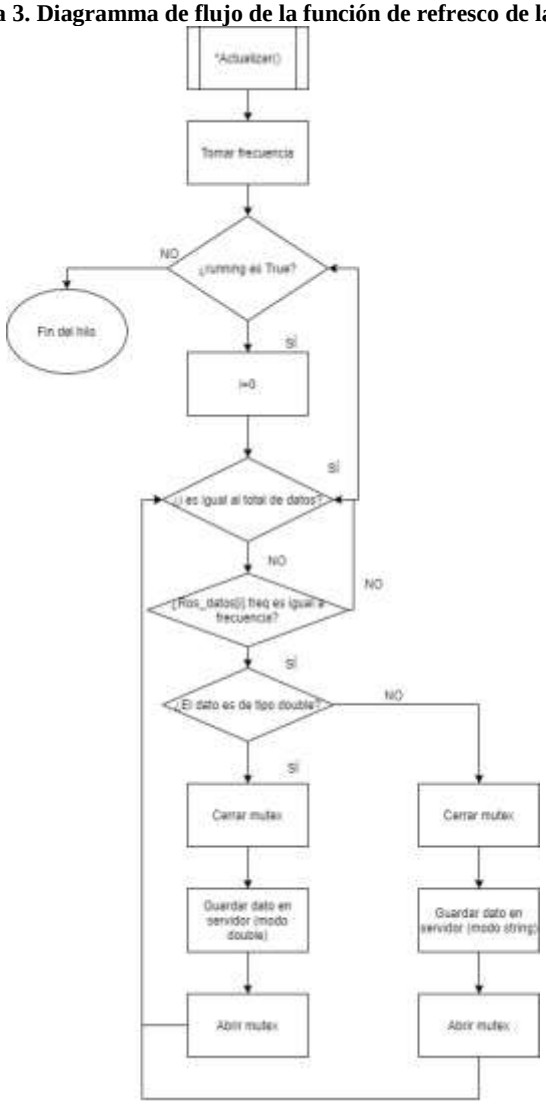
```
void *nodos_ROS(void *param) {
dato_ini ini = *(dato_ini*)param;
ros::init(ini.argc, ini.argv, "listener");
ros::NodeHandle nh;
ros::Subscriber sub_mensaje = nh.subscribe("joint_states", 1000,
StatusCallback);
ros::spin();
      }
```

Se incluye ahora la función que ejecutan los hilos de refresco, denominada Actualizar, que toma como parámetro de entrada la frecuencia de cada uno de los hilos.

```
void *Actualizar(void *param) {
      int freq = *(int*)param;
      while (running == true) {
            UA_Variant value;
            for (int i = 0; i < total_datos; i++) {
                  if (ROS_datos[i].freq == freq) {
                        if (!strcmp(ROS_datos[i].tipo_dato, "double")) {

      pthread_mutex_lock(&ROS_datos[i].mutex);
      UA_Variant_setScalarCopy(&value, &ROS_datos[i].valor,
                                          &UA_TYPES[UA_TYPES_DOUBLE]);

      pthread_mutex_unlock(&ROS_datos[i].mutex);
      UA_Server_writeValue(server,  UA_NODEID_STRING(2,  ROS_datos[i].label),
value);
                                    }
                        else {

      pthread_mutex_lock(&ROS_datos[i].mutex);
      UA_Variant_setScalarCopy(&value, &ROS_datos[i].valor_string,
                                          &UA_TYPES[UA_TYPES_STRING]);

      pthread_mutex_unlock(&ROS_datos[i].mutex);
      UA_Server_writeValue(server,  UA_NODEID_STRING(2,  ROS_datos[i].label),
value);
                                    }
                              }
                        }

                        usleep(freq);
                  }
      }
```

Para clarificar el funcionamiento de la misma se inlcuye un diagrama de flujo de la función:

Esquema 3. Diagramma de flujo de la función de refresco de las variables del servidor.



Dentro del hilo principal ya se declaran estas frecuencias de refresco, así como los nombres de los hilos:

```
pthread_t hilo_ros, hilo1000, hilo200, hilo500;
pthread_attr_t attr;
pthread_attr_init(&attr);
int freq1, freq2, freq3;
freq1 = 200;
freq2 = 500;
freq3 = 1000;
```

Para configurar desde el fichero xml, primero se leerá, se comprobará que se ha podido leer correctamente, y no ha habido errores durante la lectura y en caso afirmativo se inicializarán los campos del *struct* con los datos del fichero xml.

Es necesario recordar que la ubicación del fichero xml debe ser la misma que la del ejecutable del *server.cpp*, ya que en caso contrario se obtendrá el error de que no se encuentra el archivo.

```
//Configuración de variables que se leen a partir del archivo XML
pugi::xml_document doc;
pugi::xml_parse_result result = doc.load_file("datos.xml");
if (!result) {
	printf("Error leyendo el archivo xml\n");
	std::cout << "Load result: " << result.description();
	return -1; //Si no se puede leer el xml se cierra el programa

}

pugi::xml_node datos = doc.child("TFM").child("datos");

//Configurar datos desde el efichero .xml
int i = 0;
for (pugi::xml_node dato = datos.first_child(); dato; dato =
dato.next_sibling())
{

	ROS_datos[i].topic = (char*)dato.attribute("topic").value();
	ROS_datos[i].variable = (char*)dato.attribute("variable").value();
	ROS_datos[i].label = (char*)dato.attribute("label").value();
	ROS_datos[i].tipo_dato = (char*)dato.attribute("tipo").value();
	ROS_datos[i].freq = dato.attribute("freq").as_int();
	ROS_datos[i].mutex = PTHREAD_MUTEX_INITIALIZER;
	ROS_datos[i].articulacion = dato.attribute("articulacion").as_int();

	i++;
}
```

Finalmente, el contador de todos los datos que se han almacenado se guarda en la variable la variable global ‘total_datos’ para usarla posteriormente.

```
total_datos = i;
```

La inicialización del nodo asociado a ROS, y la configuración inicial del servidor se hace exactamente igual que en el anterior *server.cpp,* que ya se explicó en este documento, por lo que no volverá a explicarse.

Para añadir las variables al servidor se recorre el *array* de *dato_robot*, y se inicializa según el tipo de dato *double* o *string* de una manera u otra:

```
//Añadir variables de ROS_robot, de acuerdo al fichero .xml
for (int i = 0; i < total_datos; i++)
{
	if (!strcmp(ROS_datos[i].tipo_dato, "double"))
	{
		ROS_datos[i].nAttr = UA_VariableAttributes_default;
		UA_Variant_setScalar(&ROS_datos[i].nAttr.value,
&ROS_datos[i].valor, &UA_TYPES[UA_TYPES_DOUBLE]);
		UA_Server_addVariableNode(server, UA_NODEID_STRING(2,
ROS_datos[i].label), ROS_robot,
				UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
				UA_QUALIFIEDNAME(2, ROS_datos[i].variable),
				UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
ROS_datos[i].nAttr, NULL, NULL);
	}
	if (!strcmp(ROS_datos[i].tipo_dato, "string"))
	{
		ROS_datos[i].nAttr = UA_VariableAttributes_default;
		UA_Variant_setScalar(&ROS_datos[i].nAttr.value,
&ROS_datos[i].valor_string, &UA_TYPES[UA_TYPES_STRING]);
		UA_Server_addVariableNode(server, UA_NODEID_STRING(2,
ROS_datos[i].label), ROS_robot,
				UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
				UA_QUALIFIEDNAME(2, ROS_datos[i].variable),
				UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
ROS_datos[i].nAttr, NULL, NULL);
	}
}
```

Por último, se lanzan los hilos de refresco y el servidor:

```
//Lanzar 3 hilos para actualiar el valor de las variables, cada 200ms, cada
500ms, y cada 1 s
pthread_create(&hilo200, &attr, Actualizar, &freq1);
pthread_create(&hilo500, &attr, Actualizar, &freq2);
pthread_create(&hilo1000, &attr, Actualizar, &freq3);
//Lanzar servidor
UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Starting new Server...");
UA_StatusCode retval = UA_Server_run(server, &running);
UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Server was shut down");
UA_Server_delete(server);
//Matar el hilo de ROS
pthread_cancel(hilo_ros);
return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
```

5.10 CONFIGURACIÓN DEL SERVIDOR A TRAVÉS DE UN ARCHIVO .XML CON INSTROSPECCIÓN

La solución expuesta en el punto anterior es poco óptima, ya que se tendría que realizar una función de *callback* distinta por cada *topic* que se suscribiera. Para evitar esto se ha modificado el código de manera que se consiga una función de

callback genérica que sirva para todos los *topics* posibles, independientemente también del tipo de mensaje que contenga.

El fichero de configuración .xml se ha modificado al ser ahora los parámetros de configuración distintos, puesto que en el campo variable indica el número de articulación al que se refiere. El fichero tendrá la siguiente estructura:

```
< ? xml version = "1.0" encoding = "UTF-8" ? >
<TFM>
<Titulo>
Fichero de configuración donde se indica a que Topics se suscribe el servidor
< / Titulo>

<datos>
<dato topic = "/joint_states" variable = "/joint_states/position.0" label =
"posicion_articulacion_0" freq = "200" tipo = "double" / >
<dato topic = "/joint_states" variable = "/joint_states/velocity.0" label =
"velocidad_articulacion_0" freq = "500" tipo = "double" / >
<dato topic = "/joint_states" variable = "/joint_states/name.0" label =
"nombre_articulacion_0" freq = "1000" tipo = "string" / >
</datos>

< / TFM>
```

Para poder realizar la introspección se va a descargar la siguiente librería:

```
$git clone
https://github.com/facontidavide/ros_type_introspection.git
```

Este paquete será preciso copiarlo en el espacio de trabajo de ROS, es decir, en el directorio, *'catkin*_ws'. Para poder usar este paquete desde el paquete donde se está trabajando es necesario añadir las siguientes líneas de código al fichero *'CMakeLists.txt'*:

```
find_package(catkin REQUIRED COMPONENTS
     roscpp
    ros_type_introspection
    rosbag
    rosbag_storage
    topic_tools
    sensor_msgs
    geometry_msgs
    message_generation
    genmsg
    )
    generate_messages( DEPENDENCIES
    std_msgs
    )
    catkin_package(
    INCLUDE_DIRS
    LIBRARIES

    CATKIN_DEPENDS
    ros_type_introspection
    rosbag
    rosbag_storage
    topic_tools
    sensor_msgs
    geometry_msgs
    message_runtime

    DEPENDS
    )
```

Se han modificado los argumentos de *find_package*, no se ha añadido dos veces el mismo comando.
Por otro lado, también es necesario indicar las dependencias del paquete en el archivo *package.xml*, de manera que cuando se compile el proyecto se puedan encontrar todos los paquetes sin problemas. Las líneas a añadir en este fichero son:

```
<build_depend>ros_type_introspection</build_depend>
<build_depend>topic_tools</build_depend>
<build_depend>rosbag</build_depend>
<build_depend>rosbag_storage</build_depend>
<build_depend>message_generation</build_depend>
```

```
<exec_depend>ros_type_introspection</exec_depend>
<exec_depend>topic_tools</exec_depend>
<exec_depend>rosbag</exec_depend>
<exec_depend>rosbag_storage</exec_depend>
<exec_depend>message_runtime</exec_depend>
```

Una vez realizadas estas modificaciones se puede, ya incluir las librerías en el código:

```
//Para añadir introspección
#include "ros_type_introspection/ros_introspection.hpp"
#include <topic_tools/shape_shifter.h>
using namespace RosIntrospection;
using topic_tools::ShapeShifter;
```

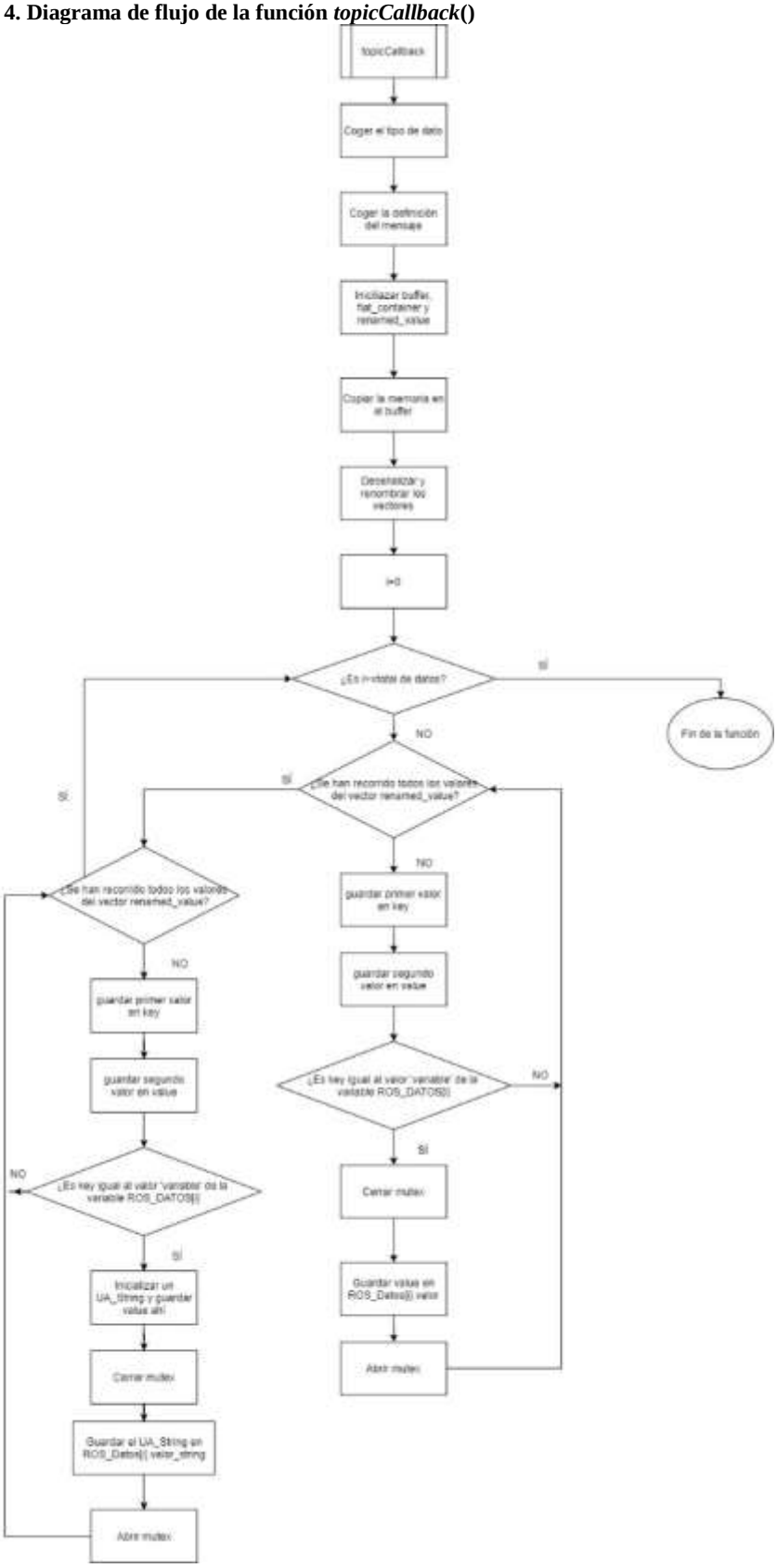
Lo siguiente a modificar será la función de *callback*, que en este caso no se indica en la cabecera el tipo de mensaje del *topic*, si no que se realizan unas funciones de deserialización para poder acceder a las partes del mensaje independientemente del tipo del mismo:

```
void topicCallback(const ShapeShifter::ConstPtr& msg, const std::string
&topic_name, RosIntrospection::Parser& parser) {

	const std::string&  datatype = msg->getDataType();
	const std::string&  definition = msg->getMessageDefinition();
	parser.registerMessageDefinition(topic_name,
		RosIntrospection::ROSType(datatype),
		definition);
	std::vector<uint8_t> buffer;
	FlatMessage   flat_container;
	RenamedValues renamed_value;

	// Copia de la memoria en el buffer.
	buffer.resize(msg->size());
	ros::serialization::OStream stream(buffer.data(), buffer.size());
	msg->write(stream);

	// deserializar y renombrar los vectores
	parser.deserializeIntoFlatContainer(topic_name, Span<uint8_t>(buffer),
&flat_container, 100);

	parser.applyNameTransform(topic_name, flat_container, &renamed_value);
```

```
	for (int i = 0; i < total_datos; i++) {
		for (auto it : renamed_value)
		{
			const std::string& key = it.first;
			const Variant& value = it.second;

			if (!strcmp(key.c_str(), ROS_datos[i].variable)) {
				pthread_mutex_lock(&ROS_datos[i].mutex);
				ROS_datos[i].valor =
(UA_Double)value.convert<double>();
				pthread_mutex_unlock(&ROS_datos[i].mutex);
			}
}

	for (auto it : flat_container.name)
		{
			const std::string& key = it.first.toStdString();
			const std::string& value = it.second;

			if (!strcmp(key.c_str(), ROS_datos[i].variable)) {
				UA_String s;
				UA_String_init(&s); // _init zeroes out the entire
//memory of the datatype
				char *test = (char *)value.c_str();
				s.length = strlen(test);
				s.data = (UA_Byte*)test;
				pthread_mutex_lock(&ROS_datos[i].mutex);
				ROS_datos[i].valor_string = s;
				pthread_mutex_unlock(&ROS_datos[i].mutex);
			}


		}
	}

}
```

Se incluye un diagrama de flujo de la función para clarificar el código:

Esquema 4. Diagrama de flujo de la función *topicCallback*()



La función donde se inicia el nodo que se dedica a suscribirse a todos los topics que se han demandado se ha modificado:

```
void *nodos_ROS(void *param) {
	dato_ini ini = *(dato_ini*)param;
	ros::init(ini.argc, ini.argv, "listener");
	ros::NodeHandle nh;

	Parser parser;
	for (int i = 0; i < total_datos; i++) {

		boost::function<void(const ShapeShifter::ConstPtr&) > callback;
		const std::string topic_name = ROS_datos[i].topic;
		callback = [&parser, topic_name](const ShapeShifter::ConstPtr& msg)
		{
			topicCallback(msg, topic_name, parser);
		};
		ROS_datos[i].subscriber = nh.subscribe(topic_name, 10, callback);
	}
	ros::spin();
}
```

También se ha modificado el *struct* de *dato_robot*, para prescindir del campo articulación, al igual que cuando se recogen los datos del archivo .xml ya que no se accede al campo articulación.

Gracias a estas modificaciones propuestas, ahora suscribirse a cualquier *topic* del robot, es bastante sencillo, puesto que sólo hay que indicarlo en el archivo .xml. Este nuevo código va a permitir obtener información de robots que tienen una gran cantidad de *topics* sin apenas esfuerzo ni modificaciones de código en el archivo .cpp.

Por último, se va a cambiar la manera de lanzar los hilos, en lugar de lanzar tres hilos fijos, se van a lanzar los hilos según se demande en el fichero de configuración. Se lanzará un hilo por cada frecuencia distinta, no habrá frecuencias infinitas, sino que estas estarán limitadas a 5, 100 milisegundos, 200 milisegundos,

```
//Declaración de un vecto de hilos

pthread_t hilo_ros, hilos_freq[5];

...

//Lanzar  hilos para actualiar sólo se lanzan aquellos hilos demandados por el
fichero de configuración
int cont_hilos = 0;
for (int i = 0; i < 5; i++) {
	if (ROS_datos[i].freq==100) {
		pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
		cont_hilos++;
		break;
	}
}
for (int i = 0; i < 5; i++) {
	if (ROS_datos[i].freq==200) {
		pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
		cont_hilos++;
		break;
	}
}
for (int i = 0; i < 5; i++) {
	if (ROS_datos[i].freq==500) {
		pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
		cont_hilos++;
		break;
	}
}
for (int i = 0; i < 5; i++) {
	if (ROS_datos[i].freq==1000) {
		pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
		cont_hilos++;
		break;
	}
}
for (int i = 0; i < 5; i++) {
	if (ROS_datos[i].freq==2000) {
		pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
		cont_hilos++;
		break;
	}
}
```

500 milisegundos, 1 segundo y 2 segundos. Se ha modificado la declaración de hilos y la parte donde se lanzan los mismos:

Para probar el codigo desarrollado se van a añadir más *topics* con distintas frecuencias, por lo que el fichero de configuración .xml, será:

```
< ? xml version = "1.0" encoding = "UTF-8" ? >
<TFM>
<Titulo>
Fichero de configuración donde se indica a que Topics se suscribe el servidor
< / Titulo>

<datos>
<dato topic = "/joint_states" variable = "/joint_states/position.0" label =
"posicion_articulacion_0" freq = "100" tipo = "double" / >
<dato topic = "/joint_states" variable = "/joint_states/velocity.0" label =
"velocidad_articulacion_0" freq = "500" tipo = "double" / >
<dato topic = "/joint_states" variable = "/joint_states/name.0" label =
"nombre_articulacion_0" freq = "2000" tipo = "string" / >
<dato topic = "/arm_controller/state" variable =
"/arm_controller/state/desired/velocities.4" label =
"velocidad_deseada_articulacion_4" freq = "1000" tipo = "double" / >
<dato topic = "/arm_controller/state" variable =
"/arm_controller/state/error/velocities.4" label =
"error_velocidad_articulacion_4" freq = "200" tipo = "double" / >

< / datos>

< / TFM>
```

5.11 INTERFAZ GRÁFICA PARA GENERAR EL ARCHIVO DE CONFIGURACIÓN .XML.

En este apartado se describe la interfaz gráfica que se ha creado usando el lenguaje de programación *Python*. Esta interfaz permite desarrollar los archivos de configuración .xml de manera automática. También se podrán editar archivos ya creados.

Es necesario realizar el cambio del compilador *Python 2.7* que es el que habitualmente usa ROS *kinetic* a *Python 3.6* para poder desarrollar la aplicación gráfica, ya que *Python 2.7* se encuentra obsoleto desde enero de 2020. En el 'Anexo 5. Configuración de los entornos de programación y del software utilizado' se detalla cómo se hace este cambio y cómo se configura. También se detalla cómo se ha creado un espacio de trabajo y una carpeta para almacenar los *scripts* de *Python* del proyecto.

Para realizar la interfaz gráfica se va usar la librería *tkinter.* Como entorno de programación se va a usar el software libre '*Visual Studio Code'*.

Las librerías necesarias para poder llevar a cabo el proyecto son :

```
import rospy
from std_msgs.msg import String
from tkinter import *
from tkinter import scrolledtext
from tkinter import ttk
from tkinter.ttk import *
from tkinter import messagebox
```

Para contar el número de datos que se han añadido se ha utilizado una variable global para que pueda ser usada en las distintas funciones que se crearán.

```
cont_lineas = 0
```

También es necesario acceder a todos los *topics* de ROS en los que se está publicando para mostarlos al usuario y que este pueda elegir a cuáles se suscribe y a cuáles no. Es importante mencionar que cuando se usa la función *get_published_topics()*, también se obtiene, aparte del nombre del *topic,* el tipo de mensajes que se publican. Se ha decidido mostrar sólo el nombre del *topic* por

```
topics_enteros = rospy.get_published_topics(namespace='/')
topics = topics_enteros
for i in range(len(topics_enteros)):
    aux = ' '
    aux = aux.join(topics_enteros[i])
    topics[i] = aux.split(" ")[0]
```

simplicidad y, también, porque el tipo de mensaje será algo que deberá conocer el usuario así como las distintas variables que se encuentran dentro del *topic*. Toda esta información es accesible en la documentanción de ROS.

La propuesta inicial es crear una aplicación gráfica sencilla, pero cuya funcionalidad cumpla las necesidades básicas de la creación de los archivos *xml* de configuración, y se facilite la tarea. Para crear la ventana se ha escogido un tamaño amplio puesto que se pretende visualizar en todo momento el archivo que se está editando:

```
#------------Configuración window----------#
window = Tk()
window.title("Crear archivo .xml para configurar servidor")
window.geometry("1500x1500")
```

Los elementos gráficos de configuración que se han considerado necesario incluir sin, por una parte, de tipo texto para mostrar y/o pedir/seleccionar información y, por otra, de tipo botones de acción, para ejecutar acciones.
Los cuadro de texto que se han incluido y su función son los siguientes:

- Un cuadro de texto donde se puede escribir el nombre que se le dará al dato, así como la etiqueta que indica lo que es el cuadro de texto.
- Un *comboText,* donde se podrán elegir todos los *topics* que hay disponibles desde la última vez que se comprobó junto con su etiqueta.
- Un cuadro de texto donde se escribirá el nombre de la variable de la que se obtendrá la información.
- Un *comboText* donde se mostrarán todas las variables posibles de refresco.
- Un cuadro de texto con funcionalidad de scrolled bar (barra de desplazamiento) para mostrar en tiempo real todos los datos que se vayan añadiendo al archivo *xml*.
- Un *comboText* donde se irán escribiendo los datos que se han incluido ya en el archivo. En lugar del nombre del dato se podrán el número de dato puesto que evitar posibles errores si el usuario añade dos veces el mismo nombre.
- Un cuadro de texto escribir la ruta de un archivo *xml*, en caso de que se quiera editar uno ya creado.
- Un cuadro de texto para elegir el nombre del archivo antes de guardarlo; si el ususario no pone ningún nombre por defecto se guardará como ‘*config.xml*’.

Se muestran en los siguientes recuadros los códigos desarrollados para su creación en la aplicación.

```
#----------Para entrar el nombre ---------#
label1 = Label(window, text = "Nombre", font= ("Calibri", 12))
label1.grid(column = 1, row = 1)
texto1 = StringVar()
txt1 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto1)
txt1.grid(column = 2, row = 1)
#--------Para mostrar los topics posibles a los que suscribirse topics--#
combo2 = Combobox(window, font= ("Calibri", 12))
combo2['values'] = (topics)
combo2.current(0)
combo2.grid(column = 2, row = 2)
label2 = Label(window, text = "Topic", font= ("Calibri", 12))
label2.grid(column = 1, row = 2)
```

```
#--------Para mostrar la variable-------#
label3 = Label(window, text = "Variable", font= ("Calibri", 12))
label3.grid(column = 1, row = 3)
texto3 = StringVar()
txt3 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto3)
txt3.grid(column = 2, row = 3)
#----------Para tipo de dato------------#
label8 = Label(window, text = "Tipo de dato", font= ("Calibri", 12))
label8.grid(column = 1, row = 4)
texto8 = StringVar()
txt8 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto8)
txt8.grid(column = 2, row = 4)
#---------Para elegir la frecuencia de refresco-----------#
combo4 = Combobox(window, font= ("Calibri", 12))
combo4['values'] = (100, 200, 500, 1000, 2000)
combo4.current(0)
combo4.grid(column = 2, row = 5)
label4 = Label(window, text = "Frecuencia refresco (ms)", font=
("Calibri", 12))
label4.grid(column = 1, row = 5)
```

```
#----------Vista previa archivo xml----------#
archivo = scrolledtext.ScrolledText(window, width=200, height=40)
archivo.grid(row=0, column=0,  columnspan=12, sticky=E+W+N+S)
```

```
#-----------------Para cargar un archivo xml------#
label6 = Label(window, text = "Ruta archivo a cargar", font= ("Calibri",
12))
label6.grid(column = 3, row = 2)
texto6 = StringVar()
txt6 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto6)
txt6.grid(column = 4, row = 2)
#-----------------Nombre del archivo xml generado------#
label7 = Label(window, text = "Nombre archivo xml", font= ("Calibri",
12))
label7.grid(column = 3, row = 3)
texto7 = StringVar()
txt7 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto7)
txt7.grid(column = 4, row = 3)
```

Las funcionalidades que aportan los botones que se han incluido en la aplicación son las siguientes:

- Añadir un dato cuando los campos de configuración hayan sido rellenados
- Borrar todos los campos en caso de que haya algo escrito
- Refrescar los *topics*, es decir, se volverá a comprobar si hay *topics* nuevos de ROS o no
- Borrar un dato que ya haya sido añadido
- Generar el archivo *xml*, una vez se haya terminado de escribir
- Cargar un archivo *xml* ya existente.

  Se muestra en el siguiente recuadro el código desarrollado para la creación de estos botones.

```
#------------------------------Botones---------------------------#
add_button = Button(window, text = "Añadir dato", command = add_topic,
width = 20)
dlt_button = Button(window, text = "Borrar campos", command = dlt_fields,
width = 20)
ref_button = Button(window, text = "Refrescar topics", command =
ref_topics,   width = 20)
dlt2_button = Button(window, text = "Borrar dato", command = dlt_topics,
width = 20)
gen_button = Button(window, text = "Generar xml", command = gen_file,
width = 20)
get_button = Button(window, text = "Cargar xml", command = get_file,
width = 20)
add_button.grid(column = 1, row = 10)
dlt_button.grid(column = 1, row = 12)
ref_button.grid(column = 2, row = 12)
dlt2_button.grid(column = 2, row = 10)
gen_button.grid(column = 3 , row = 10)
get_button.grid(column = 3, row = 12)
```

El archivo .xml, no se crea en vacío, si no que se le aporta al usuario una plantilla predeterminada, sobre la que se va escribiendo:

```
#--------------------Configurar archivo xml ----------------------#
archivo.insert('1.0','<?xml version = "1.0" encoding = "UTF-8"?>\n' )
archivo.tag_add('Linea 1', '1.0','2.0')
archivo.tag_config('Linea 1', foreground = 'black',font =("console",
10,"bold"))
archivo.insert('2.0','  <TFM>\n')
archivo.tag_add('Linea 2', '2.0','3.0')
archivo.tag_config('Linea 2', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('3.0','       <Título>\n')
archivo.tag_add('Linea 3', '3.0','4.0')
archivo.tag_config('Linea 3', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('4.0','              Topics  ROS robot\n')
archivo.tag_add('Linea 4', '4.0','5.0')
archivo.tag_config('Linea 4', foreground = 'black', font =("console",
10))
```

```
archivo.insert('5.0' , '       </Título>\n')
archivo.tag_add('Linea 5' , '5.0','6.0')
archivo.tag_config('Linea 5', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('6.0' , '               <datos>\n')
archivo.tag_add('Linea 6' , '6.0','7.0')
archivo.tag_config('Linea 6', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('7.0' , '               </datos>\n')
archivo.tag_add('Linea 7' , '7.0','8.0')
archivo.tag_config('Linea 7', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('8.0','  </TFM>\n')
archivo.tag_add('Linea 8', '8.0','9.0')
archivo.tag_config('Linea 8', foreground = 'black', font =("console",
10,"bold"))
```

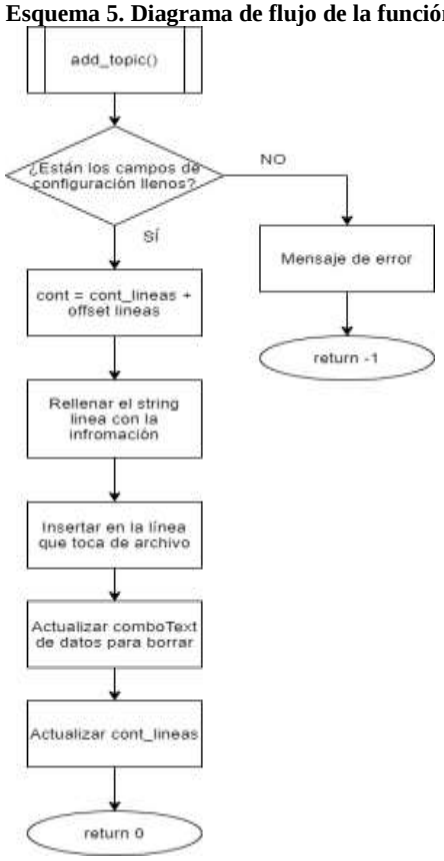
Se describen en los siguientes párrafos la funcionalidad asociada a cada botón y se incluye un diagrama de flujo de la misma.

- Añadir dato: esta funcionaliad comprueba si hay algo escrito en los campos de configuración y, en caso afirmativo, añade el dato de acuerdo a la sintaxis del archivo .xml. Además, se ocupa de actualizar la variable global que cuenta los datos, así como el *comboText* referido a los datos que se pueden borrar.

```
#----------Función añadir dato----------#
def add_topic():
    if txt1.get()=="" or txt3.get()=="":
        messagebox.showinfo(title="Mensaje de error", message="Para
añadir un topic ningún campo puede estar vacio")
        return -1
    global cont_lineas

    cont = StringVar()
    cont = str(cont_lineas + 7.0)
    linea = StringVar()
linea ='                    <dato topic="'+combo2.get()+'"
variable="'+combo2.get()+'/' +txt3.get()+'" label="'+txt1.get()+'
tipo="'+txt8.get()+'" freq="'+combo4.get()+'"/>\n'
    archivo.insert(cont, linea)
    cont_lineas = cont_lineas + 1
```

```
    lista_cont = [0]*(cont_lineas+1)
    for i in range(cont_lineas+1):
        lista_cont[i] = i
    combo5['values'] = (lista_cont)
    combo5.current(0)
    return 0
```

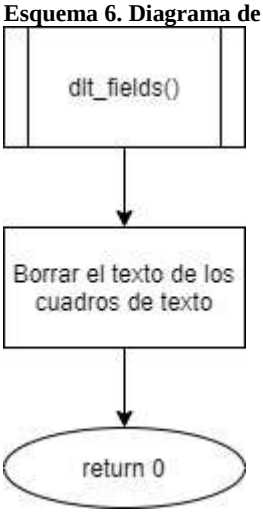
Esquema 5. Diagrama de flujo de la función add_topic()



- Borrar los campos: esta funcionalidad borra los campos en caso de haber sido rellenados, en caso contrario no hace nada.

```
#---------------Función borrar campos------#
def dlt_fields():
    texto1.set("")
    texto3.set("")
    texto6.set("")
    texto7.set("")
    return 0
```

Esquema 6. Diagrama de flujo de la función dlt_fields()



- Refrescar los *topics*: esta función vuelve a coger los *topics* de ROS sobre los que se está publicando y los añade al *comboText* que se dedica a mostrarlos, actualizando de esta manera la lista.

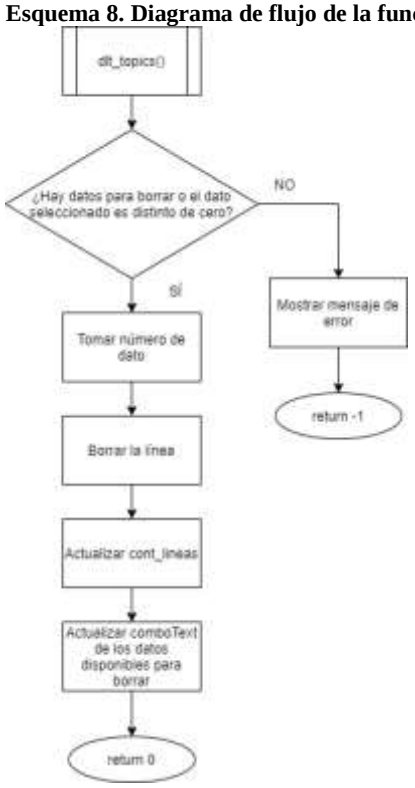
```
#---------------Función refrescar topics---------#
def ref_topics():
    topics_enteros = rospy.get_published_topics(namespace='/')
    topics = topics_enteros
    for i in range(len(topics_enteros)):
        aux = ' '
        aux = aux.join(topics_enteros[i])
        topics[i] = aux.split(" ")[0]

    combo2['values'] = (topics)
    combo2.current(0)
    return 0
```

Esquema 7. Diagrama de flujo de la función ref_topics()

- Borrar un dato: esta funcionalidad borra un dato que se ha añadido. Para ello se toma el número de dato que se ha seleccionado en el *comboText* específico para ello, se calcula en qué línea del archivo está y se borra. Posteriormente se actualiza el *comboText* donde están los datos que se pueden borrar.

```
#----------Función borrar dato----------#
def dlt_topics():
    if combo5.get()=='0' or cont_lineas==0:
        messagebox.showinfo(title="Mensaje de error", message="Por favor
selecione un dato para borrar")
        return -1
    global cont_lineas
    ini = float(combo5.get())+6.0
    fin = float(combo5.get())+7.0
    archivo.delete(ini, fin)
    cont_lineas = cont_lineas - 1
    lista_cont = [0]*(cont_lineas + 1)
    for i in range(cont_lineas+1):
        lista_cont[i] = i
    combo5['values'] = (lista_cont)
    combo5.current(0)
    return 0
```

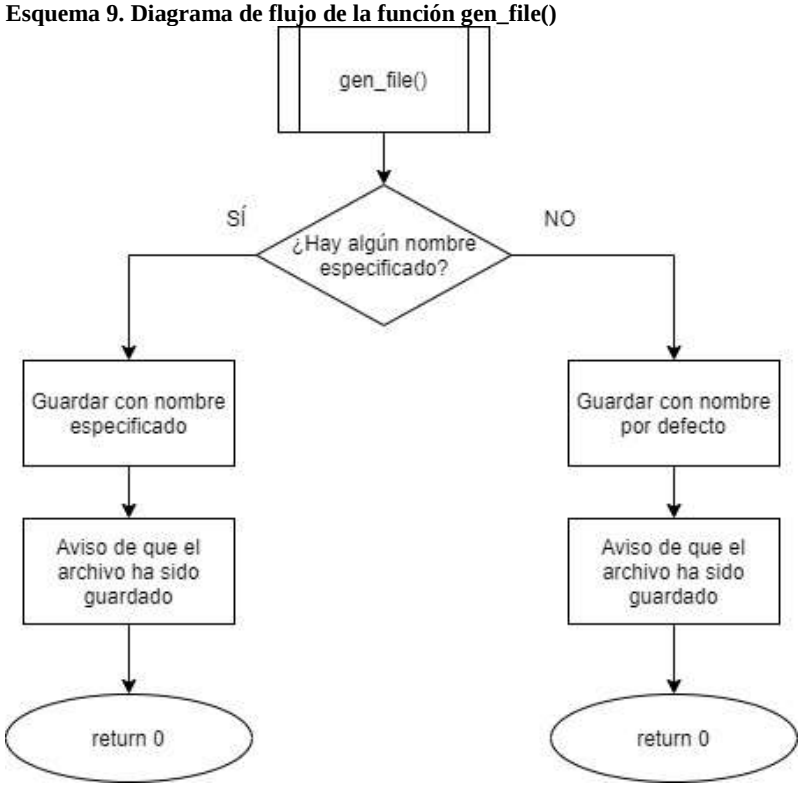
Esquema 8. Diagrama de flujo de la función dlt_topics()



- Generar archivo *xml*: para generar el archivo *xml*, esta función comprueba primero si el usuario ha escrito algún nombre en el campo correspondiente, en ese caso usa ese valor para guardarlo, en caso contrario guarda el archivo con el nombre por defecto y muestra un mensaje de aviso.

```
#------------------------Función generar archivo xml--------#
def gen_file():
    if txt7.get() == "":
        file = open ("./config.xml", 'w')
        file.write(archivo.get("1.0",END))
        messagebox.showinfo(title="Aviso", message="Archivo XML generado
con el nombre config.xml")
        return 0
    nombre = './' + txt7.get()
    file = open (nombre, 'w')
    file.write(archivo.get("1.0",END))
    messagebox.showinfo(title="Aviso", message="Archivo XML generado con
el nombre " + txt7.get())
    return 0
```

Esquema 9. Diagrama de flujo de la función gen_file()

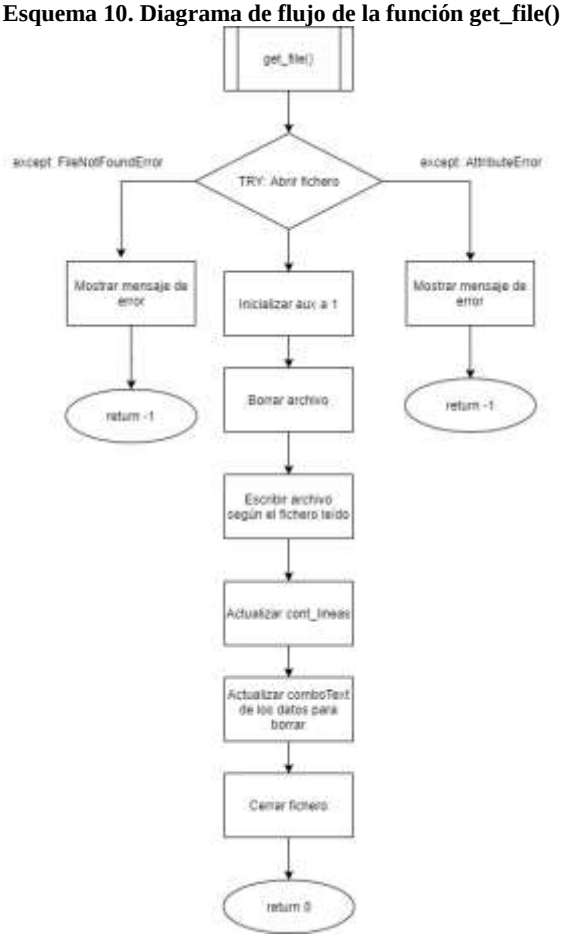


- Cargar un archivo *.xml* ya existente para poder editarlo. Para ello se intenta leer la ruta que se ha especificado, en caso de que ocurra algún error se notificará; si por el contrario todo ha sido correcto se mostrará el archivo por pantalla, y se actualiarán los datos que se pueden borrar.

```
#----------Función cargar archivo xml---------#
def get_file():
    try:
        file = open(txt6.get(), 'r')
        aux = 1.0
        archivo.delete("1.0", END) #Borrar datos de archivo
        for j in file.readlines():
            cont = StringVar()
            cont = str(aux)
            archivo.insert(cont, j)
            aux = aux + 1.0
    except AttributeError:
        messagebox.showinfo(title="Mensaje de error", message="Fichero no
insertado")
        return -1
```

```
    except FileNotFoundError:
        messagebox.showinfo(title="Mensaje de error", message="La ruta no
es correcta")
        return -1
    global cont_lineas
    file.seek(0)
    cont_lineas = len(file.readlines())-9
    lista_cont = [0]*(cont_lineas + 1)
    for i in range(cont_lineas+1):
        lista_cont[i] = i
    combo5['values'] = (lista_cont)
    combo5.current(0)
    #Escribir los datos en archivo del file que se ha abierto
    # archivo.insert(cont, linea)
    file.close()
    return 0
```

Esquema 10. Diagrama de flujo de la función get_file()



Se muestra la apariencia de la interfaz gráfica desarrollada. En el apartado de ‘Prueba 1. Funcionamiento de la interfaz gráfica para crear los archivos *.xml.*’, se mostrará su funcionamiento.

Figura 19. Apariencia de la interfaz gráfica para la generación de archivos xml.

5.12 HABILITAR LA SELECCIÓN DE CUALQUIER ARCHIVO DE CONFIGURACIÓN .XML.

Uno de los objetivos de este trabajo era realizar una aplicación que pueda trabajar con diversos robots sin tener que modificar el código de la aplicación. Sin embargo, el código del archivo *server.cpp* sólo permite la carga de un archivo de configuración .xml específico, que era el archivo '*datos.xml*' que se encontraba en el mismo directorio que el ejecutable. Por ello, se ha añadido el código que permite al usuario especificar por consola la ruta del archivo de configuración a cargar. Si no se escribe nada o si escribe una ruta erronea, automáticamente se cierra el programa. Cuando se pone un nombre correcto se carga el archivo especificado. Las líneas de código que se han añadido en la parte de la carga del archivo .xml son:

```
//Configuración de variables que se leen a partir del archivo XML
char archivo[100];
cout << "Ruta del archivo .xml de configuración:\n";
cin.getline(archivo, 100, '\n');
pugi::xml_document doc;
pugi::xml_parse_result result;
result = doc.load_file(archivo);
```

A continuación se muestra el resultado cuando se le especifica la ruta del archivo, el archivo que se especificará por teclado será el mismo *'datos.xml'*.

Figura 20. Resultado de la ejecución de server cuando no se especifica archivo por teclado.

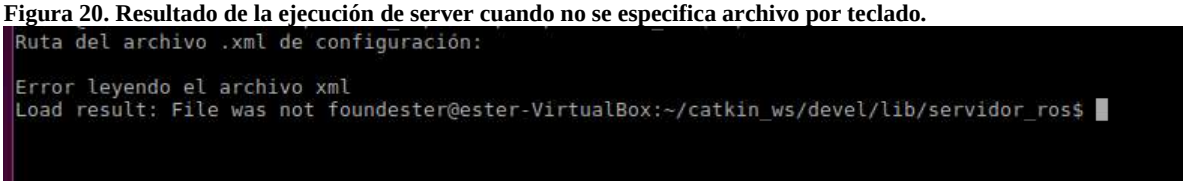


Figura 21. Resultado de la ejecución de server cuando sí se especifica archivo por teclado.

6. PRUEBAS

Este apartado va a estar dedicado al desarrollo de las pruebas para poder verificar la validez del sistema que se ha desarrollado. Se van a realizar pruebas mediante simulaciones y posteriormente con robots reales en el laboratorio.

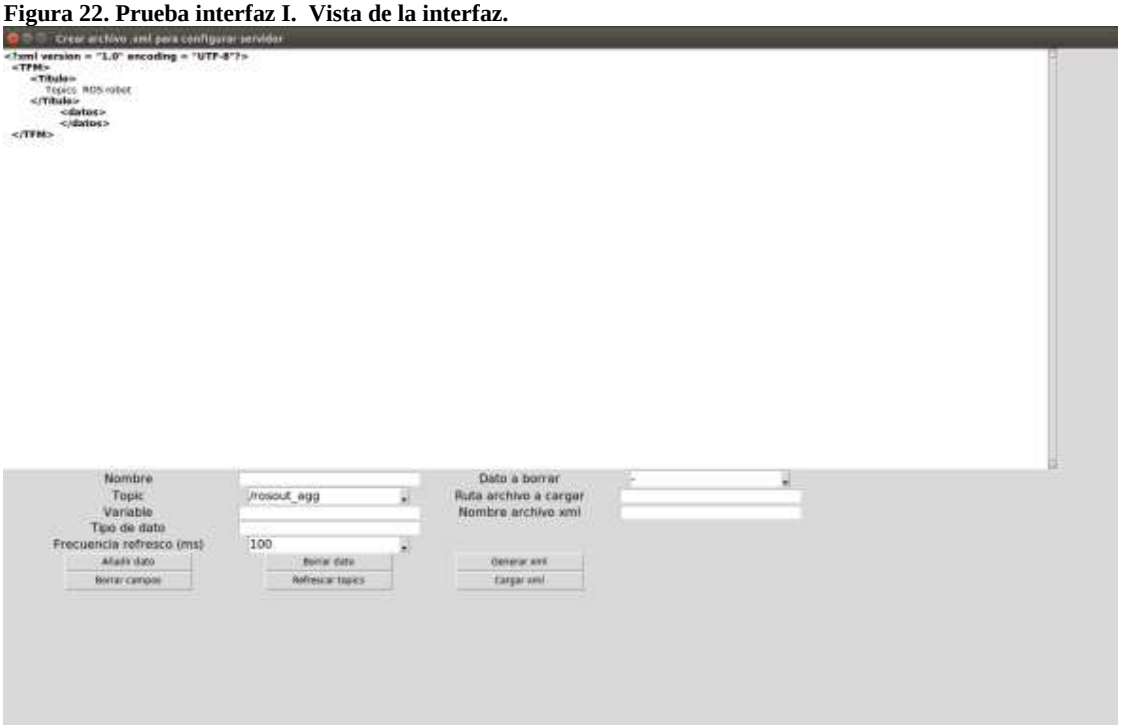
6.1 PRUEBAS TEÓRICAS.

6.1.1 Prueba 1. Funcionamiento de la interfaz gráfica para crear los archivos *.xml.*

La primera prueba permite comprobar si todas las funciones que realiza la interfaz gráfica se suceden de manera correcta y su validez como herramienta para la creación de archivos. Para la realización de esta prueba, se lanza la interfaz:
En la siguiente figura se muestra cómo es esta interfaz.

```
$python3 GUI
```

Figura 22. Prueba interfaz I. Vista de la interfaz.



Pulsado en la lista *topics* se puede comprobar que están todos los *topics* actuales del sistema.

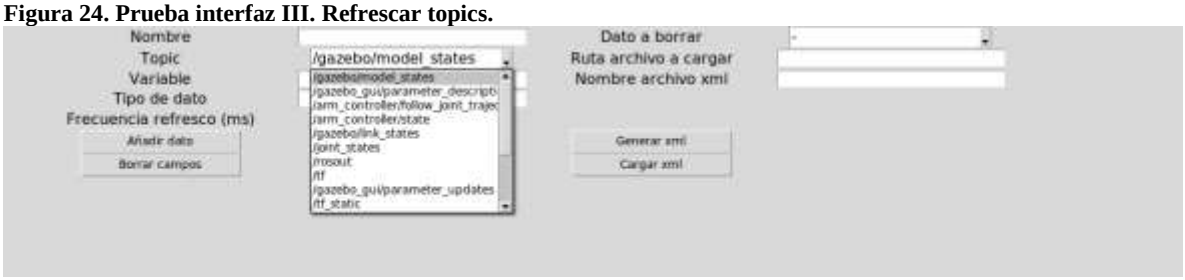
Figura 23. Prueba interfaz II. Topics disponibles.

Se verifica después que se pueden mostrar los *topics* de un robot. Para ello, se iniciar el robot en ROS, por ejemplo:

```
$roslaunch ur_gazebo ur5.launch
```

Tras realizar esto, si se vuelve a la interfaz, y se pulsa sobre el botón *'refrescar topics'* se comprobará que se han añadido todos los *topics* nuevos que hay ahora en ROS:

Figura 24. Prueba interfaz III. Refrescar topics.

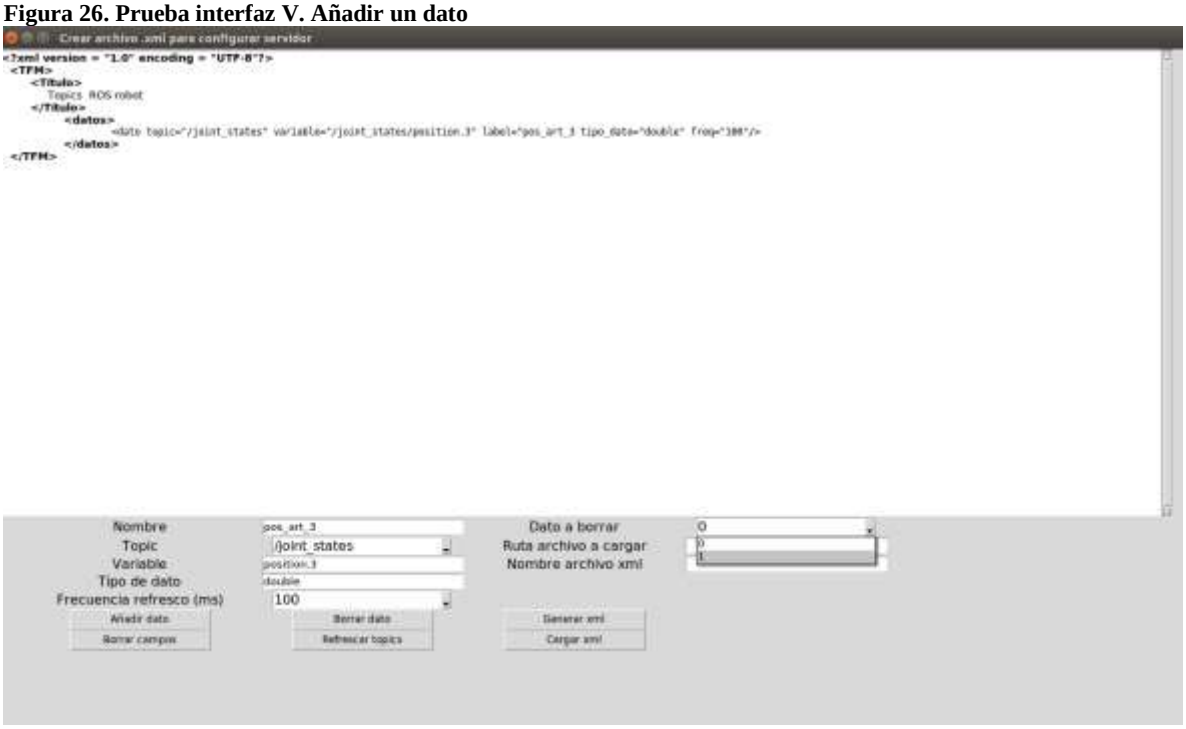


También se ha comprobado el funcionamiento del botón *borrar* cuando no hay datos añadidos. La lista de datos posibles a borrar no muestra ningún valor, como se puede apreciar en la Figura 87.

Figura 25. Prueba interfaz IV.

Para añadir un dato se deben rellenar los cinco campos de configuración, en esta prueba se va a realizar la suscripción al *topic /joint_states*, en concreto a la variable de posición de la articulación 3, con una frecuencia de refresco de 100 ms.

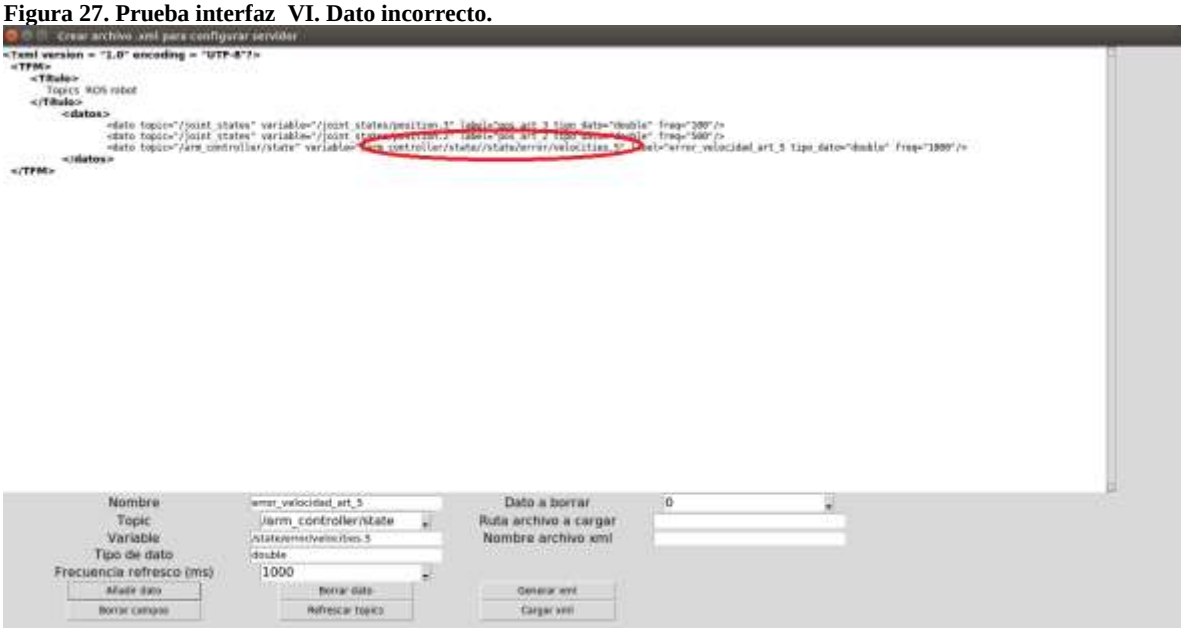
Figura 26. Prueba interfaz V. Añadir un dato



Una vez que se ha añadido un dato, se puede comprobar que la lista de datos a borrar ya lo contempla, tal como muestra en la Figura 88.

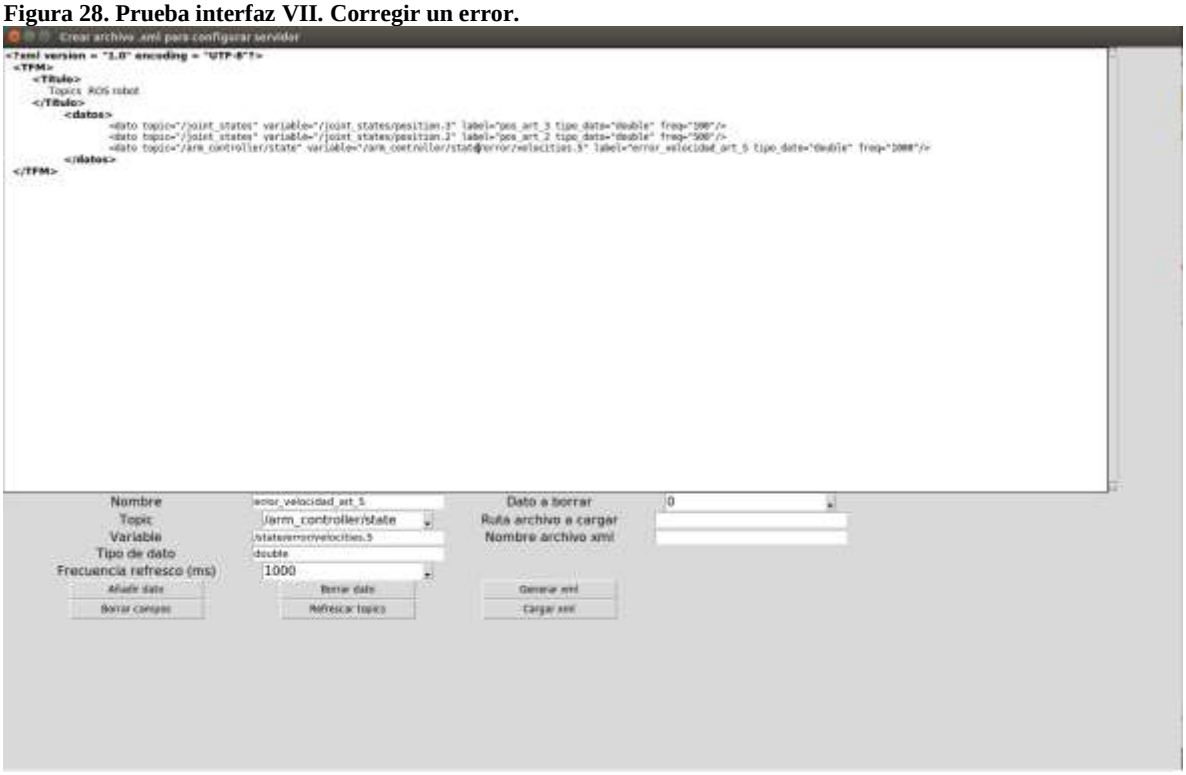
Si por ejemplo se añade un *topic* y en uno de los campos se ha cometido un error, como es el caso de la Figura 89, la interfaz permitirá corregirlo de las siguientes maneras:

Figura 27. Prueba interfaz VI. Dato incorrecto.



1. Se puede editar directamente el fichero, pues en la ventana del archivo, se puede escribir y borrar.

Figura 28. Prueba interfaz VII. Corregir un error.



2. También se puede borrar directamente el dato completo, seleccionando el número de dato que se quiere eliminar y pulsando sobre el botón '*borrar dato'*.

Se comprueba que ambas opciones funcionan de la manera esperada.

Figura 29. Prueba interfaz VIII. Borrar un dato.

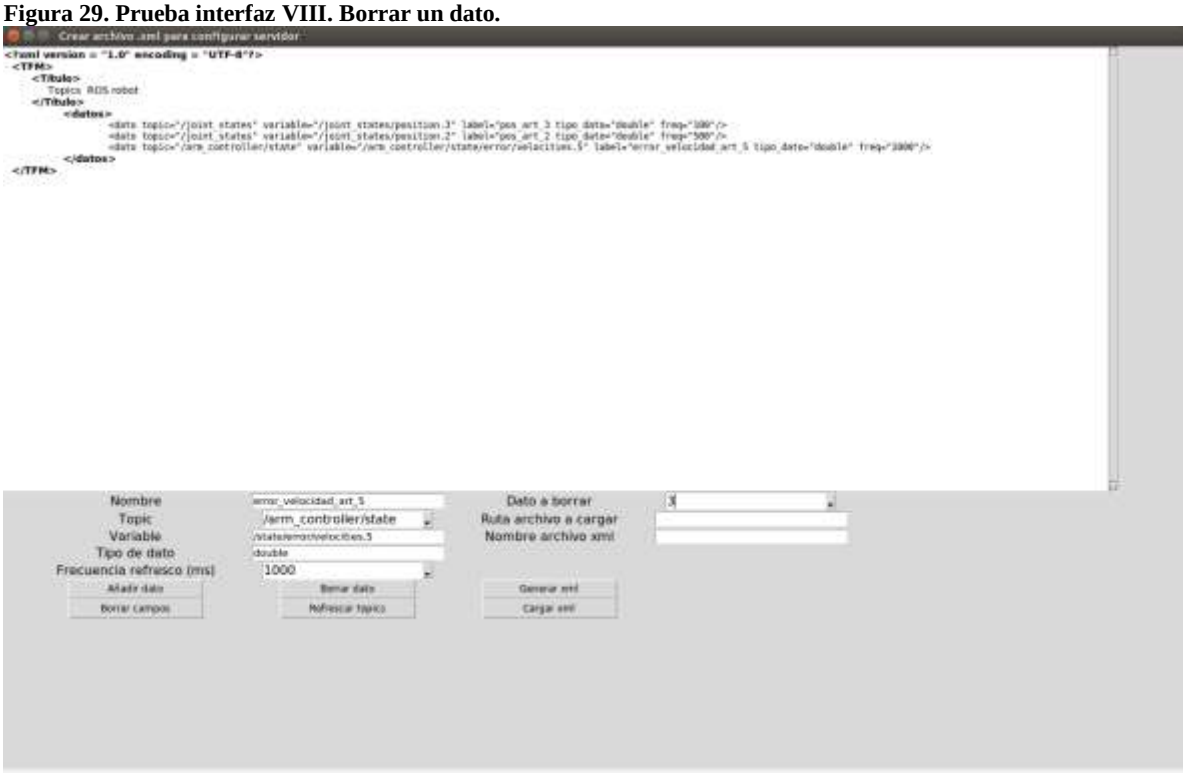
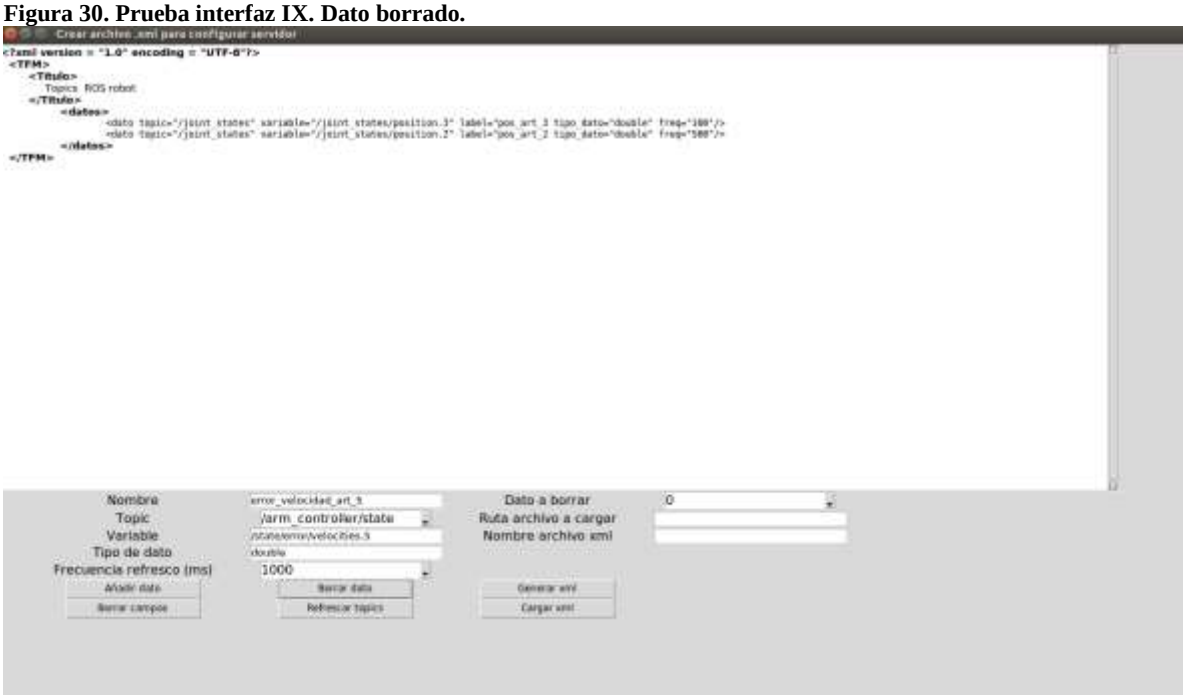
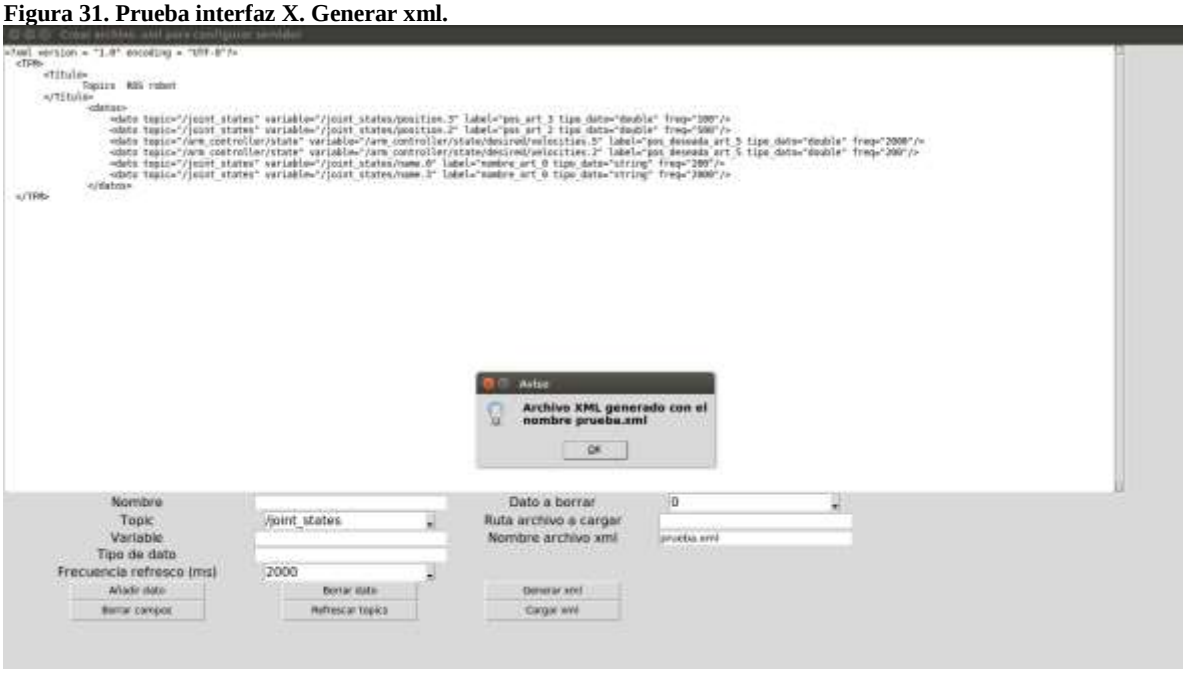


Figura 30. Prueba interfaz IX. Dato borrado.



Cuando se ha terminado de desarrollar el fichero .xml, ya se puede generar. Para ello se escribe un nombre en la casilla de '*Nombre archivo xm*l' y se pulsa el botón *'Generar xml'.*

Figura 31. Prueba interfaz X. Generar xml.



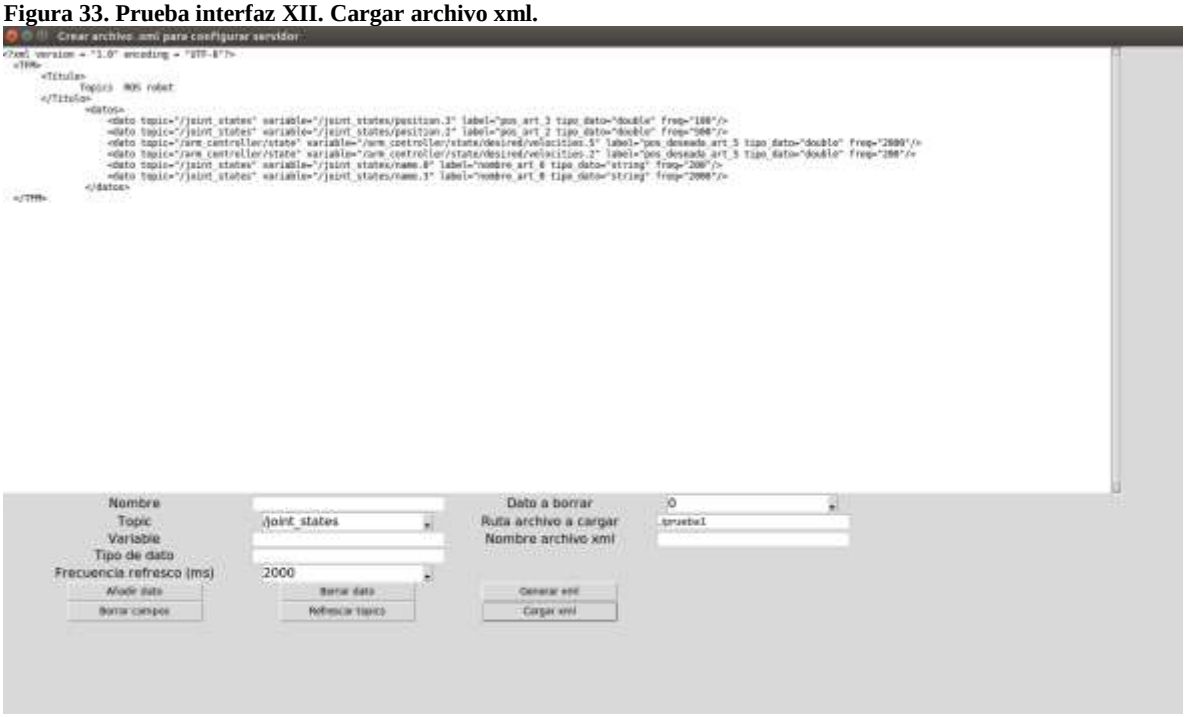
Si no se especifica ningún nombre el archivo se guarda con el nombre predeterminado 'config.xml'.

También se pueden borrar todos los campos de la sección de configuración, pulsando el botón *'Borrar campos'*.

Figura 32. Prueba interfaz XI. Borrar campos.

Otras de las funcionalidades de la aplicación es la carga y edición de un archivo .xml ya existente. Para ello es necesario rellenar el campo de '*Ruta archivo a cargar'* y pulsar el botón '*Cargar xml'*.

Figura 33. Prueba interfaz XII. Cargar archivo xml.



Sobre este archivo cargado se pueden realizar todas las acciones descritas anteriormente para modificar el archivo, y una vez hecho esto guardar el archivo, tal y como se ha indicado anteriormente.

Todos los archivos guardados se guardan en la carpeta '*scripts'* en la que se encuentra la aplicación.

Para comprobar que la prueba ha tenido validez se ha cargado desde un navegador el archivo .xml, y posteriormente se ha puesto en marcha la aplicación *server* utilizando el archivo '*prueba.xml'* (creado para estas pruebas).

Figura 34. Prueba interfaz XIII. Archivo desarrollado cargado desde un navegador.

Para continuar con las pruebas, se ha copiado este archivo en la carpeta donde se encuentra el ejecutable de la aplicación *server*:

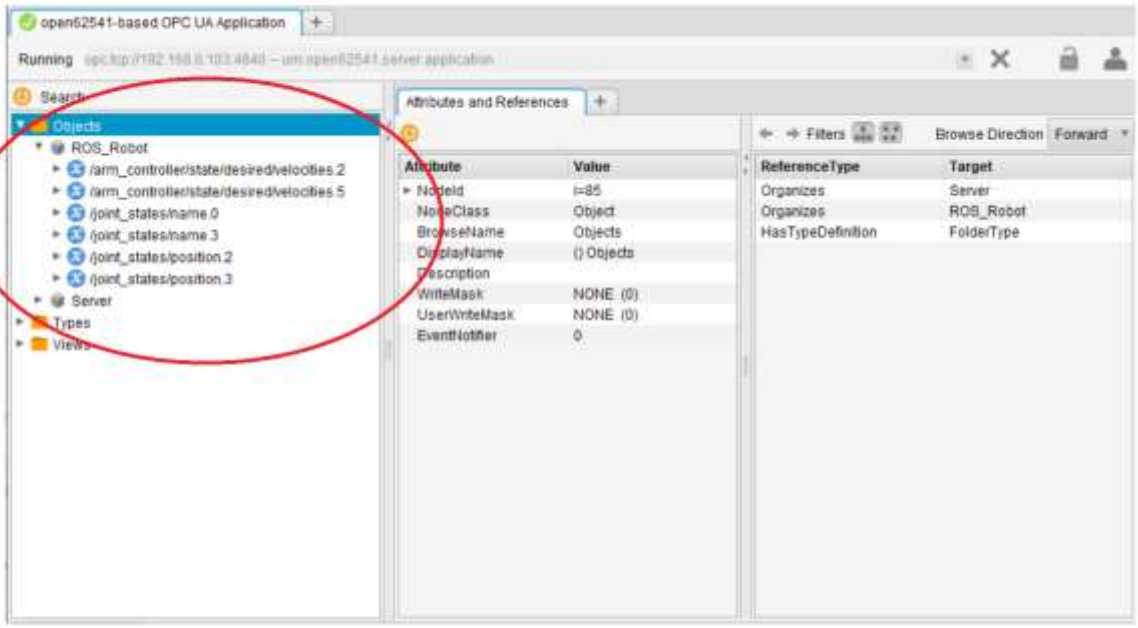
```
$cp -r ./prueba.xml ~/catkin_ws/devel/lib/servidor_ros
```

Al lanzar la aplicación con el archivo creado se obtiene:

Figura 35. Prueba interfaz XIV. Usar aplicación *sever* con el archivo creado.

Por último, se puede comprobar que los datos creados mediante el archivo *prueba.xml* se han configurado correctamente en el servidor:

Figura 36. Prueba interfaz XV. Comprobación de que se han creado correctamente los datos en el servidor.



Se por concluida con éxito esta primera prueba.

6.1.2 Prueba 2. Robot parado sin realizar ninguna tarea.

El objetivo de esta prueba es verificar que los valores que se obtienen son consistentes con la situaicón de robot parado. Para ello, la prueba se incia lanzando la aplicación gazebo, para simular el robot en un estado de parada y se comprueba si todos los datos tienen valores reales. También se comprobará si los datos teóricos que se pueden obtener desde el sistema ROS sobre los parámetros del robot coinciden con los datos que se obtienen en el servidor diseñado. De esta manera, se podrá comprobar si existen interferencias en el traspaso de datos o corrupción de los mismos, o si por el contrario todo funciona de manera correcta.

Se verificará que los valores de los parámetros se mantienen en el tiempo. Por ejemplo, no debería ocurrir que si se obtiene un valor de -0.00014534 para la posición de una articulación, y un segundo después se vuelve a comprobar dicho

valor, éste haya cambiado de manera considerable. Esto indicaría que algo estará fallando en el sistema.

La prueba consiste, por tanto, en comprobar uno a uno si todos los datos que están en el servidor son correctos y coinciden con aquello que pueden obtenerse desde un terminal usando una de las funcionalidades de ROS.

Esta prueba se ha realizado con el archivo .xml denominado *prueba1.xml*, cuyo contenido es el siguiente:

Figura 37. Contenido archivo prueba1.xml.

```
-<TFM>
  <Titulo> Topics ROS robot </Titulo>
 -<datos>
    <dato topic="/joint_states" variable="/joint_states/position.0" label="pos_art_0" tipo="double" freq="100"/>
    <dato topic="/joint_states" variable="/joint_states/position.1" label="pos_art_1" tipo="double" freq="100"/>
    <dato topic="/joint_states" variable="/joint_states/position.2" label="pos_art_2" tipo="double" freq="100"/>
    <dato topic="/joint_states" variable="/joint_states/position.3" label="pos_art_3" tipo="double" freq="100"/>
    <dato topic="/joint_states" variable="/joint_states/position.4" label="pos_art_4" tipo="double" freq="100"/>
    <dato topic="/joint_states" variable="/joint_states/position.5" label="pos_art_5" tipo="double" freq="100"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.0" label="vel_art_0" tipo="double" freq="200"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.1" label="vel_art_1" tipo="double" freq="200"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.2" label="vel_art_2" tipo="double" freq="200"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.3" label="vel_art_3" tipo="double" freq="200"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.4" label="vel_art_4" tipo="double" freq="200"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.5" label="vel_art_5" tipo="double" freq="200"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/positions.0" label="pos_deseada_art_0" freq="500" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/positions.1" label="pos_deseada_art_1" freq="500" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/positions.2" label="pos_deseada_art_2" freq="500" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/positions.3" label="pos_deseada_art_3" freq="500" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/positions.4" label="pos_deseada_art_4" freq="500" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/positions.5" label="pos_deseada_art_5" freq="500" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/velocities.0" label="vel_deseada_art_0" freq="1000" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/velocities.1" label="vel_deseada_art_1" freq="1000" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/velocities.2" label="vel_deseada_art_2" freq="1000" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/velocities.3" label="vel_deseada_art_3" freq="1000" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/velocities.4" label="vel_deseada_art_4" freq="1000" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/velocities.5" label="vel_deseada_art_5" freq="1000" tipo="double"/>
    <dato topic="/joint_states" variable="/joint_states/name.0" label="nombre_art_0" freq="2000" tipo="string"/>
    <dato topic="/joint_states" variable="/joint_states/name.1" label="nombre_art_1" tipo="string" freq="2000"/>
    <dato topic="/joint_states" variable="/joint_states/name.2" label="nombre_art_2" tipo="string" freq="2000"/>
    <dato topic="/joint_states" variable="/joint_states/name.3" label="nombre_art_3" tipo="string" freq="2000"/>
    <dato topic="/joint_states" variable="/joint_states/name.4" label="nombre_art_4" tipo="string" freq="2000"/>
    <dato topic="/joint_states" variable="/joint_states/name.5" label="nombre_art_5" tipo="string" freq="2000"/>
    <dato topic="/joint_states" variable="/joint_states/position.0" label="posicion_articulacion_0" freq="100" tipo="double"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.0" label="velocidad_articulacion_0" freq="100" tipo="double"/>
    <dato topic="/joint_states" variable="/joint_states/name.0" label="nombre_articulacion_0" freq="100" tipo="string"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/desired/velocities.4" label="velocidad_deseada_articulacion_4" freq="100" tipo="double"/>
    <dato topic="/arm_controller/state" variable="/arm_controller/state/error/velocities.4" label="error_velocidad_articulacion_4" freq="100" tipo="double"/>
  </datos>
</TFM>
```

La prueba se inicia poniendo en marcha tanto la simulación del ur_*robot* mediante la aplicación gazebo, como la aplicación del sistema desarrollado server. Como aplicación cliente se ha usado la aplicación '*Prosys OPC UA Client'* desde un sistema opertativo windows 10. Entonces:

```
$roslaunch ur_gazebo ur5.launch
```

En otro terminal:

```
$./server 192.168.0.102
```

Es opoturno recordar, que a no ser que se asigne una dirección IP fija al equipo esta cambiará cada vez que el sistema se reinicie. Sin embargo, es muy fácil comprobar que nueva IP se ha asignado mediante:

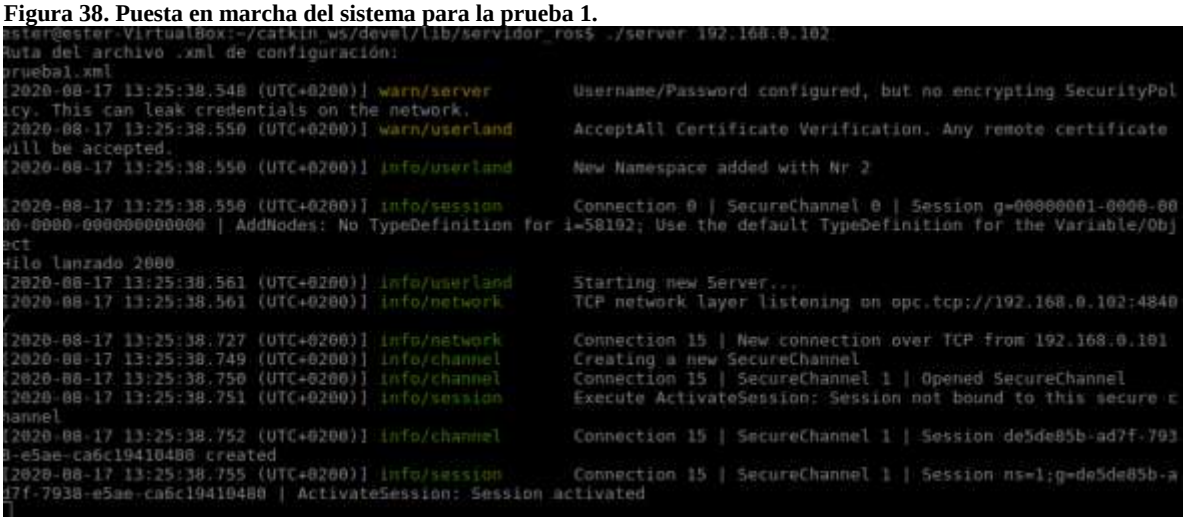
```
$ifconfig
```

Por lo que no se ve necesario realizar el procedimieno de asignar un IP fija al equipo.

Una vez puesto en marcha el sistema se notifica que todo se ha sucedido de manera correcta, y que el servidor se encuentra funcionando en la IP

192.168.0.102:4840, el puerto es el asignado por defecto, puesto que no se ha especificado uno a la hora de lanzar el servidor:

Figura 38. Puesta en marcha del sistema para la prueba 1.



Los valores reales del topic *'joint_states'* del robot se pueden observar desde un terminal con el comando:

```
$rostopic echo joint_states
```

Para terminar esta prueba se ha realizado el mismo procedimiento para el *topic 'arm_controller/status'* por lo que desde otro terminal se comprobarán sus valores reales:

```
$rostopic echo arm_controller/state
```

6.1.3 Prueba 3. *Robot UR* realizando una trayectoria.

El objetivo de esta prueba es analizar cómo se producen los cambios en los parámetros del robot que ofrece el servidor cuando este está realizando una trayectoria de una duración lo suficientemente larga como para permitir comprobar todas los parámetros, tanto los reales que son aquellos que se obtienen con la funicionalidad de ROS, como aquellos con los que se verificará su validez, que son los que ofrece el servidor. La trayectoria se planifica con '*rviz'.*

Debido a algunos problemas a la hora de iniciar la aplicación *'rviz'* para planificar trayectorias se ha modificado la forma de abrirla. Se ha usado, por tanto, para cargar el *UR robot* en *gazebo* como su planificador de trayectorias, el siguiente procedimiento:

1. En un terminal:

```
$roslaunch ur_gazebo ur5.launch limited:=true
```

2. En otro terminal:

```
$ roslaunch ur5_moveit_config
ur5_moveit_planning_execution.launch sim:=true limited:=true
```

3. Y por último, en un nuevo terminal distinto:

```
$ roslaunch ur5_moveit_config moveit_rviz.launch config:=true
```

Para poder realizar esta prueba es necesario realizar una trayectoria para el robot. La trayectoria no será muy larga, pero lo suficiente para que permita detectar errores en caso de existirlos o por el contrario verificar el correcto funcionamiento del sistema.

La trayectoria programada no es excesivamente larga en el tiempo, pues apenas dura 120 segundos, pero es el tiempo suficiente para poder realizar un seguimiento de todos los valores y comprobar que los resultados son correctos.

Con esta prueba se pretende concluir, en caso de éxito, el apartado de pruebas teóricas, y verificar que este sistema sirve para usarlo en simulaciones..

En primer lugar, se incluyen una serie de imágenes que ilustran la trayectoria que va a seguir el brazo robot:

Figura 39. Trayectoria prueba 2 parte I.

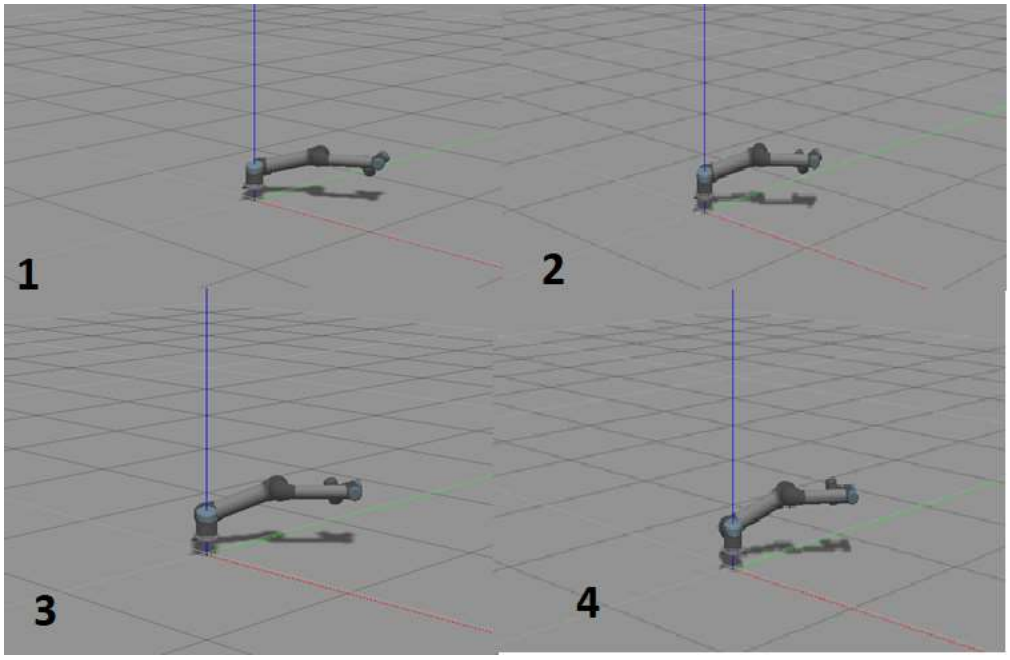


Figura 40. Trayectoria prueba 2 parte II.

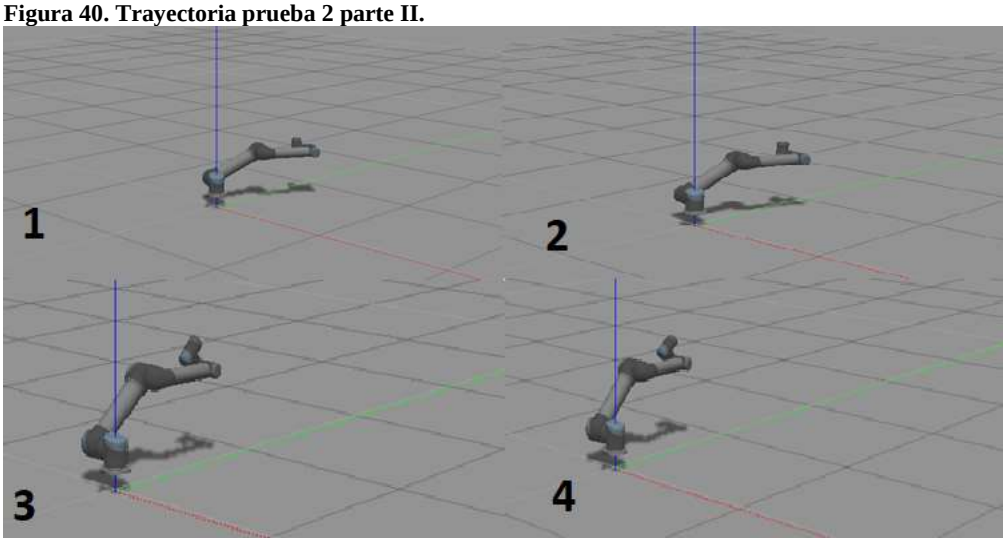
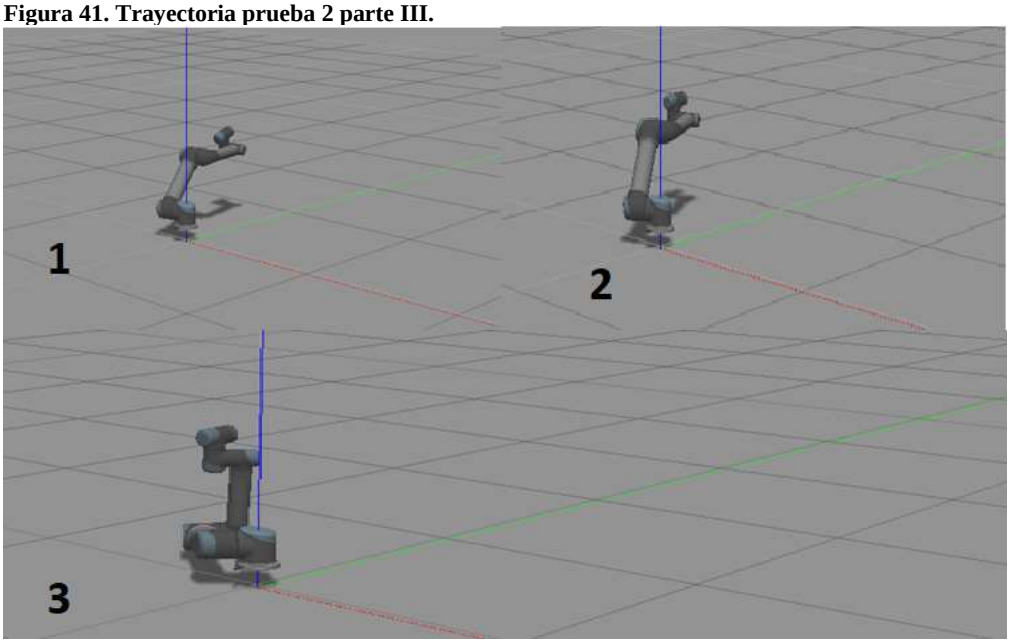


Figura 41. Trayectoria prueba 2 parte III.



Como *topic* de referencia se va a tomar el valor de la posición de la articulación 0.

6.2 PRUEBAS PRÁCTICAS.

Tal y como se ha mencionado previamente, es necesario realizar una prueba práctica que permita verificar el sistema para este tipo de situaciones. Se ha seguido el mismo procedimiento que durante la realización de las pruebas teóricas. Se han planificado dos pruebas, una donde el robot está parado y se comprueban sus valores reales con aquellos que ofrece el servidor. Y otra donde el robot realiza una trayectoria, o una tarea, donde se comprueba si durante todo el transcurso de la trayectoria los valores reales y los del servidor coinciden y si estos último tienen sentido.

6.2.1 Prueba 4. Robot real en parada.

El objetivo de esta prueba es monitorizar los parámetros de un brazo robótico real mediante el sistema diseñado y comprobar si los datos que se están ofreciendo en el servidor coinciden con los parámetros reales del robot mientras este está parado. Para poder llevar a cabo esta prueba es necesario realizar primero un archivo de configuración .xml, por lo que también se mostrarán los resultados de usar la interfaz gráfica con un robot real.

El robot escogido para realizar la prueba real es el 'Robonik', que se encuentra en el laboratorio de robótica del edificio Ai2 de la UPV. Se trata de un robot que sirve de base para un brazo robótico.

Figura 42. Robot RB1 del laborario de robótica del ai2 de la UPV.

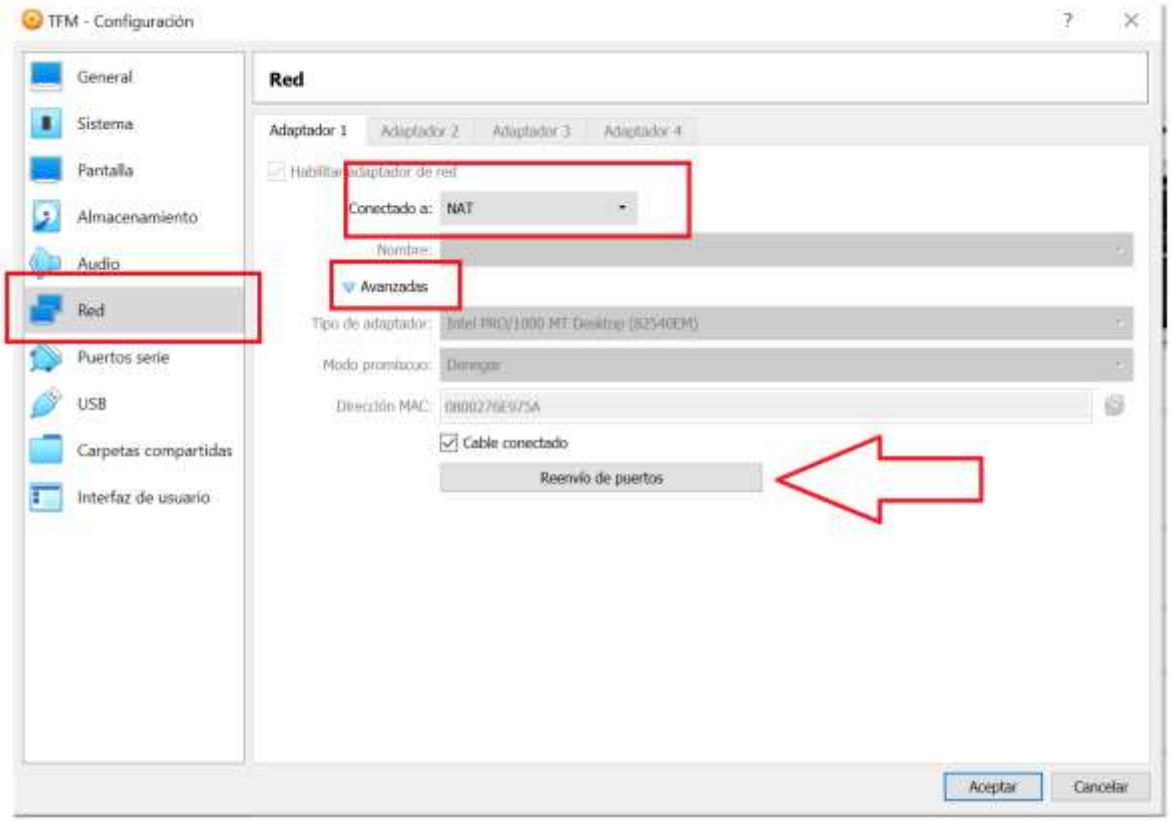
Para tener una mejor visión de los resultados obtenidos se ha realizado una interfaz gráfica mediante la aplicación de Ignition. Esta interfaz muestra los valores de los *topics* escogidos usando los elementos gráficos de Ignition y actualizando dichos valores en tiempo real según los datos que se van recogiendo del servidor. En el momento de mostrar los resultados se presentará también una visión de la aplación de Ignitnion.

Anteriormente, se realizó la conexión entre la máquina virtual y el ordenador anfitrión cambiando el tipo de red de la máquina virtual a 'Adaptador de puente'. Sin embargo, ahora para poder conectar el robot a la máquina virtual es necesario que la configuración de red sea NAT (*Network Adress Translation*). Es decir, en lugar de que la máquina virtual tenga una dirección IP distinta al anfitrión y de la misma red en la que se encuentra este, tendrá una dirección IP que será privada y asignada por el propio *VirtualBox* que actúa como servidor DHCP.

Para poder seguir realizando la conexión es necesario realizar una redirección de puertos. Para ello, es necesario entrar en la configuración de la máquina virtual en la sección de red, cambiar la opción '*conetado a:'* a la opción

NAT. A continuación seleccionar la opciones avanzadas y pulsar sobre la opción ‘Reenvío de puerto’.

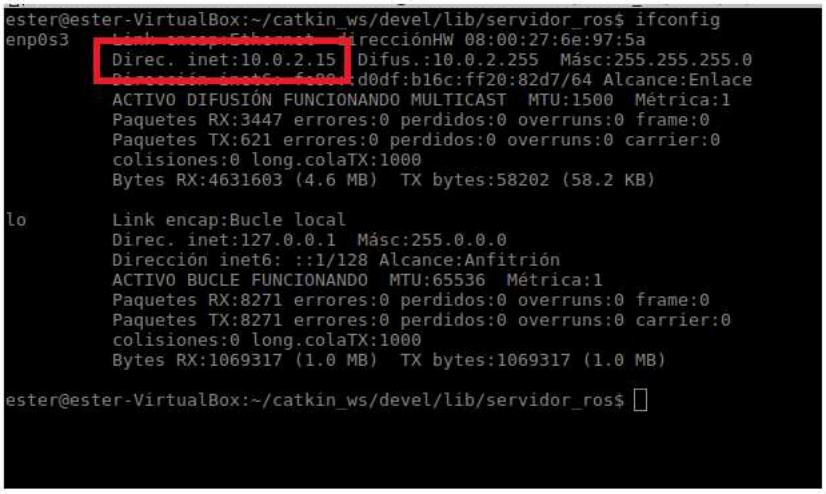
Figura 43. Prueba 5. Configuración de la red NAT de la máquina virtual.



Dentro de esta opción es necesario conocer tanto la IP privada de la máquina virtual, que por lo general siempre suele ser 10.0.2.15 y la IP de la máquina anfitrión, que es necesario actualizar cada vez que se conecte a una red nueva, puesto que no se dispone de una IP fija. Para no tener que estar creando una regla nueva, en lugar de poner la IP de la máquina anfitriona, se puede crear redireccionar al localhost, (es decir, 127.0.0.1) de manera que no sea necesario actualizar la regla cada vez que se conecte la máquina.

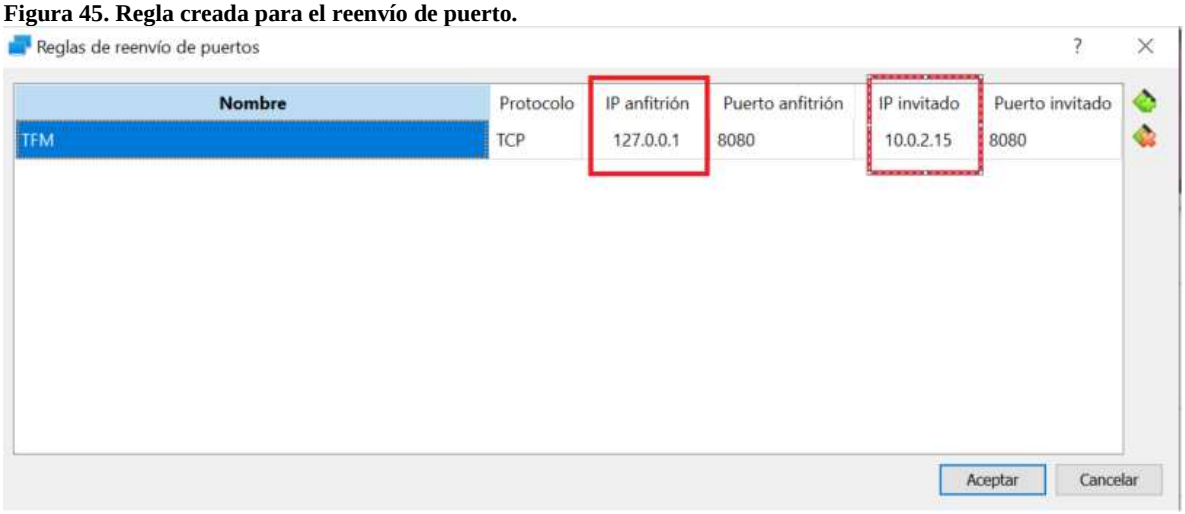
Para comprobar la dirección IP en el caso de la máquina virtual cuyo sistema operativo es Ubuntu 16.04, se realiza con el comando ‘ifconfig’ desde la consola.

Figura 44. Obtener dirección IP de la máquina virtual.



Una vez que se tienen estos datos, ya es posible configurar la redirección de puertos creando una regla de la siguiente manera:

Figura 45. Regla creada para el reenvío de puerto.



6.2.2 Prueba 5. Robot real realizando una trayectoria.

Esta prueba se ha realizado con el mismo robot que la prueba anterior y el mismo archivo de configuración y consiste en monitorizar los parámetros del robot y comprobar si coinciden con los reales mientras éste está realizando una trayectoria lo suficientemente larga en el tiempo para poder comprobar que todos los valores iguales coinciden en un instante de tiempo aleatorio.

No se describe la trayectoria seguida puesto que el robot se controla mediante un mando a distancia y simplemente se han realizado movimientos aleatorios durante el tiempo suficiente como para capturar todos los datos.

6.3 PRUEBA DE RENDIMIENTO.

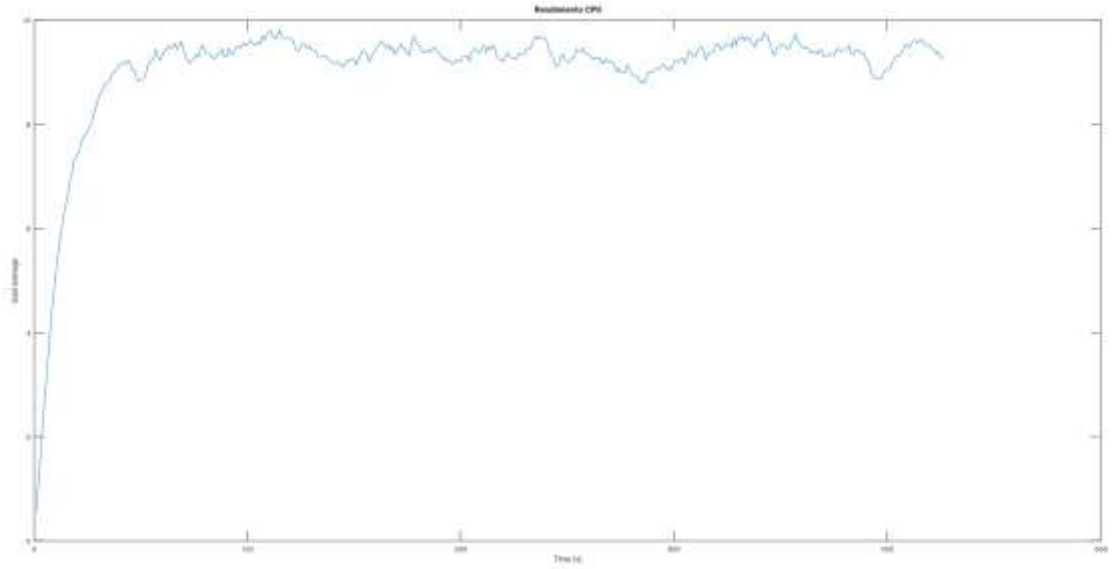
6.3.1 Prueba 6. Prueba de rendimiento.

Se va a proceder a mostrar una última prueba donde se ha monitorizado el consumo de CPU que tiene la aplicación en un periodo de funcionamiento relativamente largo. Esta última prueba se ha realizado de dos manera distintas,

primero, se pone en marcha el servidor junto al simulador gazebo, para que se tengan datos que almacenar y segundo sin el simulador gazebo. Esta decisión se ha tomado por el hecho de que el simulador gazebo tiene un gran consumo de recursos de la CPU y no se quiere por tanto que esto contamine los resultados de las pruebas.

La duración de esta primera prueba es de unos 400 minutos (casi 7 horas completas).

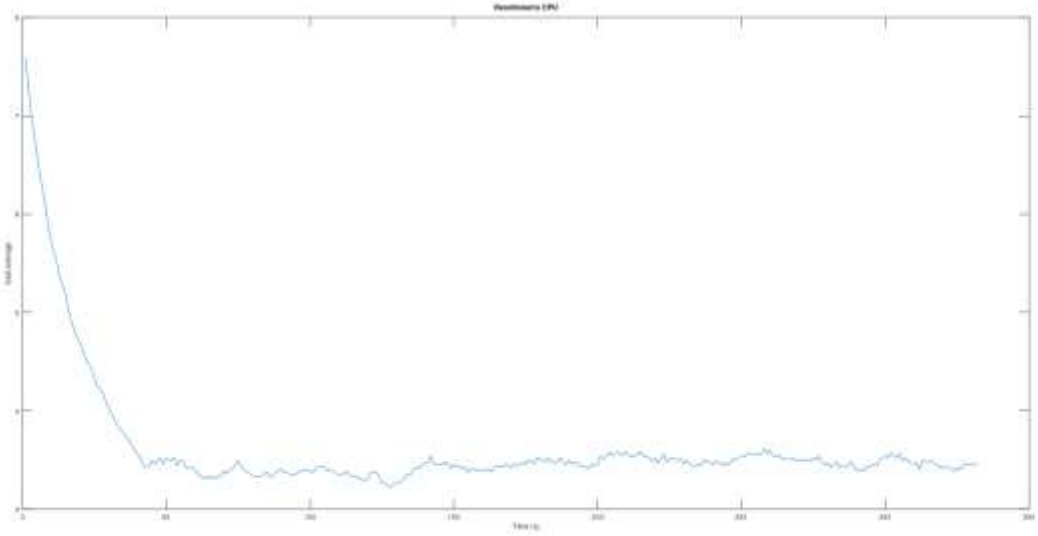
Figura 46. Prueba de rendimiento con simulador gazebo.



Se puede observar que el consumo de CPU ha sido bastante alto, de casi un 10%, y con una subida prácticamente exponencial.

En esta segunda prueba la duración ha sido de 350 minutos (casi 6 horas completas).

Figura 47. Prueba de rendimiento sin simulador gazebo.



Se observa ahora que el consumo se ha reducido de manera considerable, es importante destacar que las pruebas se hicieron una a continuación de la otra, de ahí que el consumo inicial de esta prueba sea tan alto, y luego baje de una

manera logarítmica. El consumo cuando sólo está encendido el servidor oscila entre un 3 y 4%, es bastante bajo, por lo que se puede concluir que hay un buen rendimiento y que en ningún momento la aplicación abusa de los recursos de la CPU.

7. RESULTADOS. ANÁLISIS Y DISCUSION DE RESULTADOS

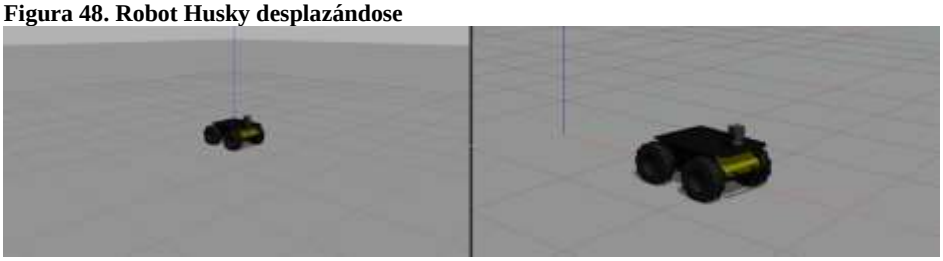
7.1 RESULTADOS INTERMEDIOS

7.1.1 Resultados. Programa en ROS para publicar una velocidad en un *topic* asociado a un robot y leer su odometría de otro *topic.*

Se muestra el primer resultado tangible del proyecto, mostrando como el robot husky se desplaza de una posición a otra gracias a que se está publicando una velocidad determinada en el *topic* asociado a la misma. También se muestra la lectura de la odometría de este robot a través del *topic* asociado a la misma.

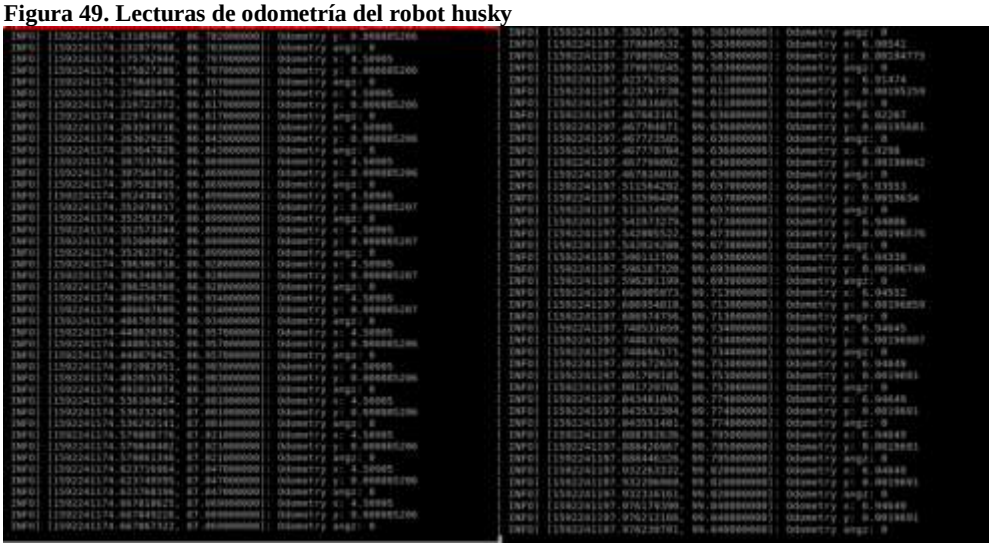
En la figura siguiente se observa cómo el robot se ha desplazado de una posición inicial a otra.

Figura 48. Robot Husky desplazándose



En la siguiente figura se muestra el terminal, donde se puede ver cómo su odometría va cambiando ya que el robot está en movimiento.

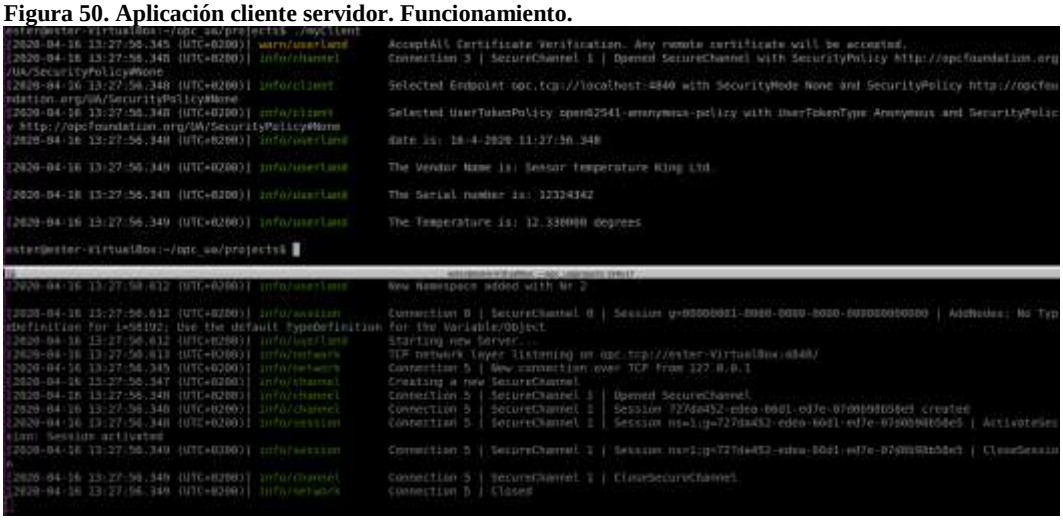
Figura 49. Lecturas de odometría del robot husky



7.1.2 Resultados. Aplicación cliente servidor con OPC-UA.

En la figura se muestran un terminal en está ejecutanto el servidor y otro en el que se ejecuta el cliente.

Figura 50. Aplicación cliente servidor. Funcionamiento.



Se puede comprobar que cada vez que el cliente realiza una petición el servidor devuelve los datos solicitados y el cliente los imprime, así también se indica en el terminal del servidor las conexiones que se relizan con cada cliente que lo solicita.

7.1.3 Resultados. Aplicación cliente servidor OPC-UA en módulo ROS.

La principal diferencia con los resultados del apartado anterior es que aquí ya no se van a fabricar los datos que ofrecerá el servidor, sino que se van a almacenar ya datos provenientes del entorno ROS, por lo que básicamente se pretende conseguir una combinación de los resultados de los dos últimos apartados.

Figura 51. Inicio del servidor OPC-UA.

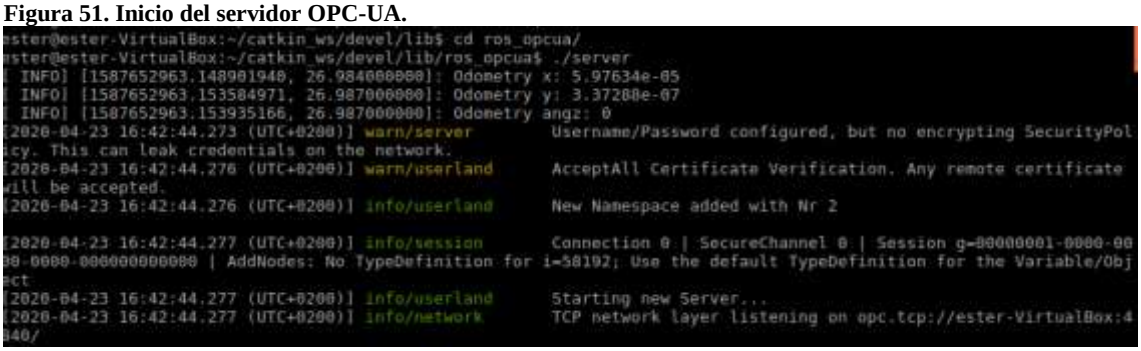


Figura 52. Prueba de funcionamiento antes de poner en marcha el programa que cambia la velocidad del robot.

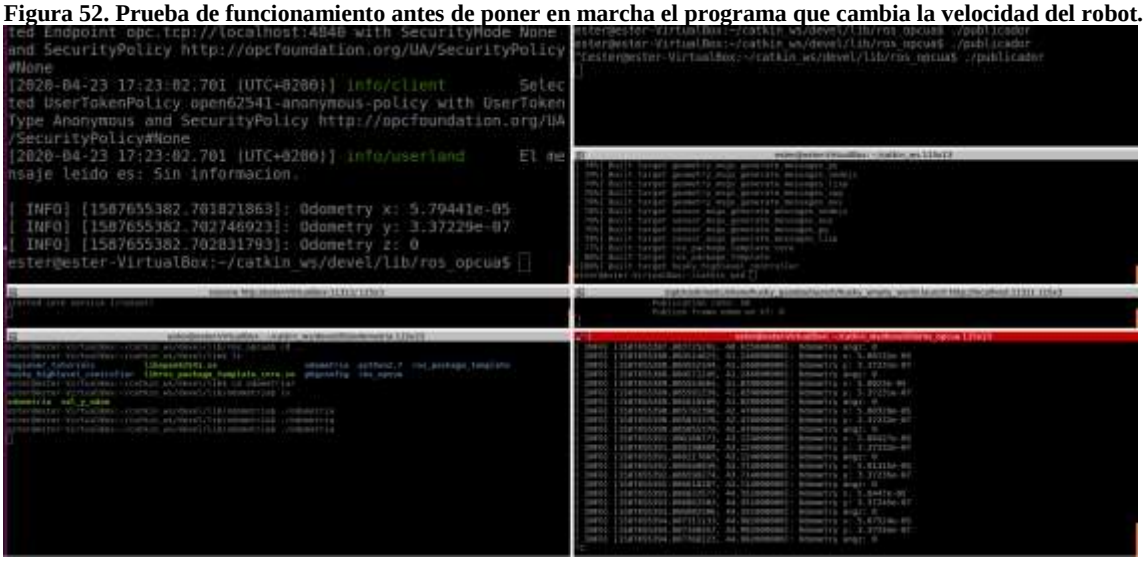


Figura 53. Después de poner en marcha el programa que cambia la velocidad del robot.

Cuando la frecuencia del publicador no es suficientemente alta es posible que, aunque existe una suscripción al topic *'chatter'* no se muestre el mensaje que se está recibiendo, como puede observarse en la Figura 52. Prueba de funcionamiento antes de poner en marcha el programa que cambia la velocidad del robot.Figura 52. Para resolver este problema es suficiente con aumentar la

frecuencia del publicador, como se puede observar en Figura 53 en la que se visualiza perfectamente el mensaje de *'Prueba TFM'*.

Como se puede comprobar ya se es capaz de ofrecer datos provenientes del entorno ROS mediante el protocolo de comunicación OPC-UA.

7.1.4 Resultados. Configuración del servidor a través de un archivo .xml.

Se visualiza el resultado desde el cliente *Prosys*. Como se puede comprobar todas las variables se han configurado de manera correcta de acuerdo a los parámetros indicados.

Figura 54. Resultado del servidor configurado con el archivo xml I.

Figura 55. Resultado del servidor configurado con el archivo xml II.

Figura 56. Resultado del servidor configurado con el archivo xml III.

Con este resultado de muestra como el proyecto está empezando a tener funcionalidad y flexibilidad al proyecto al poder seleccionar los datos que se quieren almacenar en el servidor sin necesidad de reprogramar el código cada vez que estos datos cambien.

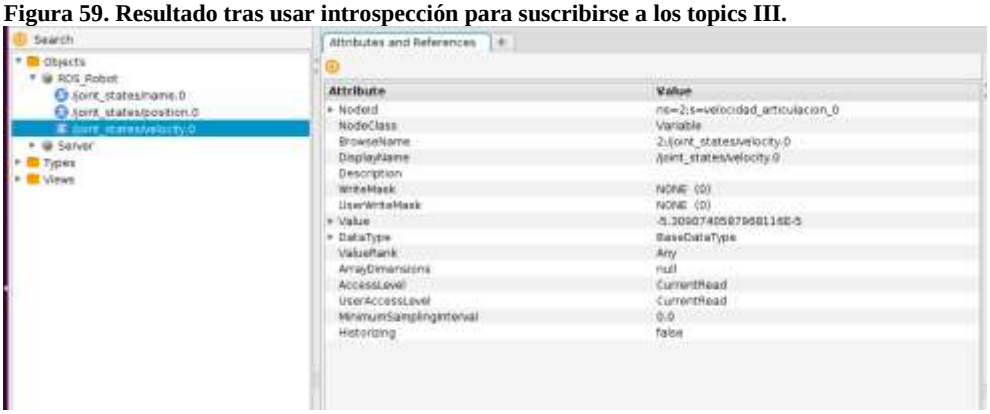
7.1.5 Resultados. Configuración del servidor a través de un archivo .xml con introspección.

Se presentan los resultados ahora tras añadir la introspección, similares a los anteriores, salvo por un detalle bastante importante que se comentará al final.

Figura 57. Resultado tras usar introspección para suscribirse a los topics I.

Figura 58. Resultado tras usar introspección para suscribirse a los topics II.

Figura 59. Resultado tras usar introspección para suscribirse a los topics III.

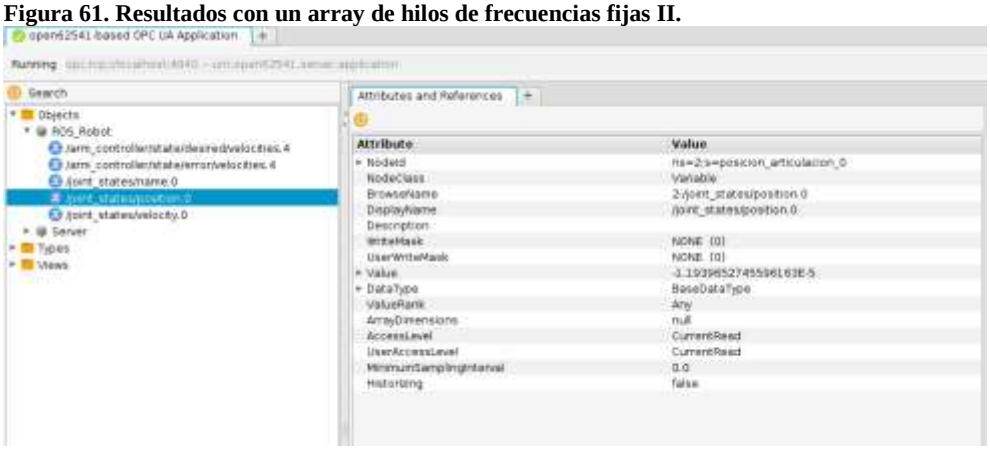


Como se puede comprobar se obtiene la información de cualquier *topic* que se demande en el fichero de configuración.

Tras realizar las modificaciones pertinentes para los hilos de refresco se obtiene:

Figura 60. Resultados con un array de hilos de frecuencias fijas I.

Figura 61. Resultados con un array de hilos de frecuencias fijas II.



Esta nueva filosofía respecto a los hilos de refresco es más compacta en cuanto a código, puesto que en lugar usar tres hilos distinos se usa un array de cinco posiciones, cada una de las posiciones guarda una frecuencia de refresco distinta. Además, al ser cinco en lugar de tres ofrece más opciones para refrescar cada dato con la frecuencia más apropiada.

Añadir la introspección al proyecto facilita de manera considerable el poder usar el proyecto con cualquier robot, ya con los resultados del apartado anterior se empezaba a otorgar flexibilidad, pero ahora la flexibilidad es completa, puesto que no es necesario reprogramar en ningún momento, simplemente con un archivo de configuración .xml cuya sintaxis sea la correcta se podrán seleccionar todos los datos que el servidor va a almacenar.

7.2 RESULTADOS FINALES.

En este apartado se recogen los resultados finales que son aquellos correspondientes al apartado de PRUEBAS.

7.2.1 Resultados. Pruebas teóricas.

Exceptuando la primera que sus resultados son necesarios para ir enlazando la explicación se van a presentar los resultados correspondientes al apartado de pruebas teóricas.

En primer lugar, se presentan los resultados correspondientes a la prueba usando el simulador gazebo cuando el robot estaba en parada.

Figura 62. Valores del topic *'joint_states'*.

```
name: [elbow_joint, shoulder_lift_joint, shoulder_pan_joint, wrist_1_joint, wrist_2_joint,
  wrist_3_joint]
position: [-1.1940345206795655e-05, 0.003496114816327278, -0.0011960133016630792, 0.0001968263444638298 3, 1.0784301
06788105e-05, 1.0243317772662408e-05]
velocity: [-7.460462038053218e-05, 0.012453101904703015, 0.0032145055290524922, 0.0005535954492046982, -6.959471357
717291e-05, 2.6816155900656858e-05]
effort: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Figura 63. Resultados de la prueba 2 de los valores numéricos del topic *'joint_states'*.

Attribute	Value
NodeId	ns=2;s=velocidad_articulacion_0
NodeClass	Variable
BrowseName	2:/joint_states/velocity.0
DisplayName	/joint_states/velocity.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	-7.468462030877851E-5
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Attribute	Value
NodeId	ns=2;s=pos_art_0
NodeClass	Variable
BrowseName	2:/joint_states/position.0
DisplayName	/joint_states/position.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	-1.1940345205907477E-5
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Attribute	Value
NodeId	ns=2;s=vel_art_4
NodeClass	Variable
BrowseName	2:/joint_states/velocity.4
DisplayName	/joint_states/velocity.4
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	-6.959471357719969E-5
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Attribute	Value
NodeId	ns=2;s=pos_art_5
NodeClass	Variable
BrowseName	2:/joint_states/position.5
DisplayName	/joint_states/position.5
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	1.0243317772662408E-5
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Se comprueba que los resultados son correctos. Sin embargo, si se realiza la misma prueba con este topic, pero con aquellos valores que son de tipo *'string'* se obtiene el siguiente resultado:

Figura 64. Resultados de la prueba 2 con los valores de tipo *'string'* del topic *'joint_states'*.

Attribute	Value
NodeId	ns=2;s=nombre_articulacion_0
NodeClass	Variable
BrowseName	2:/joint_states/name.0
DisplayName	/joint_states/name.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	wrist_3_joi
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Attribute	Value
NodeId	ns=2;s=nombre_art_1
NodeClass	Variable
BrowseName	2:/joint_states/name.1
DisplayName	/joint_states/name.1
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	□□□ □□□□
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Se observa que no sólo el nombre es incorrecto, puesto que la articulación 0, su nombre es *elbow_joint*, sino que, además, en algunos casos no se leen ni caracteres, es decir, el valor del nombre está corrupto. Esta prueba indica que hay un error en el código. La manera de guardar el valor en la variabla *ROS_datos* no es la correcta. Tras una pequeña investigación se concluye que la manera adecuada de guardar datos en una varibale de tipo *UA_string* desde otra variable del mismo tipo no es la asignación como se había utilizado previamente, si no que es encesario usar una función propia de estas variables denominada *'UA_String_copy'*, por lo que el código de esa parte se modifica de la siguiente manera:

```
if (!strcmp(key.c_str(), ROS_datos[i].variable)) {
	UA_String s;
	UA_String_init(&s); // _init zeroes out the entire memory of the datatype
	char *test = (char *)value.c_str();
	s.length = strlen(test);
	s.data = (UA_Byte*)test;
	pthread_mutex_lock(&ROS_datos[i].mutex);
	UA_String_copy(&s, &ROS_datos[i].valor_string);
	pthread_mutex_unlock(&ROS_datos[i].mutex);
```

Con este nuevo código se obtiene:

Figura 65. Resultados de la prueba 1 de algunos topics con valor de tipo '*string*' con código modificado.

Attribute	Value
NodeId	ns=2;s=nombre_articulacion_0
NodeClass	Variable
BrowseName	2:/joint_states/name.0
DisplayName	/joint_states/name.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	elbow_joint
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Attribute	Value
NodeId	ns=2;s=nombre_art_1
NodeClass	Variable
BrowseName	2:/joint_states/name.1
DisplayName	/joint_states/name.1
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	shoulder_lift_joint
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Estos valores ya son los correctos. Para comprobar que no ha sido por casualidad, se accede al mismo topic *'joint_states/name.0'* en tiempos distintos y se comprueba si se siguen manteniendo los mismo resultados.

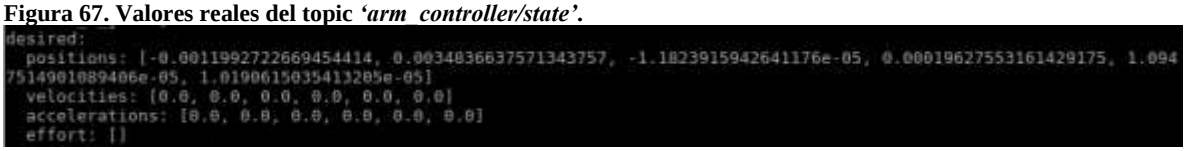
Figura 66. Resultados de la prueba 1 de algunos topics con valor de tipo '*string*' con código modificado.

Attribute	Value
NodeId	ns=2;s=nombre_art_0
NodeClass	Variable
BrowseName	2:/joint_states/name.0
DisplayName	/joint_states/name.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	elbow_joint
StatusCode	GOOD (0x00000000)
ServerTimestamp	08/17/20 11:21:58.8312670 GMT
SourceTimestamp	08/17/20 11:21:58.8386290 GMT
ServerPicoseconds	0
SourcePicoseconds	0
Value	elbow_joint
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Attribute	Value
NodeId	ns=2;s=nombre_art_0
NodeClass	Variable
BrowseName	2:/joint_states/name.0
DisplayName	/joint_states/name.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	elbow_joint
StatusCode	GOOD (0x00000000)
ServerTimestamp	08/17/20 11:21:58.8312670 GMT
SourceTimestamp	08/17/20 11:21:58.8386290 GMT
ServerPicoseconds	0
SourcePicoseconds	0
Value	elbow_joint
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Para el *topic arm_controller/state.*

Figura 67. Valores reales del topic *'arm_controller/state'*.



```
desired:
  positions: [-0.0011992722669454414, 0.0034836637571343757, -1.1823915942641176e-05, 0.00019627553161429175, 1.094
7514901089406e-05, 1.0190615035413205e-05]
  velocities: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  accelerations: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  effort: []
```

Y realizando las mismas pruebas que en el *topic* anterior se obtiene:

Figura 68. Resultados de la prueba 2 para el topic *'arm_controller/state'*.

Attribute	Value
▸ NodeId	ns=2;s=pos_deseada_art_0
NodeClass	Variable
BrowseName	2:/arm_controller/state/desired/p...
DisplayName	/arm_controller/state/desired/pos...
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
▸ Value	-0.0011992722669454414
▸ DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Attribute	Value
▸ NodeId	ns=2;s=vel_deseada_art_3
NodeClass	Variable
BrowseName	2:/arm_controller/state/desired/ve...
DisplayName	/arm_controller/state/desired/velo...
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
▸ Value	0.0
▸ DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Los resultados teóricos y prácticos coinciden, por tanto se concluye que la segunda prueba ha sido realizada con éxito y, además, ha permitido corregir un error que hasta el momento no se había notificado.

Siguiendo esta misma filosofía se presentan ahora los resultados de la prueba 3 que versa sobre el robot simulado realizando una trayectoria aleatoria.

En la Figura 69 se muestra el valor real y el valor que recoge el servidor. Se puede comprobar que incialmente es el mismo valor.

Figura 69. Valor de la posición de la articulació 0 antes de insertar la trayectoria, prueba 3.

Attribute	Value
▸ NodeId	ns=2;s=pos_art_0
NodeClass	Variable
BrowseName	2:/joint_states/position.0
DisplayName	/joint_states/position.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
▸ Value	-2.8370596505844503E-5
▸ DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Las siguientes imágenes son capturas de pantalla realizadas durante la trayectoria del robot, de manera que se puede comprobar el valor real, y el valor que ofrece el servidor. No se ha incluido una captura de todos los valores puesto que al ser demasiados los que ofrece el servidor esto sería inabordable, y carece de interés. Se han eligido distintos valores provenientes de distintos '*topics'* en distintos instantes de manera aleatoria.

Figura 70. Resultados prueba 3, posición articulación 0.

NodeId	ns=2;s=posicion_articulacion_0
NodeClass	Variable
BrowseName	2:/joint_states/position.0
DisplayName	/joint_states/position.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	-1.3141334390683177
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

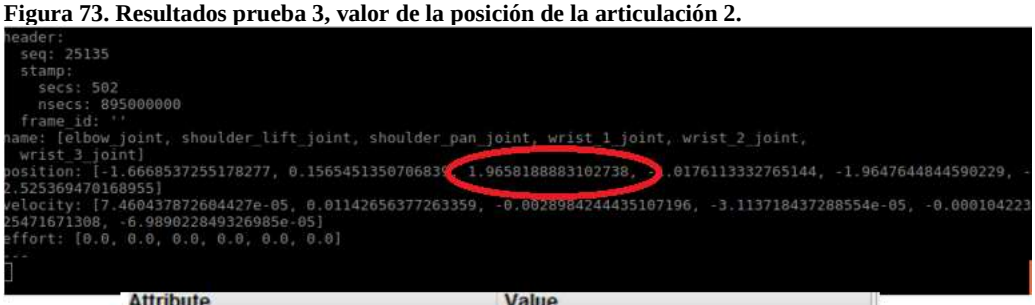
Se puede comprobar que exite un pequeña diferencia en las centésimas entre el valor real y el que se encuentra en el servidor. Este error es explicable por desfase temporal entre que se toman las capturas de un valor y otro, ocurre lo mismo en las siguientes capturas.

Figura 71. Resultados prueba 3, error de velocidad de la articulación 4.

NodeId	ns=2;s=error_velocidad_articulacion...
NodeClass	Variable
BrowseName	2:/arm_controller/state/error/velocitie...
DisplayName	/arm_controller/state/error/velocities.4
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	-0.0245940488305539
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

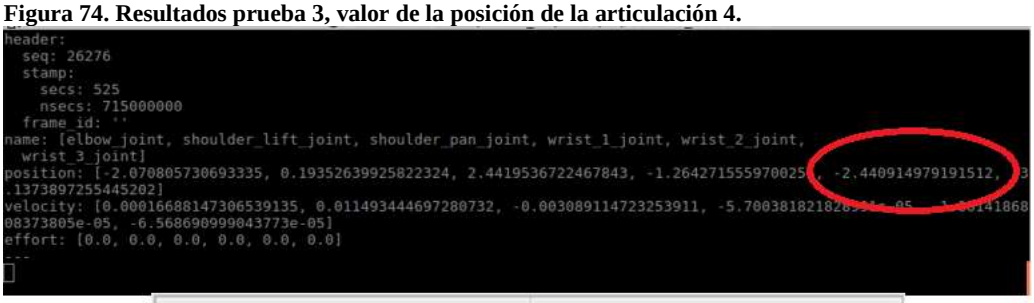
Figura 72. Resultados prueba 3, valor de la velocidad deseada en la articulación 4.

Figura 73. Resultados prueba 3, valor de la posición de la articulación 2.



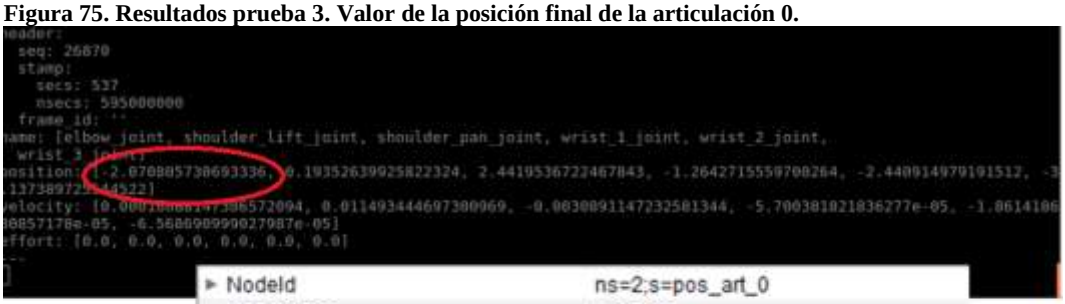
Attribute	Value
NodeId	ns=2;s=pos_art_2
NodeClass	Variable
BrowseName	2:/joint_states/position.2
DisplayName	/joint_states/position.2
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
Value	1.9369405496983205
DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Figura 74. Resultados prueba 3, valor de la posición de la articulación 4.



Attribute	Value
▸ NodeId	ns=2;s=pos_art_4
NodeClass	Variable
BrowseName	2:/joint_states/position.4
DisplayName	/joint_states/position.4
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
▸ Value	-2.440914979191513
▸ DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Figura 75. Resultados prueba 3. Valor de la posición final de la articulación 0.



▸ NodeId	ns=2;s=pos_art_0
NodeClass	Variable
BrowseName	2:/joint_states/position.0
DisplayName	/joint_states/position.0
Description	
WriteMask	NONE (0)
UserWriteMask	NONE (0)
▸ Value	-2.070805730693335
▸ DataType	BaseDataType
ValueRank	Any
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	0.0
Historizing	false

Tras examinar los resultados obtenidos y comprobar que todos son válidos y coinciden con los resultados teóricos obtenidos, se concluye con éxito la prueba teórica número tres. Esto desmuestra la validez del sistema para obtener la información paramétrica de un robot cuando éste se encuentra realizando una trayectoria aleatoria en un entorno de simulación dentro de ROS.

7.2.2 Resultados. Pruebas prácticas.

Por último se presentan los resultados tras utilizar la aplicación con un robot real. Probablemente estos resutados son los de mayor importancia porque gracias a ellos se puede inferir si se puede utilizar este desarrollo para entornos reales o no.

Se muestran ahora los resultados mediante capturas de la interfaz estando el robot en parada:

Figura 76. Prueba 4. Interfaz gráfica I.

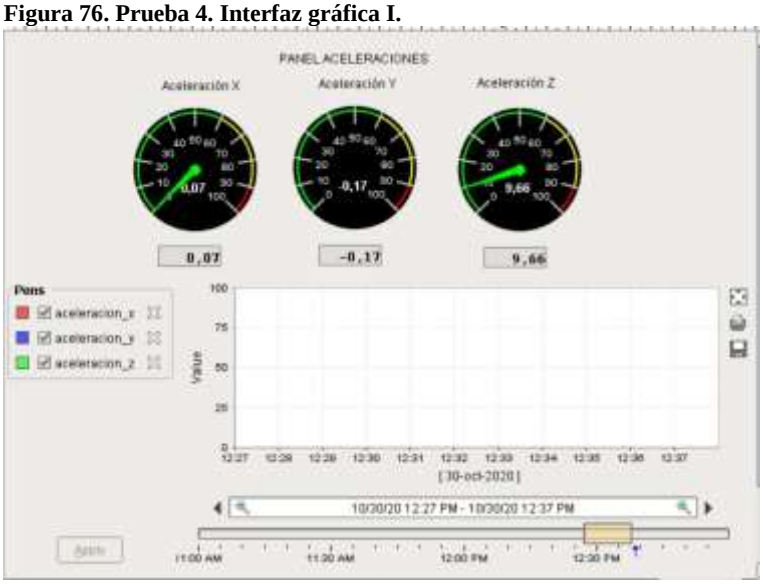


Figura 77. Prueba 4. Interfaz gráfica II.

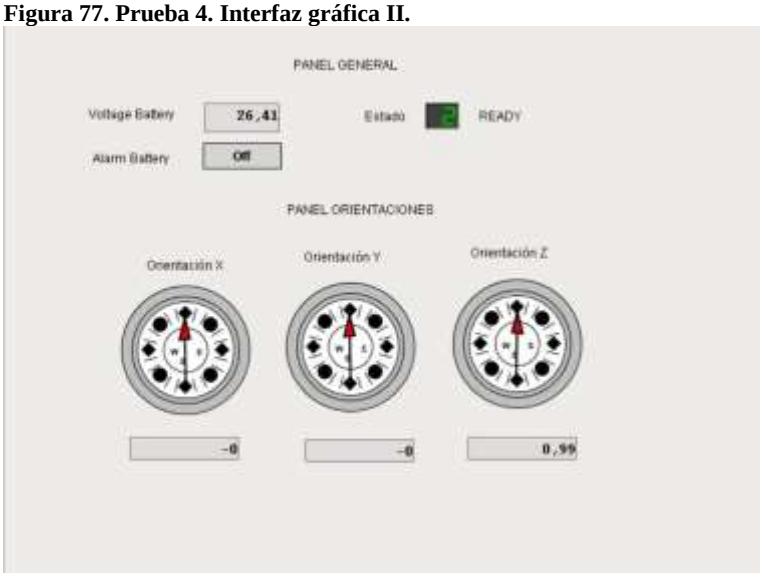


Figura 78. Prueba 4. Interfaz gráfica III.

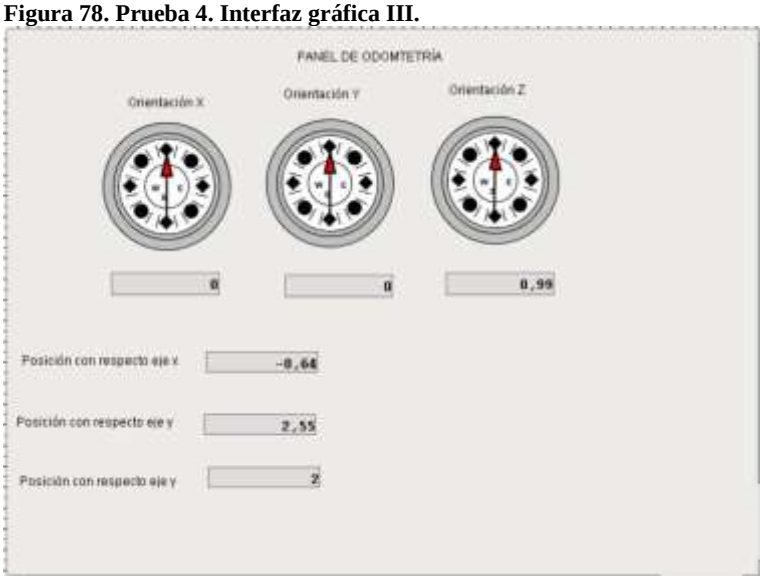


Figura 79. Prueba 4. Interfaz gráfica IV.

Los resultados son acordes a lo esperado, puesto que el robot, al estar en parada, lo lógico es que las velocidades sean cero, las aceleraciones en los ejes x e y próximas a cero, mientras que la velocidad en el eje z debe tener un valor similar a la constante de gravedad 9.8. El resto de resultados también son los esperados.

Por último se muestran obtenidos mientras el robot realizaba una trayectoria breve por la sala donde se encontraba.

Para no saturar este documento con capturas de pantalla en cada momento de la trayectoria, se van a presentar los resultados de dos momento distintos. Se ha recogido en un vídeo la trayectoria completa.

Figura 80. Prueba 5. Interfaz gráfica I.

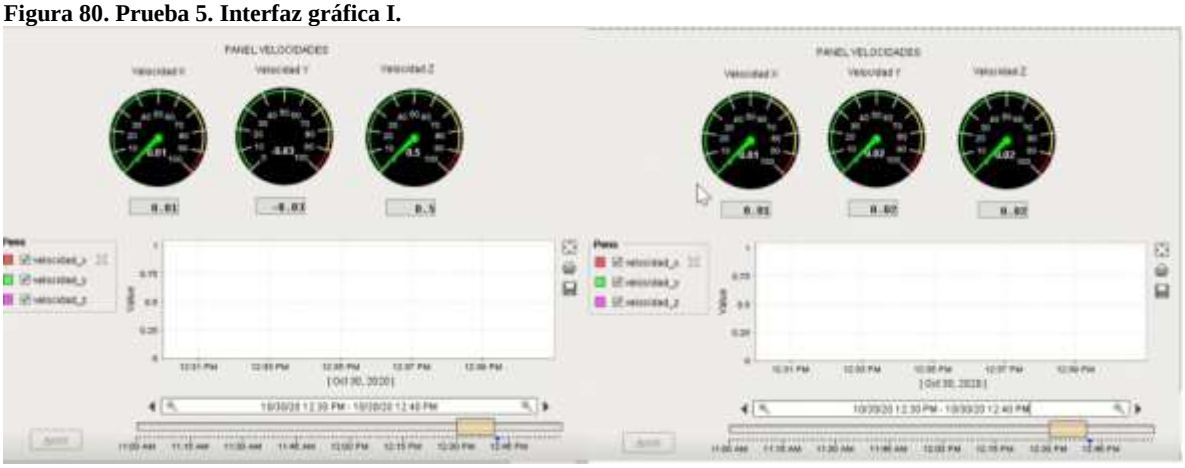


Figura 81. Prueba 5. Interfaz gráfica II.

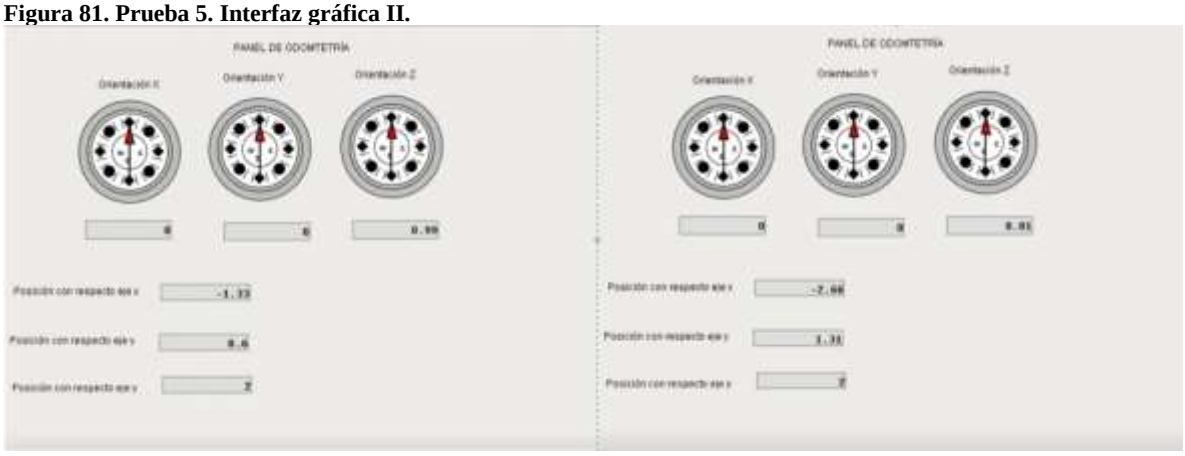


Figura 82. Prueba 5. Interfaz gráfica III.

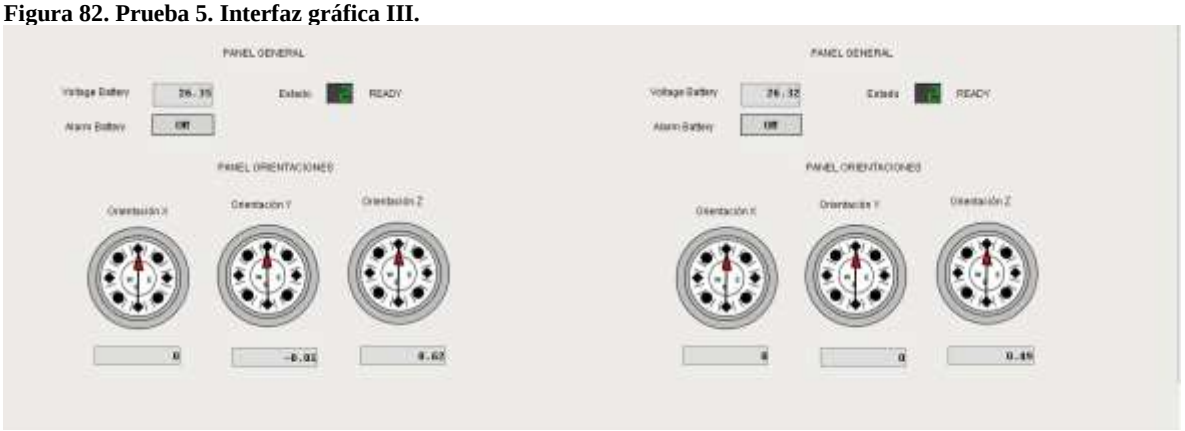
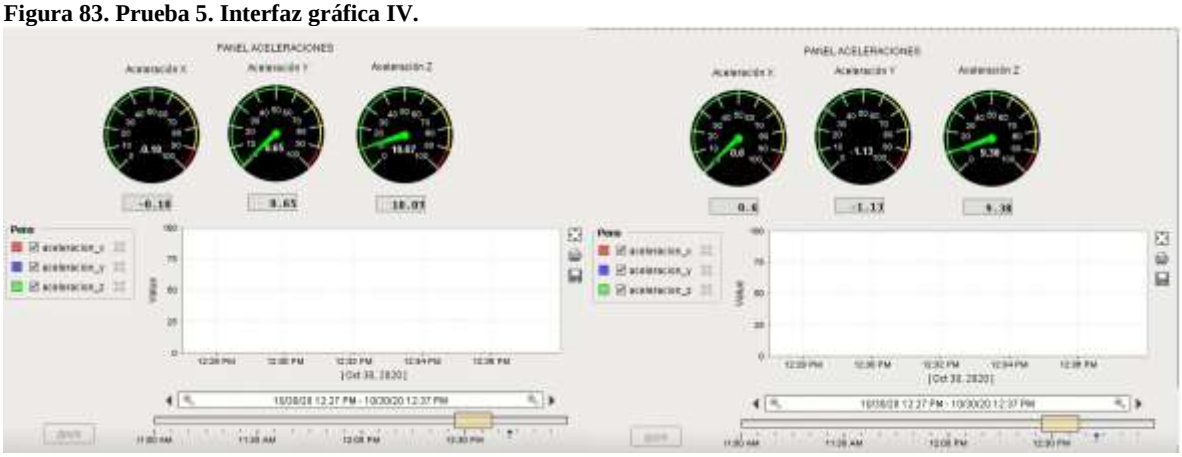


Figura 83. Prueba 5. Interfaz gráfica IV.



Los resultados son acordes a los esperados, por tanto se concluye el correcto funcionamiento del sistema para monitorizar los topics de un robot mediante el sistema operativo ROS y el protocolo OPC-UA.

8. CONCLUSIONES

La realización de este trabajo ha permitido desarrollar un proyecto en el entorno de desarrollo robótico ROS que ha incorporado el protocolo de comunicación OPC-UA. Este desarrollo es bastante útil para todos aquellos entornos industriales donde existan robots colaborativos juntos con otros dispositivos industriales (ej PLCs), que tradicionalmente mantenían comunicación con el resto del entorno con OPC-UA, entre otros protocolos. Se resuelve por tanto, el problema del aislamiento de los robots colaborativos, ya que, a priori, estos no podrían comunicarse con el resto de dispositivos con el procolo OPC-UA.

Puede concluirse, por tanto, que se ha conseguido cumplir el objetivo principal del proyecto: Desarrollo y despliegue de un módulo OPC-UA en ROS para integrar robots colaborativos en entornos industriales. Es decir, desarrollar un sistema que permita obtener la información de los *topics* de ROS y compartirla mediante el estándar OPC-UA con otros dispositivos que estén fuera de ROS.

Respecto a los objetivos específicos que se plantearon para este trabajo, se enumera y detalla el grado de consecución de cada uno de ellos:

- Se han adquirido los conocimientos necesarios sobre el sistema operativo específico de robots ROS ya que si no se podría haber llevado a cabo este proyecto.
- Se ha adquirido experiencia a la hora de configurar entornos de trabajos, sobre todo aquellos relacionados con los sistemas operativos Ubuntu (Linux). Gracias al desarrollo del proyecto se puede que el Linux ofrece sistemas operativos muy flexibles y con gran capacidad para desarrollar aplicaciones informáticas, así como una amplia gama de librerías y herramientas.
- Se ha sido capaz de desarrollar un proyecto en ROS mínimamente funcional, lo que ha permitido conocer cómo se crean y desarrollan este tipo de aplicaciones. Se conluye también que ROS es una herramienta muy potente para el desarrollo de aplicaciones robóticas.
- Se han adquirido los conocimientos necesarios sobre el estándar OPC-UA, de manera que se tiene una mínima base sobre el funcionamiento del mismo.
- Se ha sido capaz de desarrollar una aplicación cliente – servidor en el protocolo OPC-UA con la implementación open62541 lo que ha permitido que se conozca cómo trabjar con dichar implementación, además de la capacidad de aprender usando manuales de uso.
- Se ha sido capaz de combinar lo aprendido en el protocolo OPC-UA y en el sistema operativo ROS.
- Se ha conocido la forma de configurar aplicaciones o entornos a partir de los ficheros .xml. Se reitera que el hecho de programar aplicaciones que posteriormente puedan configurarse sin necesidad de reprogramar es una valiosa herramienta cuyas principales ventajas con la eficiencia y la rapidez a la hora de trabajar.
- Se ha conocido el lenguaje de programación Python 3, y se ha aprendido a desarrollar una aplicación gráfica en el mismo, que

permite la creación de ficheros .xml de una manera dinámica. Se concluye que Python es un lenguaje muy sencillo de aprender pero también muy potente y útil para el desarrollo de aplicaciones gráficas o similares.

- Se han han adquirido destrezas en el planteamiento y realización de pruebas de funcionamiento teóricas con una simulación y reales con el robot RB1, para ello ha sido necesario conocer mínimamente cómo inicializar este tipo de robots.
- Se ha realizado una discusión sobre los resultados obtenidos de manera que se ha podido concluir el éxito en la realización de esta aplicación.

9. RECOMENDACIONES Y TRABAJOS FUTUROS

Este proyecto al igual que la mayoría de proyectos científicos puede ser mejorado y profundizado. A grandes rasgos hay ciertos puntos del proyecto que no han sido abordados y que puede ser de gran interés como futuros proyectos.

- Desarrollar una apliación cliente que actualice los valores en tiempo real, es decir, a la vez que se actualizan en el servidor se actualicen una aplicación con una interfaz gráfica, de manera que permita al usuario poder observar todos los valores al mismo tiempo.
- Actualmente el servidor se encuentra en una red local, sería de profundo interes desarrollar un sistema para colgarlo en una red pública, desarrollando distintos perfiles de usuario con nombre y contraseña, de manera que según el puesto que el usuario tenga en la empresa podrá acceder a unos valores u a otros desde cualquier lugar geográfico.
- Mejorar la interfaz gráfica que se ha desarrollado para poder configurar los archivos .xml. Si bien es cierto que la interfaz propuesta cumple con su propósito hay ciertos aspectos que se pueden mejorar:
    - La apariencia de la interfaz.
    - Para añadir la variable a la que se quiere suscribir el usuario es necesario conocer el nombre de antemano, se propone un sistema que muestra todas las variables posibles dentro de un topic y que se puede elegir la que se desee.

10. REFERENCIAS BIBLIOGRÁFICAS

- Wiki.ros.org
- Stackoverflow.com aparatado de preguntas relativas a ROS.
- Manual de ROS. Universidad de Alicante. Asignatura Visión artificial y robótica 34030. Curso 2017-2018. Diego Viejo Hernando.
- Gradient.org
- Open62541.org
- Canal de youtube Johanes 4Linux. Vídeos relativos a open62541.
- openwebinars.net/blog/que-es-ros/
- Programming for Robotics. Introduction to ROS. Dominic Jud, Martin Wemelinger, Prof. Dr. Marco Hutter. ETH Zürich. 2020.
- Medium.com
- Likegeeks.com
- Canal de youtube 'Pildorasinformaticas'. Vídeos de la lista Curso Python.
- Python-para-impacientes.blogspot.com 2015.
- Answers.ros.org

11. ANEXOS

11.1 ANEXO 1. INSTALACIÓN DE ROS

Sobre un sistema operativo Ubuntu 16.04, se va a instalar ROS kinetic. Para ello hay que seguir los siguientes pasos:

1. Configuración del ordenador para poder aceptar los paquetes software de la página oficial de ROS.

```
$sudo sh –c ‘echo “deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main” > /etc/apt/sources.list.d/ros-
latest.list’
```

2. Preparación de las teclas.

```
$sudo apt-key adv –keyserver ‘hkp://keyserver.ubuntu.com:80’ –
recv-ke y C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

3. Se instala la versión completa de ROS Kinetic.

```
$sudo apt-get install ros-kinetic-desktop-full
```

4. Antes de usar ROS es necesario inicializar ‘rosdep’. ‘rosdep’ permite la instalación de forma sencilla de depencias del sistema para los archivos fuente que se desean compilar, y es necesario ejecutar algunos componentes principales en ROS.

```
$sudo rosdep init
$rosdep update
```

11.2 ANEXO 2. MANUAL DE USUARIO

Este anexo se va a utilizar para redactar un manual de usuario para una instalación estándar se ROS.

11.2.1 Requisitos de instalación.

Sistema operativo: Ubuntu 16.04.

Versión ROS: ROS Kinetic Kame.

Versión Python. Python 3.6

Memoria disponible: 50 Mb

11.2.2 Uso de la aplicación GUI para la confección de archivos .xml

Para poder usar esta aplicación es necesario instalar el compilador de python 3.6 y crear un directorio denominado catkin_build_ws (otro nombre también puede ser válido) sobre el que se realizarán las siguientes operaciones:

Instalar el compilador de python 3.6:

```
$sudo apt-get install software-properties-common python-
software-properties
$sudo apt-get update
$sudo apt-get install python3.6
```

Crear directorio:

```
$mkdir catkin_build_ws && cd catkin_build_ws
```

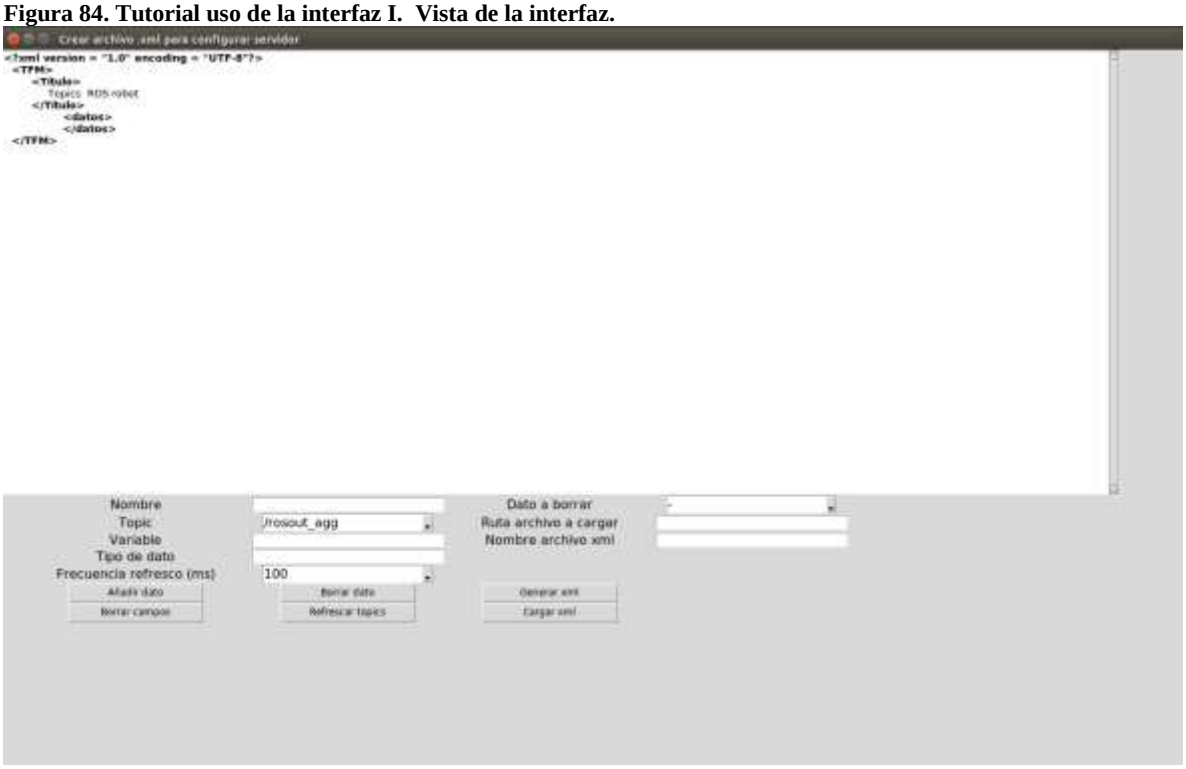
```
$cd catkin_build_ws
$catkin config -DPYTHON_EXECUTABLE=/usr/bin/python3 -
DPYTHON_INCLUDE_DIR=/usr/include/python3.6m -
DPYTHON_LIBRARY=/usr/lib/x86_64-linux-gnu/libpython3.6m.so
$catkin config –install
$catkin build
```

Por último, para ejecutar la aplicación, copiar la carpeta i*nterfaz_py*, en la carpeta *'src'* , compilar el *catkin_build_ws,* entrar dentro de la carpeta scripts y ejecutar la aplicación:

```
$catkin build
$cd src/interfaz_py/scripts
$python3 GUI
```

Se abrirá la siguiente interfaz:

Figura 84. Tutorial uso de la interfaz I. Vista de la interfaz.



Si se investiga un poco más por la interfaz se comprobará que sólo aparece el topic de */rosout_agg*.

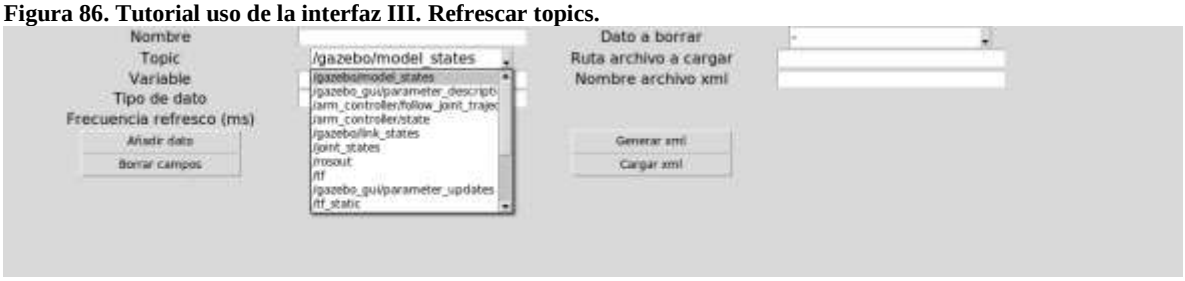
Figura 85. Tutorial uso de la interfaz II. Topics disponibles.

Para que aparezcan los topics de un robot se debe iniciar dicho robot en ROS, por ejemplo:

```
$roslaunch ur_gazebo ur5.launch
```

Tras realizar esto, si se vuelve a la interfaz, y se pulsa sobre el botón *'refrescar topics'* se comprobará que se han añadido todos los topics nuevos que hay ahora en ROS:

Figura 86. Tutorial uso de la interfaz III. Refrescar topics.



También se puede comprobar que por el momento no hay ningún dato añadido, por lo que la lista de datos posibles a borrar no deberá mostrar ningún valor, esto se puede apreciar en la siguiente figura.

Figura 87. Tutorial uso de la interfaz IV.

También se ha verificado en esta fase de pruebas la funcionalidad de añadir un dato. Para ello, se deben rellenar los 5 campos de configuración. En este ejemplo, se quiere suscribir al topic */joint_states*, en concreto a la variable de posición de la articulación 3, con una frecuencia de refresco de 100 ms.

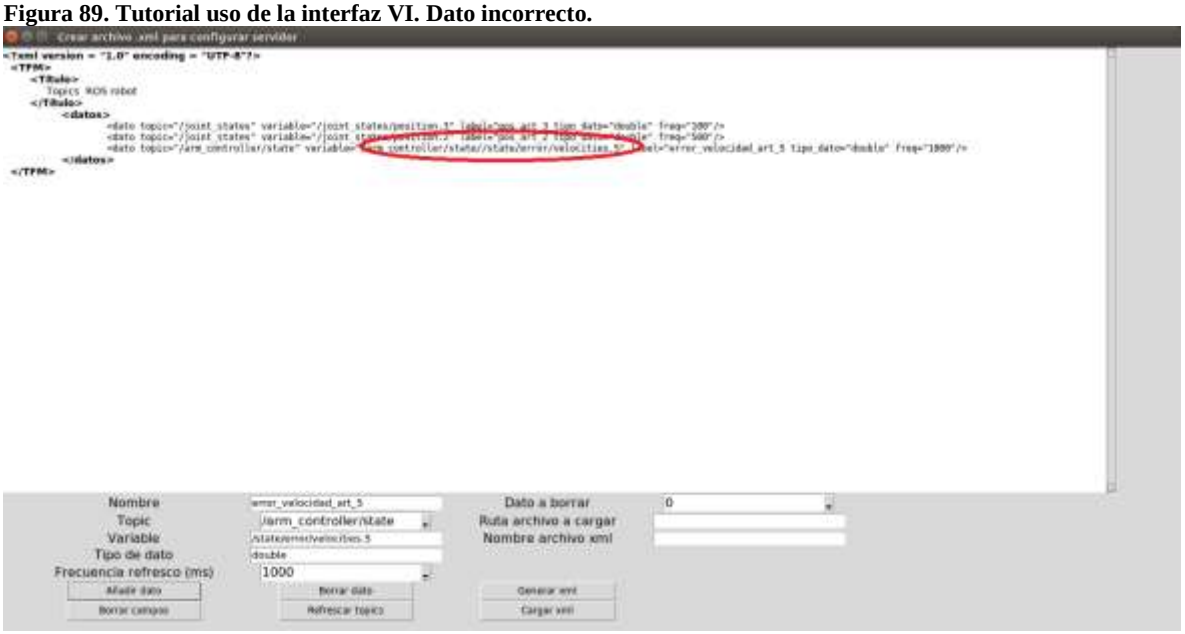
Figura 88. Tutorial uso de la interfaz V. Añadir un dato

Una vez que se ha añadido un dato, se ha comprobado que la lista de datos a borrar ya lo contempla, tal como muestra en la Figura 88.

Para esta funcionalidad se han implementado también dos formas de resolver el error que se produce cuando se añade un *topic* y en uno de los campos

se ha introducido algo incorrecto, como por ejemplo se muestra que ha ocurrido en la Figura 89.

Figura 89. Tutorial uso de la interfaz VI. Dato incorrecto.



La primera posibilidad es editar directamente el fichero, pues en la ventana del archivo, se puede escribir y borrar.

Figura 90. Tutorial uso de la interfaz VI. Corregir un error.

La segunda posiblidad es borrar directamente el dato completo, seleccionando el número de dato que se quiere eliminar y pulsando sobre el botón '*borrar dato'*.

Figura 91. Tutorial uso de la interfaz VII. Borrar un dato.

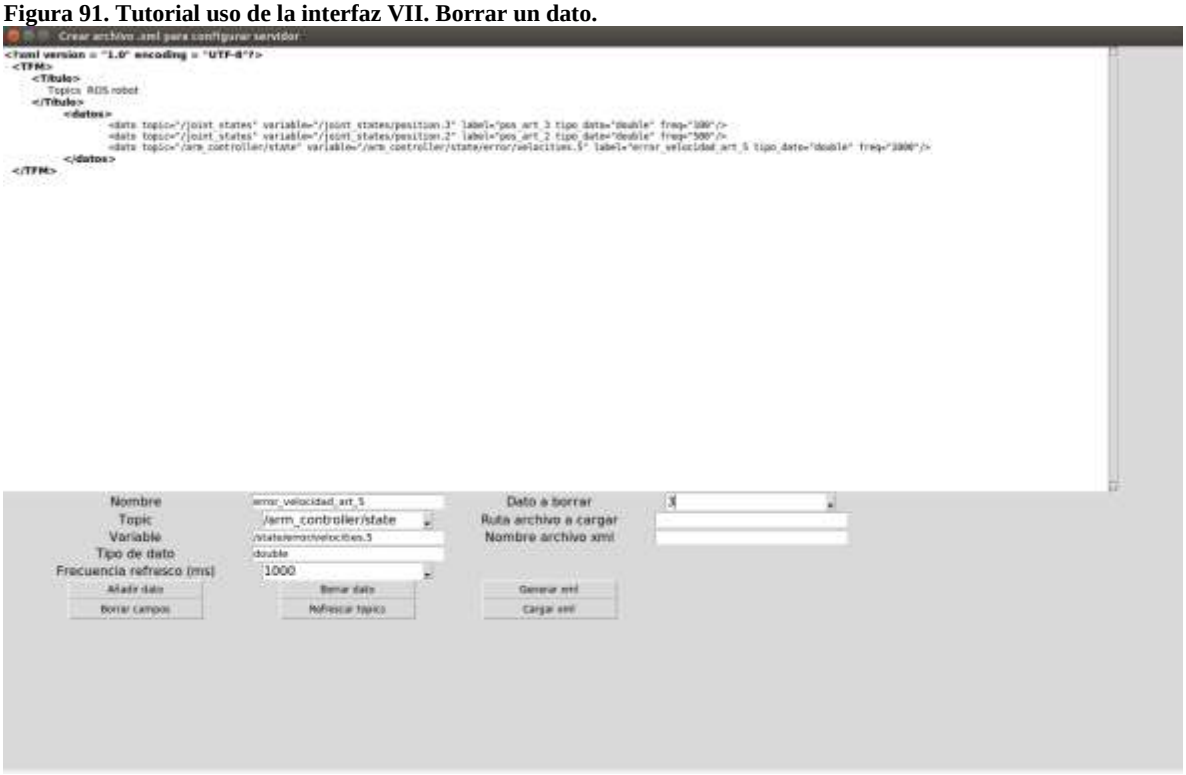
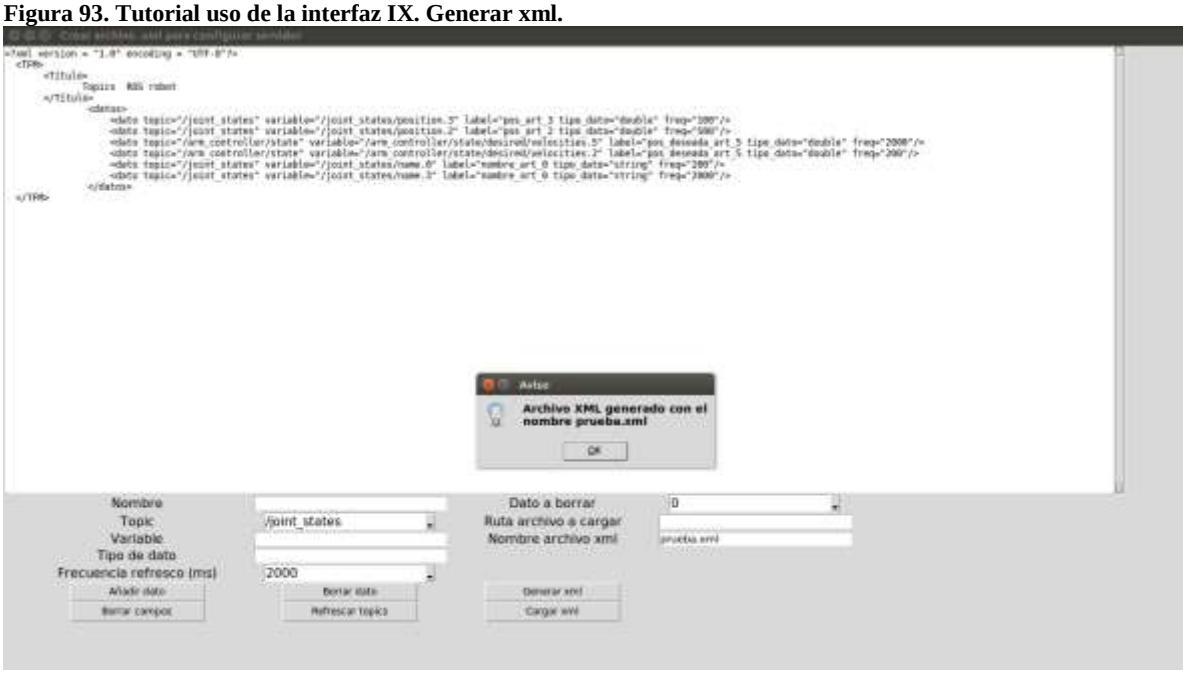


Figura 92. Tutorial uso de la interfaz VIII. Dato borrado.

El fichero .xml se puede generar cuando se ha terminado de especificar toda la información que debe incluirse en él. Para ello, se escribe un nombre en la casilla de ‘Nombre archivo xml’ y se pulsa el botón ‘Generar xml’.

Figura 93. Tutorial uso de la interfaz IX. Generar xml.



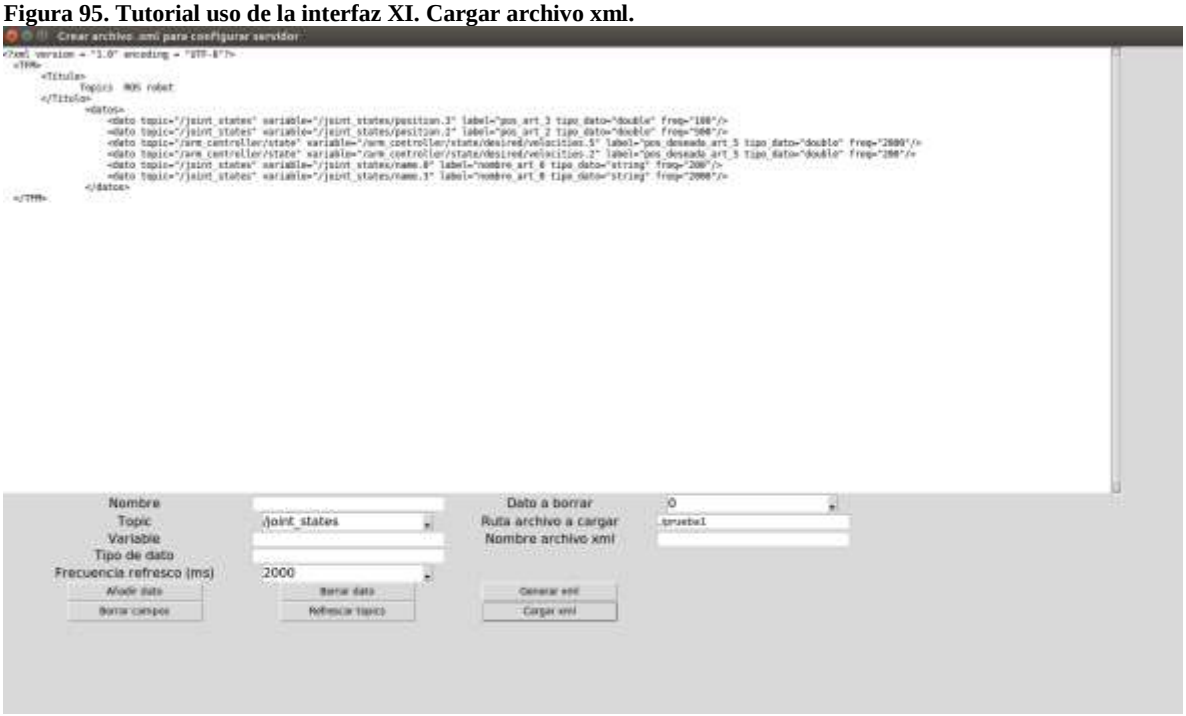
Si no se especifica ningún nombre el archivo se guardará con el nombre predeterminado ‘config.xml’.

Se ha verificado también la funcionalidad de ‘Borrar campos’. Al pulsar el botón *‘Borrar campos’* se borran todos los campos de la sección de configuración.

Figura 94. Tutorial uso de la interfaz X. Borrar campos.

Otras de las funcionalidades de la aplicación que se ha verificado es la carga y edición de un archivo .xml ya existente. Para ello es necesario rellenar el campo de ‘*Ruta archivo a cargar’* y pulsar el botón ‘*Cargar xml’*.

Figura 95. Tutorial uso de la interfaz XI. Cargar archivo xml.



Sobre este archivo cargado se han realizado todas las acciones descritas anteriormente para modificar el archivo, y una vez hecho esto se ha guardado el archivo. Se ha verificado que los cambios realizados han quedado almacenados en el archivo generado. Hay que tener en cuenta que todos los archivos guardados se guardan en la carpeta '*scripts'* en la que se encuentra la aplicación.

11.2.3 Puesta en marcha del servidor.

La verificación de la puesta en marcha del servidor se ha llevado a cabo siguiendo los pasos que se especifican a continuación.

Para el uso del servidor es necesario copiar las carpetas *'ros_type_introspection'* y *'servidor_ros',* sobre la carpeta *'src'* del espacio habitual de trabajo de ROS. Una vez realizado esto se compila el espacio de trabajo de la manera habitual.

```
$cp –r ros_type_introspection ~/catkin_ws/src
$cp –r servidor_ros ~/catkin_ws/src
$cd ~/catkin_ws
$catkin_make
```

Una vez terminada la compilación es necesario moverse al directorio donde se encuentra el ejecutable.

```
$cd ~/devel/lib/servidor_ros
```

Para evitar problemas, se aconseja copiar el archivo de configuración .xml, en la misma carpeta que el ejecutable, de manera que a la hora de indicar la ruta solo se

tenga que escribir *'./configuracion.xml'.* Una vez dentro de este carpeta para inciar se debe escribir:

```
$./server
```

Tras esto, se ha verificado que se ha empezado a ejecutar un servidor en la dirección '*localhost'* en el puerto 4840. También se ha verificado que se puede cambiar esta dirección especificándo tanto dirección como puerto en el momento de lanzar la aplicación:

```
$./server 192.168.1.231 4048
```

Cuando se ejecuta la aplicación, desde el terminal se pide espeficiar el archivo de configuración, por defecto se carga el archivo *'datos.xml'* que viene ya creado en dicha carpeta. Si se ha especificado un archivo, se verifica que es éste el que se carga. Cuando no se ha especificado ninguno, se ha verificado que se carga por defecto el archivo datos.xml.
En las pruebas se ha cargado el archivo 'prueba.xml' generado previamente. Para ello se ha copiado el archivo desde la carpeta '*scripts'*:

```
$cp –r prueba.xml ~/catkin_ws/devel/lib/servidor_ros
```

Figura 96. Tutorial uso del servidor I. Lanzar el servidor.

El servidor ya se está ejecutando, desde una aplicación cliente se puede acceder a toda la información. En las pruebas se ha utilizado la aplicación cliente de Prosys desde un sistema windows que se encuentra en la misma red que el servidor:

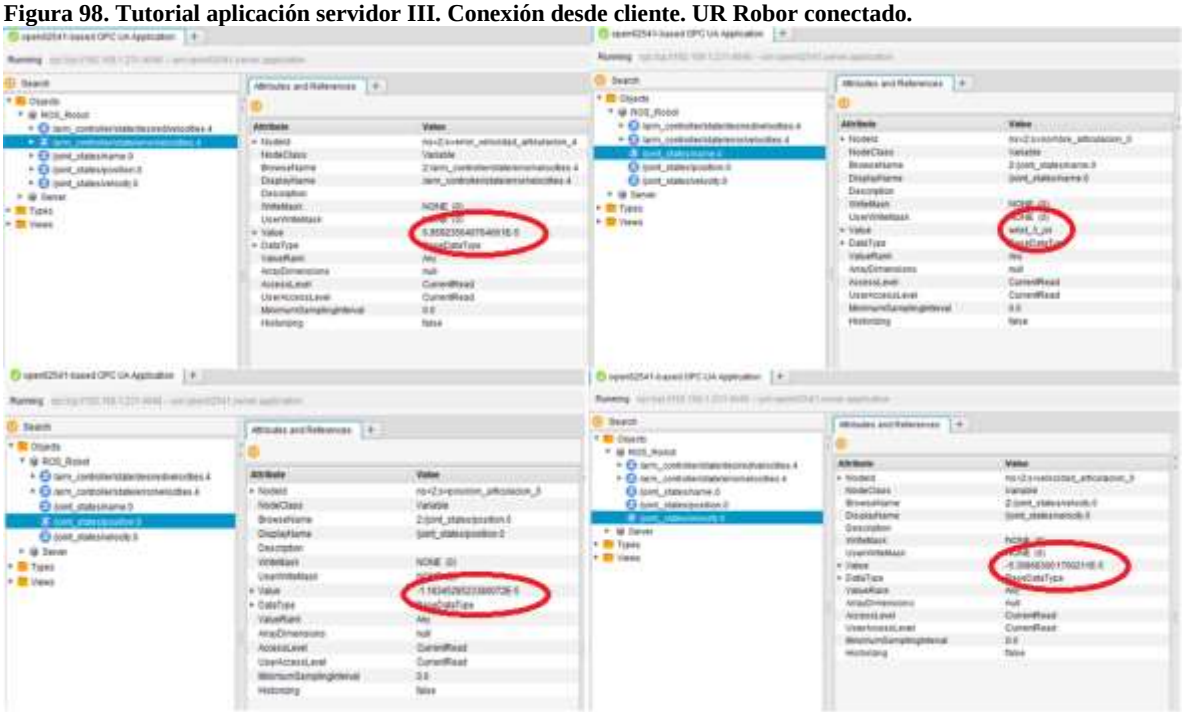
Figura 97. Tutorial aplicación servidor II. Conexión desde cliente. UR Robor sin conectar.

Como en este momento de la prueba el robot UR5 no está conectado al sistema, se ha verificado que hay un valor de error de velocidad de la articulación 4 es 0.0. Para continuar con las pruebas, se ha iniciado la simulación:

```
$roslaunch ur_gazebo ur5.launch
```

Los valores de los *topics*, donde se ha sucribido la aplicación *server*, se ha comprobado que ya sí son distintos del valor 0.0:

Figura 98. Tutorial aplicación servidor III. Conexión desde cliente. UR Robor conectado.



En estas pruebas de verificación de la aplicación se ha usado la aplicación cliente de Prosys, pero puede usarse cualquier otra aplicación para acceder al servidor, siempre y cuando se realice la configuración necesaria para poder conectarse a la dirección IP y puerto especificado.

11.2.4 Obtención de los nombres de los *topics* que se van a usar

Los distintos manuales de ROS que se han consultado para el desarrollo de este trabajo no describen de manera clara cuál es el nombre exacto que se ha de usar. Así, en algunos manuales se indica que el nombre del topic es *joint_states/position.0*, mientras que en otros se indica que es *joint_states/position/0*.

Estas pequeñas diferencias puedan llevar a error a la hora de configurar el archivo .xml, y por tanto la aplicación servidor no se suscribirá al topic correcto.

En las pruebas que se ha hecho, se ha comprobado que una forma de evitar este pequeño error es usar la aplicación servidor sin haber configurado los subtopics. Así, se configura el fichero .xml de manera que se suscriba al topic del que posteriormente se querrán obtener los subtopics, por ejemplo:

Figura 99. Manual de usario. Conocer nombre del topic. Archivo .xml.

```
−<datos>
    <dato topic="/joint_states" variable="/joint_states/position.0" label="posicion_articulacion_0" freq="100" tipo="double"/>
    <dato topic="/joint_states" variable="/joint_states/velocity.0" label="velocidad_articulacion_0" freq="500" tipo="double"/>
```

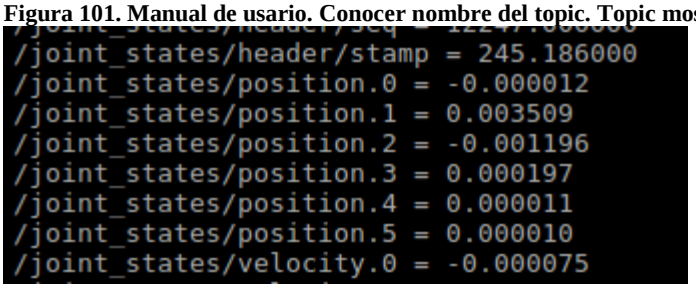
Se rellena el campo del subtopic, es decir, el campo de la variable, aunque no se sepa si ese campo estará o no bien. Se abre el código fuente y se descomentan las líneas 106 para mostrar los topics de tipo númerico, y 125 para los topics de tipo string.

Figura 100. Manual de usario. Conocer nombre del topic. Código fuente.

```
103                 pthread_mutex_unlock(&ROS_datos[i].mutex);
104             }
105             /*Descomentar en caso de querer mostrar el nombre el topic*/
106             printf(" %s = %f\n", key.c_str(), value.convert<double>() );
107         }
108         for (auto it: flat_container.name)
109         {
110             const std::string& key    = it.first.toStdString();
111             const std::string& value  = it.second;
112
113             if(!strcmp(key.c_str(),ROS_datos[i].variable)){
114                 UA_String s;
115                 UA_String_init(&s); // _init zeroes out the entire memory of the datat
116                 char *test = (char *)value.c_str();
117                 s.length = strlen(test);
118                 s.data = (UA_Byte*)test;
119                 pthread_mutex_lock(&ROS_datos[i].mutex);
120                 UA_String_copy(&s,&ROS_datos[i].valor_string);
121                 pthread_mutex_unlock(&ROS_datos[i].mutex);
122
123             }
124             /*Descomentar en caso de querer mostrar el nombre el topic*/
125         printf(" %s = %s\n", key.c_str(), value.c_str() );
126
127         }
```

Se recompila el archivo, y se ejecuta la aplicación servidor, de manera que la salida por el terminal mostrará el nombre completo de todos los subtopics asociado a ese topic:

Figura 101. Manual de usario. Conocer nombre del topic. Topic mostrado



Con esta propuesta se ha resuelto de manera satisfactoria el problema descrito.

11.3 ANEXO 3. CÓDIGO DEL SERVIDOR.

```
/*
 * server.cpp
 *
 *  Created on: 24 jun. 2020
 *      Author: ester
 */


#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>
#include <open62541/server_config_default.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <pthread.h>
#include "geometry_msgs/Twist.h"
#include "nav_msgs/Odometry.h"
#include "control_msgs/JointTrajectoryControllerState.h"
#include "sensor_msgs/JointState.h"
 //Para poder leer archivos XML
#include <pugixml.hpp>
#include <iostream>
#include <fstream>
using namespace std;
//Para añadir introspección
#include "ros_type_introspection/ros_introspection.hpp"
#include <topic_tools/shape_shifter.h>
using namespace RosIntrospection;
using topic_tools::ShapeShifter;
#define MAX 100 //En principio habrá un máximo de 100 datos, la idea es hacer
flexible el número total de datos

struct dato_robot {
	char* topic;
	char* variable;
	char* label;
	int freq;
	char* tipo_dato;
	pthread_mutex_t mutex;
	UA_Double valor;
	UA_String valor_string;
	UA_VariableAttributes nAttr;
	ros::Subscriber subscriber;
	//const char* valor_string;
};
struct dato_ini { //Dato auxiliar para guardar los argumentos del main y usarlos en
el hilos que lanza los suscriptores de ROS
	int argc;
	char** argv;
};
//Para controlar el apagado y encendido del servidor
static volatile UA_Boolean running = true;
```

```
//Declaración del servidor
UA_Server *server;
//Variables
dato_robot ROS_datos[MAX];
int total_datos;
//Función callback:
void topicCallback(const ShapeShifter::ConstPtr& msg, const std::string &topic_name,
RosIntrospection::Parser& parser) {

	const std::string&  datatype = msg->getDataType();
	const std::string&  definition = msg->getMessageDefinition();
	parser.registerMessageDefinition(topic_name,
		RosIntrospection::ROSType(datatype),
		definition);
	std::vector<uint8_t> buffer;
	FlatMessage   flat_container;
	RenamedValues renamed_value;

	// copy raw memory into the buffer
	buffer.resize(msg->size());
	ros::serialization::OStream stream(buffer.data(), buffer.size());
	msg->write(stream);

	// deserialize and rename the vectors
	parser.deserializeIntoFlatContainer(topic_name, Span<uint8_t>(buffer),
&flat_container, 100);


	//   ROS_datos[0]_valor = (UA_Double)
flat_container.value[2].second.convert<double>();
	  //
parser.deserializeIntoFlatContainer(topic_name,Span<uint8_t>(buffer,&flat_container,
100);
	parser.applyNameTransform(topic_name, flat_container, &renamed_value);

	// Print the content of the message
   // printf("--------- %s ----------\n", topic_name.c_str() );
	for (int i = 0; i < total_datos; i++) {
		for (auto it : renamed_value)
		{
			const std::string& key = it.first;
			const Variant& value = it.second;

			if (!strcmp(key.c_str(), ROS_datos[i].variable)) {
				pthread_mutex_lock(&ROS_datos[i].mutex);
				ROS_datos[i].valor = (UA_Double)value.convert<double>();
				pthread_mutex_unlock(&ROS_datos[i].mutex);
			}

			// printf(" %s = %f\n", key.c_str(), value.convert<double>() );
		}
		for (auto it : flat_container.name)
		{
			const std::string& key = it.first.toStdString();
			const std::string& value = it.second;

			if (!strcmp(key.c_str(), ROS_datos[i].variable)) {
				UA_String s;
```

```
                        UA_String_init(&s); // _init zeroes out the entire memory
of the datatype
                        char *test = (char *)value.c_str();
                        s.length = strlen(test);
                        s.data = (UA_Byte*)test;
                        pthread_mutex_lock(&ROS_datos[i].mutex);
                        UA_String_copy(&s, &ROS_datos[i].valor_string);
                        pthread_mutex_unlock(&ROS_datos[i].mutex);
                        //printf(" %s = %s\n", key.c_str(), value.c_str() );
                    }
                    //printf(" %s = %s\n", key.c_str(), value.c_str() );

            }
        }

}
void *nodos_ROS(void *param) {
        dato_ini ini = *(dato_ini*)param;
        ros::init(ini.argc, ini.argv, "listener");
        ros::NodeHandle nh;

        Parser parser;
        for (int i = 0; i < total_datos; i++) {

                boost::function<void(const ShapeShifter::ConstPtr&) > callback;
                const std::string topic_name = ROS_datos[i].topic;
                callback = [&parser, topic_name](const ShapeShifter::ConstPtr& msg)
                {
                        topicCallback(msg, topic_name, parser);
                };
                ROS_datos[i].subscriber = nh.subscribe(topic_name, 10, callback);
        }
        ros::spin();
}
//Función que el servidor llama cada vez que va a actualizar la información
void *Actualizar(void *param) {
        int freq = *(int*)param;
        while (running == true) {
                UA_Variant value;
                for (int i = 0; i < total_datos; i++) {
                        if (ROS_datos[i].freq == freq) {
                                if (!strcmp(ROS_datos[i].tipo_dato, "double")) {
                                        pthread_mutex_lock(&ROS_datos[i].mutex);
                                        UA_Variant_setScalarCopy(&value,
&ROS_datos[i].valor,
                                                &UA_TYPES[UA_TYPES_DOUBLE]);
                                        pthread_mutex_unlock(&ROS_datos[i].mutex);
                                        UA_Server_writeValue(server, UA_NODEID_STRING(2,
ROS_datos[i].label), value);
                                }
                                else {
                                        pthread_mutex_lock(&ROS_datos[i].mutex);
                                        UA_Variant_setScalarCopy(&value,
&ROS_datos[i].valor_string,
                                                &UA_TYPES[UA_TYPES_STRING]);
                                        pthread_mutex_unlock(&ROS_datos[i].mutex);
                                        UA_Server_writeValue(server, UA_NODEID_STRING(2,
ROS_datos[i].label), value);
```

```
                        }
                    }
                }

                usleep(freq);
        }
}


static void stopHandler(int sign) {
        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
        running = false;
}

int main(int argc, char *argv[]) {

        signal(SIGINT, stopHandler);
        signal(SIGTERM, stopHandler);

        pthread_t hilo_ros, hilos_freq[5];
        pthread_attr_t attr;
        pthread_attr_init(&attr);

        //Configuración de variables que se leen a partir del archivo XML
        char archivo[100];
        cout << "Ruta del archivo .xml de configuración:\n";
        cin.getline(archivo, 100, '\n');
        pugi::xml_document doc;
        pugi::xml_parse_result result;
        /*  if(!strcmp(archivo[0], (char)'\n'))  result = doc.load_file("datos.xml");
          else result = doc.load_file(archivo);*/
        result = doc.load_file(archivo);



        if (!result) {
                printf("Error leyendo el archivo xml\n");
                std::cout << "Load result: " << result.description();
                return -1; //Si no se puede leer el xml se cierra el programa

        }

        pugi::xml_node datos = doc.child("TFM").child("datos");

        //Configurar datos desde el fichero .xml
        int i = 0;
        for (pugi::xml_node dato = datos.first_child(); dato; dato =
dato.next_sibling())
        {


                ROS_datos[i].topic = (char*)dato.attribute("topic").value();
                ROS_datos[i].variable = (char*)dato.attribute("variable").value();
                ROS_datos[i].label = (char*)dato.attribute("label").value();
                ROS_datos[i].freq = dato.attribute("freq").as_int();
                ROS_datos[i].mutex = PTHREAD_MUTEX_INITIALIZER;
                ROS_datos[i].tipo_dato = (char*)dato.attribute("tipo").value();
```

```
			i++;
		}

		total_datos = i;
//Servidor
		server = UA_Server_new();

		//Chequear argumentos
		if (argc > 2) { //hostname or ip adress and a port number are available
			UA_Int16 port_number = atoi(argv[2]);
			UA_ServerConfig_setMinimal(UA_Server_getConfig(server), port_number,
0);

		}
		else
			UA_ServerConfig_setDefault(UA_Server_getConfig(server));
		if (argc > 1) {
			//hostname or ip address available
			//Copy hostname from char * to an open62541 variable
			UA_String hostname;
			UA_String_init(&hostname);
			hostname.length = strlen(argv[1]);
			hostname.data = (UA_Byte *)argv[1];

			//change configuration
			UA_ServerConfig_setCustomHostname(UA_Server_getConfig(server),
hostname);
		}
		//Variable config para cambiar el verifyTimeRequest y poder conectarse con
client Prosys
		UA_ServerConfig* config = UA_Server_getConfig(server);
		config->verifyRequestTimestamp = UA_RULEHANDLING_ACCEPT;

		//Configuración del espacio ROS dentro del servidor
		UA_Int16 ns_ros = UA_Server_addNamespace(server, "ROS");
		UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "New Namespace added with
Nr %d\n", ns_ros);

		//Añadir un objeto nuevo denominado ROS_robot,
		UA_NodeId ROS_robot; /* get the nodeid assigned by the server */
		UA_ObjectAttributes oAttr = UA_ObjectAttributes_default;

		UA_Server_addObjectNode(server, UA_NODEID_NULL,
			UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
			UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
			UA_QUALIFIEDNAME(2, "ROS_Robot"),
			UA_NODEID_NULL,
			oAttr, NULL, &ROS_robot);



		//Añadir variables de ROS_robot, de acuerdo al fichero .xml
		for (int i = 0; i < total_datos; i++)
		{
			if (!strcmp(ROS_datos[i].tipo_dato, "double"))
			{
				ROS_datos[i].nAttr = UA_VariableAttributes_default;
```

```
                    UA_Variant_setScalar(&ROS_datos[i].nAttr.value,
&ROS_datos[i].valor, &UA_TYPES[UA_TYPES_DOUBLE]);
                    UA_Server_addVariableNode(server, UA_NODEID_STRING(2,
ROS_datos[i].label), ROS_robot,
                           UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                           UA_QUALIFIEDNAME(2, ROS_datos[i].variable),
                           UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
ROS_datos[i].nAttr, NULL, NULL);
             }
             if (!strcmp(ROS_datos[i].tipo_dato, "string"))
             {
                    ROS_datos[i].nAttr = UA_VariableAttributes_default;
                    UA_Variant_setScalar(&ROS_datos[i].nAttr.value,
&ROS_datos[i].valor_string, &UA_TYPES[UA_TYPES_STRING]);
                    UA_Server_addVariableNode(server, UA_NODEID_STRING(2,
ROS_datos[i].label), ROS_robot,
                           UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                           UA_QUALIFIEDNAME(2, ROS_datos[i].variable),
                           UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
ROS_datos[i].nAttr, NULL, NULL);
             }
      }



      //Lanzar hilo de ROS
      dato_ini inicializacion;
      inicializacion.argc = argc;
      inicializacion.argv = argv;
      pthread_create(&hilo_ros, &attr, nodos_ROS, &inicializacion);
      //Lanzar  hilos para actualiar sólo se lanzan aquellos hilos demandados por
el fichero de configuración
      int cont_hilos = 0;
      for (int i = 0; i < total_datos; i++) {
             if (ROS_datos[i].freq == 100) {
                    pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
                    cont_hilos++;
                    break;
             }
      }
      for (int i = 0; i < total_datos; i++) {
             if (ROS_datos[i].freq == 200) {
                    pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
                    cont_hilos++;
                    break;
             }
      }
      for (int i = 0; i < total_datos; i++) {
             if (ROS_datos[i].freq == 500) {
                    pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
                    cont_hilos++;
                    break;
             }
      }
      for (int i = 0; i < total_datos; i++) {
```

```
            if (ROS_datos[i].freq == 1000) {
                  pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
                  cont_hilos++;
                  break;
            }
      }
      for (int i = 0; i < total_datos; i++) {
            if (ROS_datos[i].freq == 2000) {
                  pthread_create(&hilos_freq[cont_hilos], &attr, Actualizar,
&ROS_datos[i].freq);
                  cont_hilos++;
                  printf("Hilo lanzado 2000\n");
                  break;
            }
      }

      //Lanzar servidor
      UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Starting new
Server...");
      UA_StatusCode retval = UA_Server_run(server, &running);
      UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Server was shut down");
      UA_Server_delete(server);
      //Matar el hilo de ROS
      pthread_cancel(hilo_ros);
      return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;

}
```

11.4 ANEXO 4. CÓDIGO DE LA INTERFAZ GRÁFICA.

```
#!/usr/bin/python3.6

import rospy
from std_msgs.msg import String
from tkinter import *
from tkinter import scrolledtext
from tkinter import ttk
from tkinter.ttk import *
from tkinter import messagebox

cont_lineas = 0
#---------Topics para mostrar----------#
topics_enteros = rospy.get_published_topics(namespace='/')
topics = topics_enteros
for i in range(len(topics_enteros)):
    aux = ' '
    aux = aux.join(topics_enteros[i])
    topics[i] = aux.split(" ")[0]
#------------Configuración window----------#
window = Tk()
```

```
window.title("Crear archivo .xml para configurar servidor")
window.geometry("1500x1500")
#-----------Para entrar el nombre ---------#
label1 = Label(window, text = "Nombre", font= ("Calibri", 12))
label1.grid(column = 1, row = 1)
texto1 = StringVar()
txt1 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto1)
txt1.grid(column = 2, row = 1)
#--------Para mostrar los topics posibles a los que suscribirse topics--#
combo2 = Combobox(window, font= ("Calibri", 12))
combo2['values'] = (topics)
combo2.current(0)
combo2.grid(column = 2, row = 2)
label2 = Label(window, text = "Topic", font= ("Calibri", 12))
label2.grid(column = 1, row = 2)
#--------Para mostrar la variable-------#
label3 = Label(window, text = "Variable", font= ("Calibri", 12))
label3.grid(column = 1, row = 3)
texto3 = StringVar()
txt3 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto3)
txt3.grid(column = 2, row = 3)
#----------Para tipo de dato------------#
label8 = Label(window, text = "Tipo de dato", font= ("Calibri", 12))
label8.grid(column = 1, row = 4)
texto8 = StringVar()
txt8 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto8)
txt8.grid(column = 2, row = 4)
#---------Para elegir la frecuencia de refresco-----------#
combo4 = Combobox(window, font= ("Calibri", 12))
combo4['values'] = (100, 200, 500, 1000, 2000)
combo4.current(0)
combo4.grid(column = 2, row = 5)
label4 = Label(window, text = "Frecuencia refresco (ms)", font= ("Calibri",
12))
label4.grid(column = 1, row = 5)

#-----------Vista previa archivo xml-----------#
archivo = scrolledtext.ScrolledText(window, width=200, height=40)
archivo.grid(row=0, column=0,  columnspan=12, sticky=E+W+N+S)

#------------Para elegir qué topic se borra de los ya añadidos-----#
combo5 = Combobox(window, font= ("Calibri", 12))
```

```
combo5['values'] = ('-')
combo5.current(0)
combo5.grid(column = 4, row = 1)
label5 = Label(window, text = "Dato a borrar", font= ("Calibri", 12))
label5.grid(column = 3, row = 1)
#------------------Para cargar un archivo xml------#
label6 = Label(window, text = "Ruta archivo a cargar", font= ("Calibri",
12))
label6.grid(column = 3, row = 2)
texto6 = StringVar()
txt6 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto6)
txt6.grid(column = 4, row = 2)
#------------------Nombre del archivo xml generado------#
label7 = Label(window, text = "Nombre archivo xml", font= ("Calibri", 12))
label7.grid(column = 3, row = 3)
texto7 = StringVar()
txt7 = Entry(window, width = 30, font= ('Calibri 10'), textvariable =
texto7)
txt7.grid(column = 4, row = 3)

#----------Función añadir dato----------#
def add_topic():
    if txt1.get()=="" or txt3.get()=="":
        messagebox.showinfo(title="Mensaje de error", message="Para añadir
un topic ningún campo puede estar vacio")
        return -1
    global cont_lineas

    cont = StringVar()
    cont = str(cont_lineas + 7.0)
    linea = StringVar()
linea ='                    <dato topic="'+combo2.get()+'"
variable="'+combo2.get()+'/' +txt3.get()+'" label="'+txt1.get()+'
tipo="'+txt8.get()+'" freq="'+combo4.get()+'"/>\n'
    archivo.insert(cont, linea)
    cont_lineas = cont_lineas + 1
    lista_cont = [0]*(cont_lineas+1)
    for i in range(cont_lineas+1):
        lista_cont[i] = i
    combo5['values'] = (lista_cont)
    combo5.current(0)
    return 0

#--------------------Funciones botones--------------#
```

```
#--------------Función borrar campos------#
def dlt_fields():
    texto1.set("")
    texto3.set("")
    texto6.set("")
    texto7.set("")
    return 0

#--------------Función refrescar topics---------#
def ref_topics():
    topics_enteros = rospy.get_published_topics(namespace='/')
    topics = topics_enteros
    for i in range(len(topics_enteros)):
        aux = ' '
        aux = aux.join(topics_enteros[i])
        topics[i] = aux.split(" ")[0]

    combo2['values'] = (topics)
    combo2.current(0)
    return 0

#----------Función borrar dato----------#
def dlt_topics():
    if combo5.get()=='0' or cont_lineas==0:
        messagebox.showinfo(title="Mensaje de error", message="Por favor
selecione un dato para borrar")
        return -1
    global cont_lineas
    ini = float(combo5.get())+6.0
    fin = float(combo5.get())+7.0
    archivo.delete(ini, fin)
    cont_lineas = cont_lineas - 1
    lista_cont = [0]*(cont_lineas + 1)
    for i in range(cont_lineas+1):
        lista_cont[i] = i
    combo5['values'] = (lista_cont)
    combo5.current(0)
    return 0

#------------------------Función generar archivo xml--------#
def gen_file():
    if txt7.get() == "":
        file = open ("./config.xml", 'w')
        file.write(archivo.get("1.0",END))
```

```
        messagebox.showinfo(title="Aviso", message="Archivo XML generado con
el nombre config.xml")
        return 0
    nombre = './' + txt7.get()
    file = open (nombre, 'w')
    file.write(archivo.get("1.0",END))
    messagebox.showinfo(title="Aviso", message="Archivo XML generado con el
nombre " + txt7.get())
    return 0

#----------Función cargar archivo xml---------#
def get_file():
    try:
        file = open(txt6.get(), 'r')
        aux = 1.0
        archivo.delete("1.0", END) #Borrar datos de archivo
        for j in file.readlines():
            cont = StringVar()
            cont = str(aux)
            archivo.insert(cont, j)
            aux = aux + 1.0
    except AttributeError:
        messagebox.showinfo(title="Mensaje de error", message="Fichero no
insertado")
        return -1
    except FileNotFoundError:
        messagebox.showinfo(title="Mensaje de error", message="La ruta no es
correcta")
        return -1
    global cont_lineas
    file.seek(0)
    cont_lineas = len(file.readlines())-9
    lista_cont = [0]*(cont_lineas + 1)
    for i in range(cont_lineas+1):
        lista_cont[i] = i
    combo5['values'] = (lista_cont)
    combo5.current(0)
    #Escribir los datos en archivo del file que se ha abierto
    # archivo.insert(cont, linea)
    file.close()
    return 0



#-------------------------------Botones----------------------------#
```

```
add_button = Button(window, text = "Añadir dato", command = add_topic, width
= 20)
dlt_button = Button(window, text = "Borrar campos", command = dlt_fields,
width = 20)
ref_button = Button(window, text = "Refrescar topics", command = ref_topics,
width = 20)
dlt2_button = Button(window, text = "Borrar dato", command = dlt_topics,
width = 20)
gen_button = Button(window, text = "Generar xml", command = gen_file,
width = 20)
get_button = Button(window, text = "Cargar xml", command = get_file,   width
= 20)
add_button.grid(column = 1, row = 10)
dlt_button.grid(column = 1, row = 12)
ref_button.grid(column = 2, row = 12)
dlt2_button.grid(column = 2, row = 10)
gen_button.grid(column = 3 , row = 10)
get_button.grid(column = 3, row = 12)

#---------------------Configurar archivo xml ----------------------#
archivo.insert('1.0','<?xml version = "1.0" encoding = "UTF-8"?>\n' )
archivo.tag_add('Linea 1', '1.0','2.0')
archivo.tag_config('Linea 1', foreground = 'black',font =("console",
10,"bold"))
archivo.insert('2.0','  <TFM>\n')
archivo.tag_add('Linea 2', '2.0','3.0')
archivo.tag_config('Linea 2', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('3.0','       <Título>\n')
archivo.tag_add('Linea 3', '3.0','4.0')
archivo.tag_config('Linea 3', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('4.0','              Topics  ROS robot\n')
archivo.tag_add('Linea 4', '4.0','5.0')
archivo.tag_config('Linea 4', foreground = 'black', font =("console", 10))
archivo.insert('5.0' , '       </Título>\n')
archivo.tag_add('Linea 5' , '5.0','6.0')
archivo.tag_config('Linea 5', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('6.0' , '               <datos>\n')
archivo.tag_add('Linea 6' , '6.0','7.0')
archivo.tag_config('Linea 6', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('7.0' , '               </datos>\n')
archivo.tag_add('Linea 7' , '7.0','8.0')
```

```
archivo.tag_config('Linea 7', foreground = 'black', font =("console",
10,"bold"))
archivo.insert('8.0','  </TFM>\n')
archivo.tag_add('Linea 8', '8.0','9.0')
archivo.tag_config('Linea 8', foreground = 'black', font =("console",
10,"bold"))

window.mainloop()
```

11.5 ANEXO 5. CONFIGURACIÓN DE LOS ENTORNOS DE PROGRAMACIÓN Y DEL SOFTWARE UTILIZADO

Se describen en este anexo la forma de configurar los entornos de programación utilizados y el software.

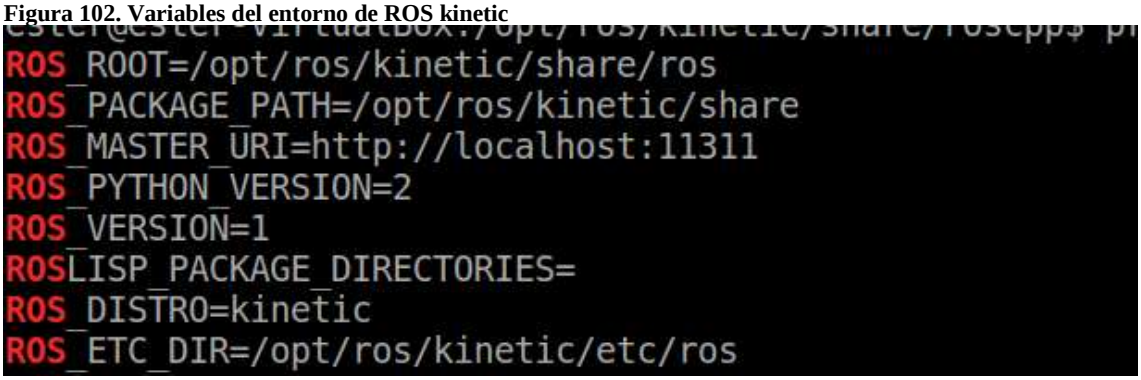
11.5.1 Configuración del entorno y espacio de trabajo.

Verificación de la instalación de las variables del entorno ROS:

```
$printenv | grep ROS
```

La respuesta que se obtiene se muestra en la siguiente figura.

Figura 102. Variables del entorno de ROS kinetic



Creación de un espacio de trabajo:

```
$mkdir -p ~/catkin_ws/src
$cd ~/catkin_ws/
$catkin_make
```

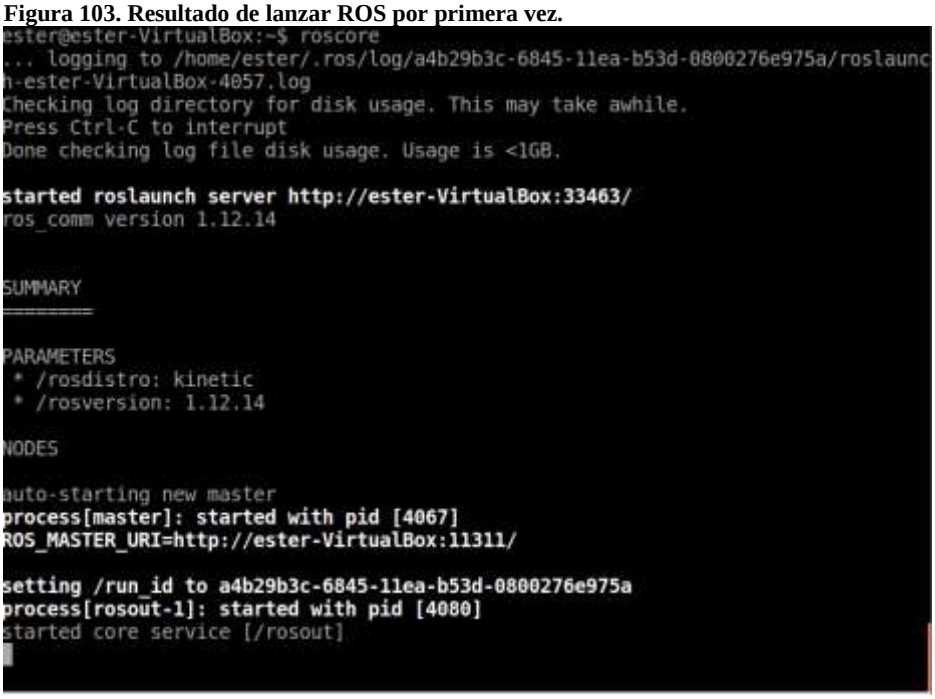
11.5.2 Preparación para la realización de las primeras pruebas

Arrancar un *master*:

```
$roscore
```

Resultado:

Figura 103. Resultado de lanzar ROS por primera vez.



11.5.3 Puesta en marcha de los nodos *talker* y el *listener* y utilidades relacionadas

```
$rosrun roscpp_tutorials talker
$rostun roscpp_tutorial listener
```

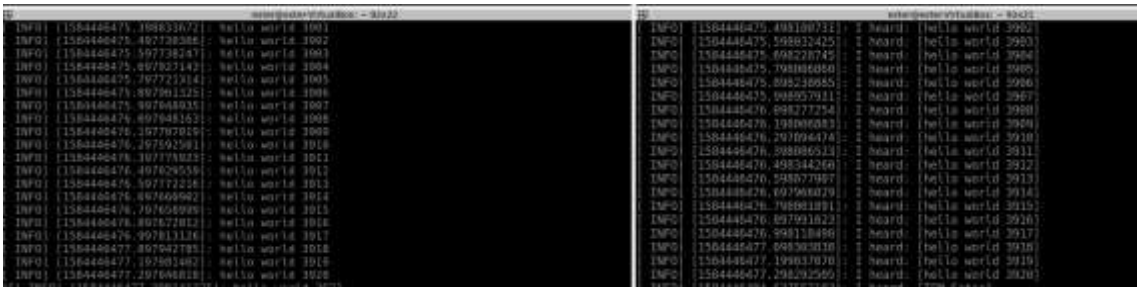


Figura 1. Resultado de ejecutar los nodos listener y talker a la vez.

Para listar la información del *topic /chatter*:

```
$rostopic info/chatter
```

Herramienta para graficar los nodos y *topics* que se encuentran funcionando, además de mostrar como se interconectan:

```
$rqt_graph
```

Figura 2. Digrama de interconexión de los nodos /talker y /listener

Para publicar un mensaje sin haber lanzado previamente un nodo, desde la consola, se elige en qué *topic* se quiere publicar, el tipo de mensaje y el contenido del mismo:

```
$rostopic pub /chatter std_msgs/String “data:’TFM Ester’”
```

Figura 104. *Listener* escuchando el mensaje que se ha publicado desde consola.

```
ester@ester-VirtualBox:~$ rostopic pub /chatter std_msgs/String "data: 'TFM Ester'"
publishing and latching message. Press ctrl-C to terminate
```

```
ester@ester-VirtualBox: ~ 92x21
[ INFO] [1584446475.498100731]: I heard: [hello world 3902]
[ INFO] [1584446475.598032425]: I heard: [hello world 3903]
[ INFO] [1584446475.698228745]: I heard: [hello world 3904]
[ INFO] [1584446475.798006860]: I heard: [hello world 3905]
[ INFO] [1584446475.898238085]: I heard: [hello world 3906]
[ INFO] [1584446475.998957931]: I heard: [hello world 3907]
[ INFO] [1584446476.098277254]: I heard: [hello world 3908]
[ INFO] [1584446476.198006883]: I heard: [hello world 3909]
[ INFO] [1584446476.297894474]: I heard: [hello world 3910]
[ INFO] [1584446476.398086523]: I heard: [hello world 3911]
[ INFO] [1584446476.498344260]: I heard: [hello world 3912]
[ INFO] [1584446476.598077907]: I heard: [hello world 3913]
[ INFO] [1584446476.697966029]: I heard: [hello world 3914]
[ INFO] [1584446476.798001891]: I heard: [hello world 3915]
[ INFO] [1584446476.897991023]: I heard: [hello world 3916]
[ INFO] [1584446476.998118490]: I heard: [hello world 3917]
[ INFO] [1584446477.098303838]: I heard: [hello world 3918]
[ INFO] [1584446477.199037078]: I heard: [hello world 3919]
[ INFO] [1584446477.298292505]: I heard: [hello world 3920]
[ INFO] [1584446484.637563102]: I heard: [TFM Ester]
```

El propio *listener* que ya había lanzado, escucha el mensaje que se ha publicado.

11.5.4 Creación de paquetes

Para crear un paquete es necesario indicar en el comando *‘catkin_create_*pkg’ el nombre del paquete y las distintas dependencias, es decir tendría la siguiente estructura:

```
$catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

Con argumentos quedaría algo así:

```
$catkin_create_pkg paquete std_msgs rospy roscpp
```

Figura 105. Interior de un paquete creado.

11.5.5 Uso de eclipse

Comando para abrir los ficheros en el IDE de eclipse (hay que ejecutarlo en un terminal en el directorio del espacio de trabajo):

```
$catkin_make –force-cmake –G”Eclipse CDT4 – Unix MakeFiles”
```

Para abrir los proyectos, una vez en eclipse se selecciona la opción de importar nuevo proyecto, a continuación, proyecto existente en el *workspace* y se selecciona la carpeta de *build*, en lugar de la carpeta *src*. Una vez dentro de eclipse y con la carpeta *build* importada se abre *source directory* (*src*) para editar los archivos .h y los .cpp, así como el ‘*CMakeLists.txt*’, en caso de editarlo. Los paquetes también pueden ser compilados desde eclipse, así como desde el *catkin_ws*, como se ha descrito anteriomente.

11.5.6 Lanzar un robot

Comando para lanzar el robot instalado en gazebo.

```
$roslaunch husky_gazebo husky_empty_world.launch
```

11.5.7 Instalación OPEN62541
Descargar el paquete:

```
$git clone https://github.com/open62541/open62541.git
```

Antes de hacer el *‘make’* para ejecutar el paquete descargado es necesario preparar el computador:

Una vez hecho esto, se puede proceder a instalar el paquete usando el '*cmake*'.

```
$git submodule update --init --recursive
$mkdir build && cd build
$cmake -DBUILD_SHARED_LIBS=ON -
DCMAKE_BUILD_TYPE=RelWithDebInfo -$DUA_NAMESPACE_ZERO=FULL –
DUA_ENABLE_AMALGAMATION=ON ..
$make
$sudo make install
```

Para crear un proyecto e implementar en ellos un servidor y/o un cliente OPC-UA, siempre y cuando se copie en la carpeta del proyecto los siguientes archivos:

```
$sudo apt-get install git build-essential gcc pkg-config cmake
Python
# habiltar características adicionales
$sudo apt-get install cmake-curses-gui #interfaz gráfica ccmak
$sudo apt-get install libmbedtls-dev #apoyo encriptación
$sudo apt-get install check libsubunit-dev #unit tests
$sudo apt-get install python-sphinx graphviz #generación de
documentación
$sudo apt-get install python-sphinx-rtd-theme #estilo
documentación
```

```
$ cp ~/opc_ua/projects/open62541.* .
```

11.5.8 Instalación de un servidor OPC-UA

Se crea un fichero .c al que se denominará *myServer.c* con el siguiente código:

```c
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>
#include <open62541/server_config_default.h>

#include <signal.h>
#include <stdlib.h>

static volatile UA_Boolean running = true;
static void stopHandler(int sig) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "received ctrl-c");
    running = false;
}

int main(void) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_Server *server = UA_Server_new();
    UA_ServerConfig_setDefault(UA_Server_getConfig(server));

    UA_StatusCode retval = UA_Server_run(server, &running);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

La inclusión de las librerías:

```c
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>
#include <open62541/server_config_default.h>
```

Si esto da problemas, se pueden sustituir esas tres librerías por una única librería que es:

```c
#include <open62541.h>
```

Para compilar el código:

```
$gcc -std=c99 open62541.c myServer.c -o myServer
```

Y para ejecutarlo:

```
$./myServer
```

Con ello se obtiene un servidor que está escuchando en la dirección *opc.tcp://localhost:4840*.

Figura 106. Servidor OPC-UA escuchando en *localhost:4840*

Ç

El código que se ha usado para este ejemplo muestra las tres partes del ciclo de vida de un servidor: 'crear el servidor', 'ejecutar el servidor' y 'borrar el servidor'. Tanto la creación como la eliminación del servidor son bastante triviales una vez que la configuración ya está hecha. Para que comience la ejecución del servidor se usa el método '*UA_Server_run'* al que se le pasan dos parámetros, el servidor que se quiere iniciar y un dato llamado *'running'* que es un boolenao especial de OPC-UA. Internamente el servidor usa '*Timeouts*' para programar la ejecución de sus tareas, los intervalos que ocurren entre estos '*Timeouts*' son usado por el servidor para la escucha y recepción de posibles mensajes.

Para que el servidor pare de ejecutarse es necesario que la variable *running* cambie su valor a '*false*'. Para ello se usa el método '*stophandler*', que consiste en usar un manejador de señal para atrapar la señal *crtl-c*, que tradicionalmente se usa para abortar procesos. Este método atrapa la señal, y cambia el valor de *running* a ***false***, lo que provoca el apagado del servidor.

11.5.9 Instalación de un cliente Prosys

Para instalar un cliente Prosys hay que descargarlo de la página para *Linux*. Su instalación se hace con la instrucción:

```
$sudo dpkg -i prosys-opc-ua-client-3.2.0-328.deb
```

11.5.10 Cambio de Python 2.7 a Python 3.6

Para poder realizar este cambio, es necesario crear un nuevo directorio de catkin, puesto que gran parte de los comandos de ROS *Kinetic* usan el compilador *Python 2.7* y crearlo en el mismo directorio donde ya se ha estado trabajando provacaría interfenrecias. Para ello se crea en el home un nuevo espacio de trabajo:

También es necesario instalar *Python 3.6*, esto se puede hacer:

```
$mkdir catkin_build_ws
```

```
$sudo apt-get install software-properties-common python-
software-properties
$sudo apt-get update
$sudo apt-get install python3.6
```

A continuación es necesario compilar el directorio, para ello, en lugar de usar la habitual *catkin_make*, se usará *catkin build*, pero antes:

```
$cd catkin_build_ws
$catkin config -DPYTHON_EXECUTABLE=/usr/bin/python3 -
DPYTHON_INCLUDE_DIR=/usr/include/python3.6m -
DPYTHON_LIBRARY=/usr/lib/x86_64-linux-gnu/libpython3.6m.so
$catkin config –install
$catkin build
```

11.5.11 Instalación y configuración de Python 3.6

Para que la compilación del proyecto se resuelva sin problemas también es necesario añadir en la carpeta *interfaz_py*, un archivo de configuración, denomidado *setup.py*, donde se indica el nombre de la carpeta donde están los *scripts* de *Python* que contendrá el siguiente código:

```
## ! DO NOT MANUALLY INVOKE THIS setup.py, USE CATKIN INSTEAD

from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

# fetch values from package.xml
setup_args = generate_distutils_setup(
    packages=['scripts'],
)

setup(**setup_args)
```

Tal como se indica en el comentario del archivo, la forma de invocarlo es mediante el *CMakeLists.txt*, el cuál tendrá el siguiente aspecto:

```
cmake_minimum_required(VERSION 3.0.2)
project(interfaz_py)


find_package(catkin REQUIRED COMPONENTS message_generation rospy)


#invoke setup file
catkin_python_setup()

## Generate added messages and services with any dependencies
generate_messages()

catkin_package(
  CATKIN_DEPENDS message_runtime
)
```

Por último en el archivo *package.xml*, es necesario añadir:

```
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>message_runtime</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>message_generation</exec_depend>
  <exec_depend>message_runtime</exec_depend>
```

Para poder crear los *scripts* de *Python* desde la carpeta *scripts* que se ha habilitado para ello, se abre el proyecto *interfaz_py* desde la aplicación '*Visual*

Studio Code'. Se crea dentro de esa carpeta un archivo con extensión *.py*. La primera línea de todos los fichero de código *Python* debe indicar la ruta al compilador de código, en este caso es:

```
#!/usr/bin/python3.6
```

Es importante añadir en el *CmakeLists.txt*, la ruta del archivo para que el proyecto se compile de manera adecuado, por lo que se añade:

```
catkin_install_python(PROGRAMS scripts/GUI.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
```

A diferencia de *C++*, los fichero de código de *Python* no hace falta compilarlos con todo el proyecto con el comando *catkin build*, si no que simplemente se guarda el fichero y se ejecuta directamente desde la propia carpeta de *scripts*, con el siguiente comando:

```
$python3 GUI.py
```